Confidential Computing Deployment Guide - (AMD SEV-SNP & 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	2/23/2024	Rob Nertney	GA Version Release

Table of Contents

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
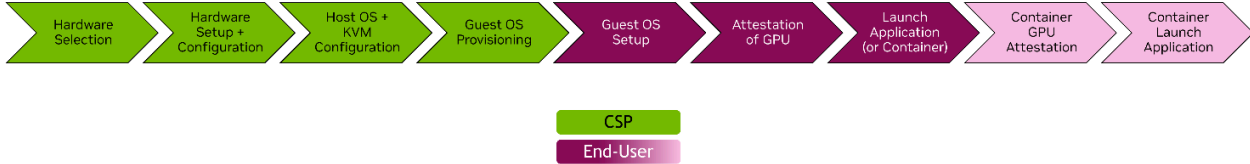
In this example, the CSP does not require a policy for how often the GPU must be checked for integrity/validity, nor does it need to consider the infrastructure requirements for Confidential Containers. The user/tenant of the CSP does not need to consider the steps required to configure the GPU for confidential 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
AMD	5.19	KVM	5.19	

Hardware IT Administrator

Figure 3.   The Hardware IT Administrator Persona
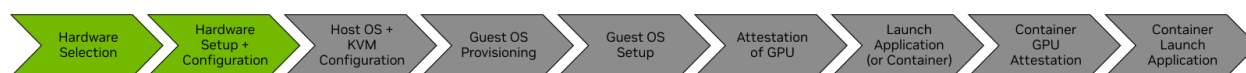
 The Hardware IT Administrator persona is at the beginning of the CC chain and attention needs to be paid to selecting your CPU and GPU. This persona should contain **System Architects** and **IT Administrators**, selects the correct part numbers, and configures the BIOS/UEFI for the subsequent steps.

Selecting Hardware

CC requires CPUs and GPUs with specific functionality that enable the security outlined by the CC Consortium.
- CPU Requirements
  - AMD with SEV-SNP support
- GPU Requirements
  - NVIDIA Hopper H100 GPU
- Other Recommendations
  - Your motherboard vendor can be configured with Secure Boot and SNP 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

ASRockRack BIOS Version L3.12C

```
Advanced -->
     AMD CBS ->
          CPU Common ->
                    SEV ASID Count -> 509 ASIDs
                    SEV-ES ASID space Limit Control -> Manual
                    SEV-ES ASID space limit -> 100
                    SNP Memory Coverage -> Enabled
                    SMEE -> Enabled
          NBIO common ->
                    SEV-SNP Support -> Enabled
```

Supermicro System: BIOS Firmware Version 2.5

```
Advanced -->
     CPU Configuration -->
             SMEE -> Enabled
              SEV ASID Count -> 509 ASIDs
             SEV-ES ASID Space Limit Control -> Manual
             SEV-ES ASID Space Limit -> 100
             SNP Memory Coverage -> Enabled

     NB Configuration ->
             IOMMU -> Enabled
             SEV-SNP support -> Enabled
```

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

Host OS Administrator

Figure 4.   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 -y ninja-build iasl nasm  flex bison openssl dkms autoconf
zlib1g-dev python3-pip libncurses-dev libssl-dev libelf-dev libudev-dev libpci-dev
libiberty-dev libtool libsdl-console libsdl-console-dev libpango1.0-dev libjpeg8-dev
libpixman-1-dev libcairo2-dev  libgif-dev libglib2.0-dev git-lfs jq

$ sudo pip3 install numpy flex bison
```

We will now build a 5.19-based kernel with SEV-SNP 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 SEV-SNP guests.

> **Note**: Here is some important information:
>
> - The AMD SEV-SNP tree is continually evolving in sync with the kernel version.
> - The only supported AMD SEV SNP branch for HCC use with KVM is the [sev-snp-devel](#) branch.
> - For GPU support, you will need to apply two patches to the 5.19-rc6 kernel.

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

```
$ cd /shared/
$ git clone https://github.com/AMDESE/AMDSEV.git
$ git clone https://github.com/NVIDIA/nvtrust.git
$ cd AMDSEV
$ git checkout sev-snp-devel
```

Preparing to Build the Kernel

NVIDIA's driver requires that LKCA be enabled in the kernel. It is not enabled by default in the AMDESE package, so you need to edit `/shared/AMDSEV/common.sh` to include the following lines **after** line 81:

