Confidential Computing Deployment Guide

DU-11462-001_2.0 | August 2023

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

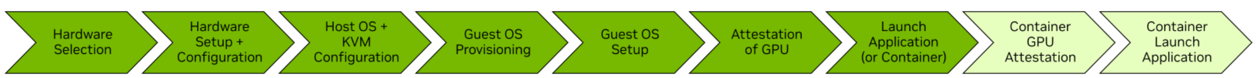
Table of Contents

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.

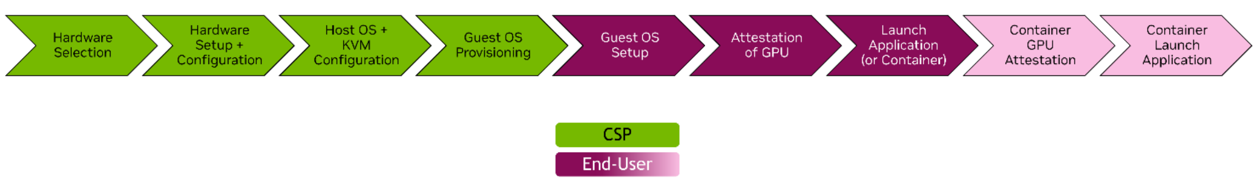
This guide is intended to provide instructions that are targeted to various personas who want to use Hopper Confidential Compute (HCC). These personas are rough definitions of individuals who might have different responsibilities in the overall confidential system. The overall flow of using one is illustrated 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`:

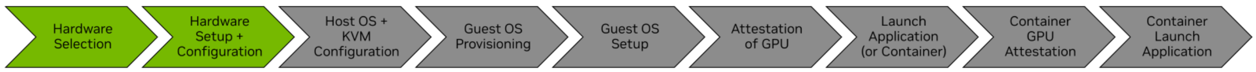
```
$ shell-command-A

$ 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.

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
  - Intel with TDX support
- GPU Requirements
  - NVIDIA Hopper H100 GPU
- Other Recommendations
  - Your motherboard vendor can be configured with Secure Boot and SNP/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 (AMD SEV-SNP)

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.

Setting Up the Hardware Setup and Configuring Your System (Intel TDX)

Supermicro System: BIOS Firmware Version 2.1

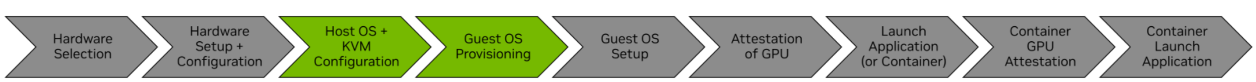
```
CPU Configuration -->
      Processor Configuration -->
            Limit CPU PA to 46 Bits -> Disable
            Intel TME, Intel TME-MT, Intel TDX -->
                  Total Memory Encryption (Intel TME) -> Enable
                  Total Memory Encryption (Intel TME) Bypass -> Auto
                  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

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**.

Currently, the supported Host OSes are

- **AMD - Ubuntu 22.04 with Kernel 5.19**.

- **Intel - Ubuntu 22.04 with Kernel 6.2**

Setting Up the Host OS (AMD SEV-SNP)

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/linux/patches/*.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.

Setting Up the Host OS (Intel TDX)

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

$ sudo pip3 install numpy flex bison
```

We will now build a 6.12-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 SEV-SNP 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 the2023ww15 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

```
$ cd /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
```

Build the Kernel

Rebuild and package the linux files required:

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

Installing the Host OS

Run the following commands to install the SNP-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
```

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 localhost 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

$ 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: atributes 0x0, vendor_id 0x8086, major_version 1,
minor_version 0, build_date 20220420, build_num 342
[   71.635893] tdx: additional output: rcx 0x0, rdx 0xa00000300000008, 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=/boot/vmlinuz-5.19.0-mvp12v2+2-generic
root=UUID=8cf9209a-f59b-480c-b2c3-84e5067229a2 ro numa_balancing=disable console=tty0
console=ttyS0,115200n8 crashkernel=1G-:0M