```
run_cmd ./scripts/config --enable  CONFIG_CRYPTO_ECC
run_cmd ./scripts/config --enable  CONFIG_CRYPTO_ECDH
run_cmd ./scripts/config --enable  CONFIG_CRYPTO_ECDSA
run_cmd ./scripts/config --enable  CONFIG_CGROUP_MISC
```

Now, we can run the script that will fetch and build the required components:

```
$ ./build.sh --package
```

Modifying the Kernel

At this point, you have built the unpatched 5.19 from the `sev-snp-devel` branch Unfortunately, this code tree could not be patched before the build because the AMD-SNP repository automates many of these steps.

1. Now, we can patch the code and build the kernel required for the H100:

   ```
   # copy the patch files from /shared/nvtrust into /shared/AMDSEV
   $ cp /shared/nvtrust/infrastructure/patches/linux/*.patch /shared/AMDSEV
   ```

```
$ pushd /shared/AMDSEV/linux/host
$ patch -p1 -l < ../../iommu_pagefault.patch
$ patch -p1 -l < ../../iommu_pagesize.patch
$ popd
```

2. Rebuild and package the linux files required:
    ```
    $ ./build.sh --package
    ```

Installing the Host OS

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

```
$ sudo cp kvm.conf /etc/modprobe.d/

# The following folder will be appended with the build date
$ cd /shared/AMDSEV/snp-release-<DATE>
$ sudo ./install.sh
#GRUB should automatically use the new linux image.

#reboot the host:
$ sudo reboot
```

Validating the Host Detects SNP

1. After the host reboots, to check that our kernel is the new SNP-aware version, and that
    our configuration options were correctly applied, run the following commands.
    ```
    $ uname -a
    Linux hostname1 5.19.0-rc6-snp-host-d9bd54fea4d2 #3 SMP Wed Nov 30 10:23:09 UTC
    2022 x86_64 x86_64 x86_64 GNU/Linux
    ```

    The dates and hashes above might vary slightly. The important thing is to see `snp` in the
    string.
2. Validate the kernel that was configured with the proper CC options. This is done by
    reviewing the `/boot/config-5.19.0-rc6-snp-host<...>`.
    ```
    $ grep CONFIG_CRYPTO_EC /boot/config-5.19.0-rc6-snp-host<...>
    CONFIG_CRYPTO_ECC=y
    CONFIG_CRYPTO_ECDH=y
    CONFIG_CRYPTO_ECDSA=y
    ```

3. Ensure that the kernel actually detects the SEV-SNP processor.
    If you do not see this correct output, then please review the *Hardware IT Administrator*
    section above, or contact your IT Administrator and have them review that section.
    ```
    $ sudo dmesg | grep -i -e rmp -e sev
    ```

```
[    0.768000] SEV-SNP: RMP table physical address 0x0000000035600000 -
0x0000000075bfffff
[    3.868558] ccp 0000:45:00.1: sev enabled
[    3.918694] ccp 0000:45:00.1: SEV firmware update successful
[    7.315402] ccp 0000:45:00.1: SEV API:1.51 build:3
[    7.315410] ccp 0000:45:00.1: SEV-SNP API:1.51 build:3
[    7.322019] SEV supported: 410 ASIDs
[    7.322019] SEV-ES and SEV-SNP supported: 99 ASIDs
```

(**Optional**) Upgrade out-of-date SEV Firmware (<1.51)

In the command above, you might notice that the output is like the following output. This means your SEV firmware is out of date and needs to be updated.

```
$ sudo dmesg | grep -i -e rmp -e sev

[    0.564845] SEV-SNP: RMP table physical address 0x0000000088900000 -
0x00000000a8efffff
[    3.257600] ccp 0000:45:00.1: sev enabled
[    3.274785] ccp 0000:45:00.1: SEV-SNP support requires firmware version >= 1:51
[    3.284535] ccp 0000:45:00.1: SEV: failed to INIT error 0x1, rc -5
[    3.284541] ccp 0000:45:00.1: SEV API:1.49 build:3
[    3.424129] SEV supported: 410 ASIDs
[    3.424130] SEV-ES and SEV-SNP supported: 99 ASIDs
```

SEV-SNP support requires a firmware version that is later than version 1.51:1. The latest SEV-SNP firmware is available on https://developer.amd.com/sev and through the linux-firmware project.

The following steps document the firmware upgrade process for the latest SEV-SNP firmware on https://developer.amd.com/sev at the time this was written. A similar procedure can also be used for newer firmware:

1. Run the following commands to reboot your system.

```
$ wget
https://developer.amd.com/wp-content/resources/amd_sev_fam19h_model0xh_1.33.03.zip

$ unzip amd_sev_fam19h_model0xh_1.33.03.zip

$ sudo mkdir -p /lib/firmware/amd
$ sudo cp amd_sev_fam19h_model0xh_1.33.03.sbin
/lib/firmware/amd/amd_sev_fam19h_model0xh.sbin

$ sudo reboot
```

2. After your system reboots, you should see correct messages in the `dmesg` output.

```
$ sudo dmesg | grep -i -e rmp -e sev
[   0.768000] SEV-SNP: RMP table physical address 0x0000000035600000 -
0x0000000075bfffff
[   3.868558] ccp 0000:45:00.1: sev enabled
[   3.918694] ccp 0000:45:00.1: SEV firmware update successful
[   7.315402] ccp 0000:45:00.1: SEV API:1.51 build:3
[   7.315410] ccp 0000:45:00.1: SEV-SNP API:1.51 build:3
[   7.322019] SEV supported: 410 ASIDs
[   7.322019] SEV-ES and SEV-SNP supported: 99 ASIDs
```

3. Run the following commands to do a final check for SNP support.

```
# All outputs should be "Y"
$ cat /sys/module/kvm_amd/parameters/sev
Y
$ cat /sys/module/kvm_amd/parameters/sev_es
Y
$ cat /sys/module/kvm_amd/parameters/sev_snp
Y
```

Your system is now configured for CC modes. To use these features, launch a VM.

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/nvtrust/host_tools/python
$ git clone https://github.com/nvidia/gpu-admin-tools
$ cd gpu-admin-tools

$ 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 onVMs with KVM, you can skip
the next section.

(**Optional**) Guest VM Setup

This section provides instructions about how to build and configure a basic Guest VM that
is CC enlightened.

Create the VM Base Image

1. Run the following commands to obtain an ISO of a supported operating system.

In this example, we are using Ubuntu 22.04.

```
$ cd /shared/nvtrust/host_tools/sample_kvm_scripts/isos
#download an ISO of a supported OS here, for example:

$ wget https://releases.ubuntu.com/22.04.4/ubuntu-22.04.4-live-server-amd64.iso
```

2. Create a blank VM drive, which is one file that acts as the VM's storage drive.

```
# Create the empty Virtual Disk Drive. Change 500G to your desired size

$ /shared/AMDSEV/snp-release-<DATE>/usr/local/bin/qemu-img create -f qcow2
/shared/nvtrust/host_tools/sample_kvm_scripts/images/ubuntu22.04.qcow2 500G
```

Installing the Guest OS

1. To confirm some of the parameters in the top of `prepare_vm.sh` to align with your VM location, run the following commands.

```
# Edit /shared/nvtrust/host_tools/sample_kvm_scripts/prepare_vm.sh
#
# Modify Variables to point to your correct paths for:
# AMD_SEV_DIR   -> /shared/AMDSEV/snp-release-<DATE>
# VDD_IMAGE -> /shared/nvtrust/host_tools/sample_kvm_scripts/images/<your qcow2
drive file>
# ISO       -> /shared/nvtrust/host_tools/sample_kvm_scripts/isos/<ubuntu22.04>.iso
```

2. After the file points to the correct paths for appropriate items, run the following command to launch the VM.