# Run the test to validate the host
$ cd /shared/tdx-tools/utils$ sudo ./check-tdx-host.sh
```

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

# Query the current state CC state:
$ sudo python3 gpu_cc_tool.py --gpu-name=H100 --query-cc-settings
NVIDIA GPU Tools version %VERSION%
Topo:
  PCI 0000:c0:03.1 0x1022:0x1483
   PCI 0000:c1:00.0 0x15b3:0x1979
    PCI 0000:c2:02.0 0x15b3:0x1979
     PCI 0000:c4:00.0 0x15b3:0x1979
      PCI 0000:c5:00.0 0x15b3:0x1979
       GPU 0000:c6:00.0 H100-PCIE 0x2313 BAR0 0x11042000000

# To change the state:
$ sudo python3 gpu_cc_tool.py --gpu-name=H100 --set-cc-mode devtools
--reset-after-cc-mode-switch
NVIDIA GPU Tools version %VERSION%
Topo:
  PCI 0000:c0:03.1 0x1022:0x1483
```

```
   PCI 0000:c1:00.0 0x15b3:0x1979
    PCI 0000:c2:02.0 0x15b3:0x1979
     PCI 0000:c4:00.0 0x15b3:0x1979
      PCI 0000:c5:00.0 0x15b3:0x1979
       GPU 0000:c6:00.0 H100-PCIE 0x2313 BAR0 0x11042000000

# Query the state again:
$ sudo python3 gpu_cc_tool.py --gpu-name=H100 --query-cc-settings
NVIDIA GPU Tools version %VERSION%
Topo:
  PCI 0000:c0:03.1 0x1022:0x1483
   PCI 0000:c1:00.0 0x15b3:0x1979
    PCI 0000:c2:02.0 0x15b3:0x1979
     PCI 0000:c4:00.0 0x15b3:0x1979
      PCI 0000:c5:00.0 0x15b3:0x1979
       GPU 0000:c6:00.0 H100-PCIE 0x2313 BAR0 0x11042000000
WARNING:root:GPU 0000:c6:00.0 H100-PCIE 0x2313 BAR0 0x11042000000 has CC mode
devtools, some functionality may not work
```

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 (AMD SEV-SNP)