```
$ sudo ./prepare_vm.sh
```

Modifying GRUB to Print the Kernel Launch to the TTY

1. Select install option "Try or Install Ubuntu Server"

Figure 5.   Selecting an Installation Option


```
                        GNU GRUB  version 2.06

 ┌──────────────────────────────────────────────────────────────────────┐
 │*Try or Install Ubuntu Server                                         ▓│
 │ Ubuntu Server with the HWE kernel                                     │
 │ Test memory                                                           │
 │                                                                       │
 │                                                                       │
 │                                                                       │
 │                                                                       │
 │                                                                       │
 │                                                                       │
 │                                                                       │
 └──────────────────────────────────────────────────────────────────────┘

        Use the ↑ and ↓ keys to select which entry is highlighted.
        Press enter to boot the selected OS, `e' to edit the commands
        before booting or `c' for a command-line.
```

2. To edit the command, press **e**.

Figure 6.   Editing the Command


```
                    GNU GRUB   version 2.06

 setparams 'Try or Install Ubuntu Server'

         set gfxpayload=keep
         linux           /casper/vmlinuz  ---
         initrd          /casper/initrd




     Minimum Emacs-like screen editing is supported. TAB lists
     completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for
     a command-line or ESC to discard edits and return to the GRUB menu.
```

3. To modify the Linux launch and print to the local console, edit the following command.

```
linux          /casper/hwe-vmlinuz  console=ttyS0 ---
```

Figure 7.   Printing the Local Console

```
                         GNU GRUB   version 2.06

 ┌──────────────────────────────────────────────────────────────────────┐
 │setparams 'Try or Install Ubuntu Server'                                │
 │                                                                        │
 │        set gfxpayload=keep                                             │
 │        linux         /casper/vmlinuz  console-ttyS0 ▋──                │
 │        initrd        /casper/initrd                                    │
 │                                                                        │
 │                                                                        │
 │                                                                        │
 │                                                                        │
 │                                                                        │
 │                                                                        │
 │                                                                        │
 │                                                                        │
 └──────────────────────────────────────────────────────────────────────┘

        Minimum Emacs-like screen editing is supported. TAB lists
        completions. Press Ctrl-x or F10 to boot, Ctrl-c or F2 for
        a command-line or ESC to discard edits and return to the GRUB menu.
```

4. To continue the launch, press CTRL+X.

You can now configure the Guest OS install parameters. No specific options during this install are required. After the Guest OS is installed, Ubuntu will prompt you to reboot, and the VM will terminate, which returns you to the host.

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

Update the Guest Kernel

Restart your Guest VM:

```
# On the Host
$ sudo ./launch_vm.sh -x
```

Copy over the SNP-aware guest kernel from the host into the guest.

```
# On the Host (replace guestUser with the username you created in the previous steps)
```

```
$ scp -P9899 /shared/AMDSEV/snp-release-<DATE>/linux/guest/* guestUser@localhost:~/
```

Log into your CVM.
```
hostUser@host:~$ ssh -p9899 guestUser@localhost
guestUser@localhost's password:
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.19.0-43-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Wed Jun  7 10:00:56 PM UTC 2023

  System load:               0.080078125
  Usage of /:                5.5% of 982.19GB
  Memory usage:              0%
  Swap usage:                0%
  Processes:                 220
  Users logged in:           0
  IPv4 address for docker0: 172.17.0.1
  IPv4 address for enp0s3:  10.0.2.15
  IPv6 address for enp0s3:  fec0::5054:ff:fe12:3456


 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

      https://ubuntu.com/pro

Expanded Security Maintenance for Applications is not enabled.

57 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Wed Jun  7 21:46:48 2023
guestUser@guestVM:~$
```

Install the kernel
```
# On the Guest
$ sudo apt install ~/*.deb
```

```
$ sudo reboot
```

Relaunch the VM

```
$ sudo ./launch_vm.sh
```

Validate that the Guest is in CVM mode:

```
$ sudo dmesg | grep -i sev
[sudo] password for user:
[    1.063740] Memory Encryption Features active: AMD SEV SEV-ES SEV-SNP
[    1.248095] SEV: Using SNP CPUID table, 31 entries present.
[    6.632952] SEV: SNP guest platform device initialized.
```

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"
   ```

> **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.

Launching the Guest OS

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!

We have provided the launch_vm.sh script to help with this task. Again, it will need to be modified to match your system's folder configuration.

```
# Modify /shared/nvtrust/host_tools/sample_kvm_scripts/launch_vm.sh
#
# Modify Variables to point to your correct paths for:
# AMD_SEV_DIR   -> /shared/nvtrust/AMDSEV/snp-release-<DATE>
# VDD_IMAGE     -> /shared/nvtrust/host_tools/sample_kvm_scripts/images/<your qcow2
drive file>
# NVIDIA_GPU    -> the slot ID found in the steps above
# MEM           -> specifies amount of RAM the guest
# FWDPORT       -> specifies which port will forward to the sshd on this machine
```

After the modifications are complete, you can launch your Confidential VM:

```
$ sudo ./launch_vm.sh
```

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.