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/22.04.2/ubuntu-22.04.2-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
```

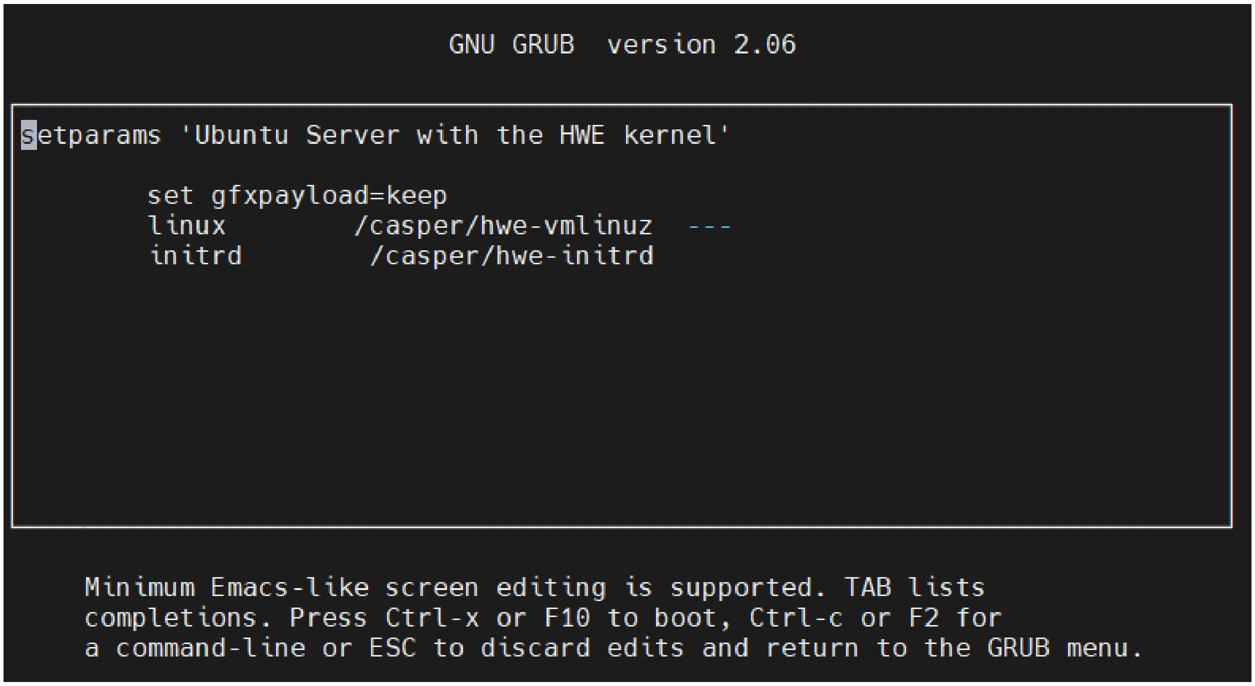
Modifying GRUB to Print the Kernel Launch to the TTY

1. Select an install option

Figure 5. Selecting an Installation Option

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

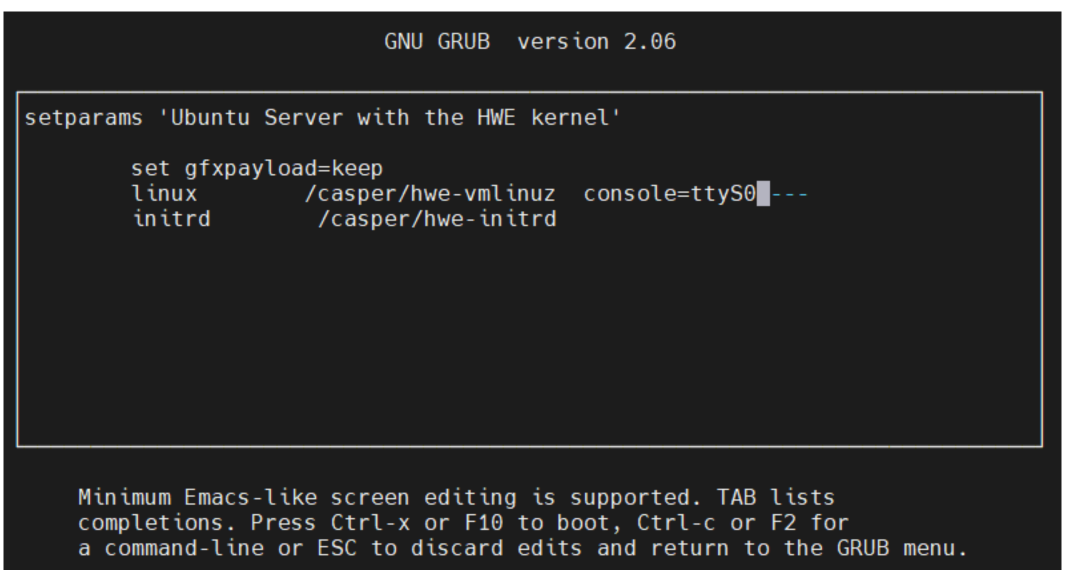
Figure 6. Editing the Command



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



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.

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
```

(**Optional**) Guest VM Setup (Intel TDX)

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.

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.

```
$ cd /shared/tdx-tools/build/ubuntu-22.04/guest-image
$ ./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
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
vfio-pci,host=81:00.0,bus=pci.1 -fw_cfg name=opt/ovmf/X-PciMmio64
```

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

```
$ cd /shared/tdx-tools/
$ ./start-qemu.sh -i 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:

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

Validating the TDX Guest:

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

```
$ ssh -p10026 root@localhost
passwd: 123456

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

nvidia@tdx-guest:~$ sudo dmesg | grep -i tdx root@ubuntu:~# 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@viking-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