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:

```
# Obtain the NVIDIA keys to download the CUDA Toolkit
$ wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-keyrin
g_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
    VBIOS Version                    : 96.00.5E.00.25
```

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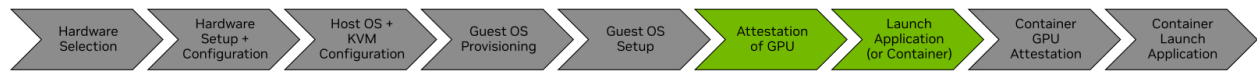
The Virtual Machine user might (or might not) be the administrator of the system (refer to [Virtual Machine Administrator](#) for more information). This Persona assumes that the system is correctly configured for CC.

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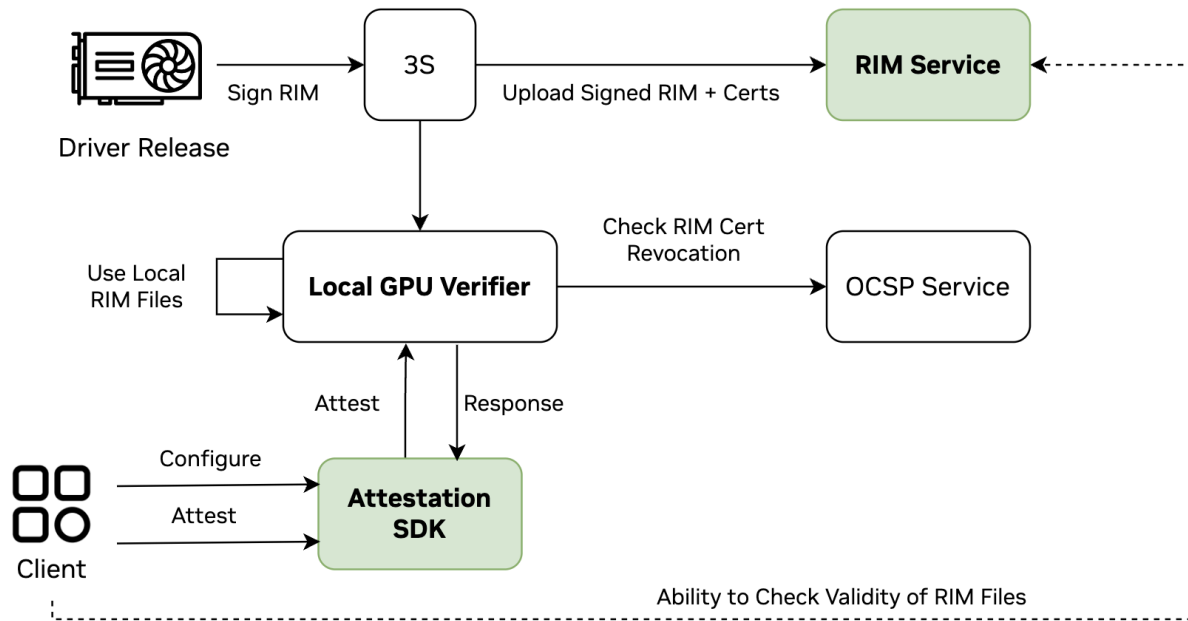
The recommended flow for attestation is to directly use the Attestation SDK and its APIs . However, you can directly call the Local GPU Verifier. This flow to learn more about the Local GPU Verifier, refer to the *NVIDIA Attestation SDK* guide at https://docs.nvidia.com/nvtrust

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/SmallGPUTest.py`
However, here is the Python code, and you can run this code on the `python3` command line.

```python
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}}'

# Run Attest
print(client.attest())

# Call validate_token to validate the results against the Appraisal policy for
Attestation Results
print(client.validate_token(attestation_results_policy))
```

The primary focus of the attestation should be the yellow highlighted variable. As the developer, you can decide which measurements 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 measurements 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!

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