# Note: We have added noccfilter in kernel commandline, it disable Confidential
computing (CC) guest filter support.
$ cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-6.2.0-mvp10v1+8-generic
root=UUID=4a01875a-75bb-4778-b1a6-dd6c5f270fc1 ro noccfilter console=tty1
console=ttyS0
```

(Known Issue) Cannot Change the CC State after a Guest OS Shutdown

There is an issue where after the Guest OS has shut down, the Host cannot change the CC state, and sees errors in the CC mode changing tool related to timeouts.

Workarounds

There are two workarounds for this issue:

Option 1

Unbind the device ID from the VFIO before setting the state and then rebind.

a. Start the Host OS.
b. (Initialization only) Create a VFIO bind for device IDs.
```
sudo sh -c "echo 10de 2336 > /sys/bus/pci/drivers/vfio-pci/new_id"
```
c. Start the Guest OS.
d. Do work on the Guest OS.
e. Shutdown the Guest OS.
f. Unbind the device (Slot ID) from the VFIO.
```
sudo sh -c "echo 0000:81:00.0 >
/sys/bus/pci/drivers/vfio-pci/unbind"
```
g. Change the CC state with the `gpu_cc_ tool.py` file.
h. Rebind the device (Slot ID) to the VFIO.
```
sudo sh -c "echo 0000:81:00.0 > /sys/bus/pci/drivers/vfio-pci/bind"
```
i. Repeat step c.

Option 2

Modify the boot params to disable the PCI idle.

a. Start the Host OS.
b. Edit `/etc/default/grub.`
c. Add `"vfio-pci.disable_idle_d3=1"` to the `GRUB_CMDLINE_LINUX_DEFAULT` line.
d. From the Host's command line, run the following command.
```
sudo update-grub
```
e. Reboot the Host OS.

In Option 2, you do not need to complete the unbind/bind, but this does prevent the GPU from going into a low-power mode.

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 -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:~$
```

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 runfile method of installing the NVIDIA drivers, because this has the *OpenRM* flavor as an option. OpenRM is the open-source version of our Kernel drivers, and the source can be found on our GitHub.

Hopper CC is enabled with CUDA 12.2 and is paired with driver r535-TRD2 (535.86.06), which can be downloaded from our website:

```
$ wget
https://developer.download.nvidia.com/compute/cuda/12.2.1/local_installers/cuda_12.2.1
_535.86.10_linux.run

# The runfile wants a few prerequisites
$ sudo apt install gcc g++ make

# Install the driver
$ sudo sh cuda_12.2.1_535.86.10_linux.run -m=kernel-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
    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
   ```

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**

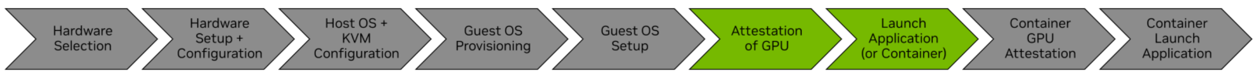
   ```
   $ nvidia-smi -q | grep VBIOS
       VBIOS Version                         : 96.00.5E.00.25
   ```

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.

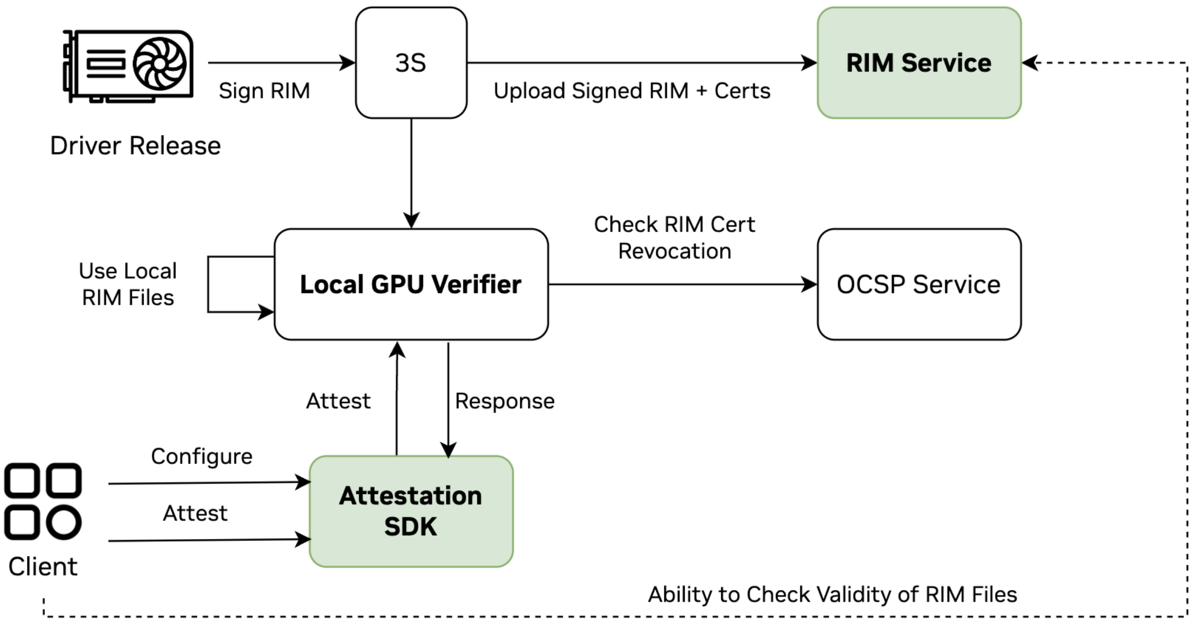
> **Note**: We recommend that you complete your work in the `/shared` folder in the guest VM.

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.

For the early access of Hopper CC, the method of attesting the GPU is by using our Attestation SDK with the local GPU verifier plugin. The SDK accepts various verifier plugins that will independently attest the various hardware modules.

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/confidential-computing

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.

```
$ cd /shared/nvtrust/guest_tools/gpu_verifiers/local_gpu_verifier
```

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).

```
$ cd /shared/nvtrust/guest_tools/attestation_sdk/
$ 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.

```
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

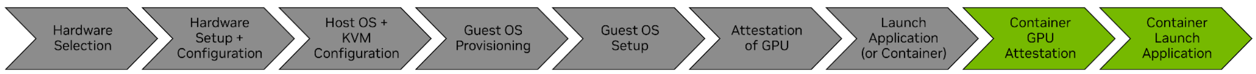
When the Attestation SDK has successfully returned a valid result, you should see something like below (varies slightly based on your specific system):

```
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)
```

Confidential Container User

Figure 11. Confidential Container User



Using containers is a deployment strategy that is frequently used for various computing workloads. They are used to provide deterministic, repeatable, and isolated environments and differ from VMs because containers share the host kernel on the system from which they are running, rather than having their own independent kernels. The trust boundary is the host (or VM) that is running the containers and not the containers themselves.

Kata was created to provide ultra lightweight VMs to bridge the gap between low-overhead, low-isolation containers and the high-isolation benefits of VMs. Microservices provided by confidential containers (CoCo) can run in these lightweight CVMs with the same security benefits as a traditional CVM.

Kata + CoCo, despite being a VM, has been built to be deployed and orchestrated by traditional Kubernetes (k8s) clusters. As a result, it is integrated into NVIDIA's GPU Operator.

Installing Container Runtime and Kubernetes

You can build a single-node k8s pod as a continuation of the flow of this document. It is only a representative sample, because building a full k8s cluster is beyond the scope of this document.

> **Note**: If you already have H100s and your own k8s cluster, follow the instructions in the GPU Operator Install Page.

1. Purge and remove any stray docker installs on your system and reinstall.

```
$ sudo apt remove docker docker-engine docker.io containerd runc
$ sudo apt -yqq install 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 update
$ sudo apt -yqq install docker-ce docker-ce-cli containerd.io docker-buildx-plugin
docker-compose-plugin

# Test our Docker Install
$ sudo docker run hello-world
Hello from Docker!
This message shows that your installation appears to be working correctly.
```

2. Configure and reload containerd.

```
$ sudo mkdir /etc/containerd
$ containerd config default | sudo tee /etc/containerd/config.toml
$ sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g'
/etc/containerd/config.toml

# Reload the container daemon
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now containerd
$ sudo systemctl restart containerd
$ sudo systemctl status containerd
```

3. Prepare the host for Kubernetes.

```
$ sudo apt install -yqq apt-transport-https ca-certificates curl
$ cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
$ sudo modprobe -a overlay br_netfilter
$ cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-iptables  = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward                 = 1
EOF
$ sudo sysctl --system
```

4. Install Kubernetes:

```
$ sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg \
   https://dl.k8s.io/apt/doc/apt-key.gpg
$ echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg]
https://apt.kubernetes.io/ kubernetes-xenial main" \
  | sudo tee /etc/apt/sources.list.d/kubernetes.list

$ sudo apt update
$ sudo apt install -yqq kubelet kubeadm kubectl
$ sudo apt-mark hold kubelet kubeadm kubectl

$ sudo swapoff -a
$ sudo sed -i -e '/swap/d' /etc/fstab

$ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
$ mkdir -p $HOME/.kube
$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

5. At this point, we are going to configure the system as a single-node deployment.
   a. Taint and label the single node to run workloads.

```
# The first command may fail depending on the Kubernetes version installed

# Remove the control plane label from this pod so that it can run jobs instead
of
# only controlling the cluster
$ kubectl taint nodes --all node-role.kubernetes.io/control-plane-

# Add the label to this node such that it is a worker, capable of accepting
jobs
$ kubectl label node --all node-role.kubernetes.io/worker="" --overwrite
$ kubectl label node --all node.kubernetes.io/worker="" --overwrite
```

   b. Provide a manifest that will configure networking for the cluster.

```
$ kubectl  apply -f
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flan
nel.yml
```

Install the Confidential Containers Operator

To configure the NVIDIA GPU Operator to operate on a node, label the system.

1. Get the name of your node.

```
$ kubectl get node
NAME        STATUS   ROLES                    AGE    VERSION
```

```
myServer    Ready    control-plane,worker   5d1h   v1.28.1
```

2. Apply the appropriate label to the node.

```
$ export VERSION=v0.7.0
```

3. Set the Operator version in an environment variable.

```
$ kubectl label node myServer nvidia.com/gpu.workload.config=vm-passthrough
```

4. Install the Operator.

```
$ kubectl apply -k
"github.com/confidential-containers/operator/config/release?ref=${VERSION}"

namespace/confidential-containers-system created
customresourcedefinition.apiextensions.k8s.io/ccruntimes.confidentialcontainers.org
created
serviceaccount/cc-operator-controller-manager created
role.rbac.authorization.k8s.io/cc-operator-leader-election-role created
clusterrole.rbac.authorization.k8s.io/cc-operator-manager-role created
clusterrole.rbac.authorization.k8s.io/cc-operator-metrics-reader created
clusterrole.rbac.authorization.k8s.io/cc-operator-proxy-role created
rolebinding.rbac.authorization.k8s.io/cc-operator-leader-election-rolebinding
created
clusterrolebinding.rbac.authorization.k8s.io/cc-operator-manager-rolebinding
created
clusterrolebinding.rbac.authorization.k8s.io/cc-operator-proxy-rolebinding created
configmap/cc-operator-manager-config created
service/cc-operator-controller-manager-metrics-service created
deployment.apps/cc-operator-controller-manager create
```

5. In the confidential-containers-system namespace, view the pods and services.

```
$ kubectl get pod,svc -n confidential-containers-system

NAME                                                 READY   STATUS    RESTARTS
AGE
pod/cc-operator-controller-manager-c98c4ff74-ksb4q   2/2     Running   0
2m59s

NAME                                                     TYPE        CLUSTER-IP
EXTERNAL-IP   PORT(S)    AGE
service/cc-operator-controller-manager-metrics-service   ClusterIP   10.98.221.141
<none>        8443/TCP   2m59s
```

6. Install the sample Confidential Containers runtime by creating the manifests and then edit the node selector so that the runtime is installed only on the labeled nodes.

```
$ kubectl apply --dry-run=client -o yaml \
    -k
"github.com/confidential-containers/operator/config/samples/ccruntime/default?ref=$
{VERSION}" > ccruntime.yaml
```

7. Edit the ccruntime.yaml file and set the node selector.

```
apiVersion: confidentialcontainers.org/v1beta1
kind: CcRuntime
metadata:
  annotations:
...
spec:
  ccNodeSelector:
    matchLabels:
      nvidia.com/gpu.workload.config: "vm-passthrough"
...
```

8. Apply the modified manifests.

```
$ kubectl apply -f ccruntime.yaml
ccruntime.confidentialcontainers.org/ccruntime-sample created
```

9. Wait for a few minutes for the Operator to create the base runtime classes.

```
$ kubectl get runtimeclass
```

> **Note**: It will take a few **minutes** for the runtime classes to be created. If the above function returns nothing, wait for a little longer until the following output below is displayed.

Here are the runtime statuses:

```
NAME            HANDLER         AGE
kata            kata            13m
kata-clh        kata-clh        13m
kata-clh-tdx    kata-clh-tdx    13m
kata-qemu       kata-qemu       13m
kata-qemu-sev   kata-qemu-sev   13m
kata-qemu-snp   kata-qemu-snp   13m
kata-qemu-tdx   kata-qemu-tdx   13m
```

Install NVIDIA GPU Operator

(Optional) Install Helm

The NVIDIA GPU Operator requires Helm for its install process.

1. To install helm installed, run the following commands.

```
# Get the stable version of helm

$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3

$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

2. Add and update the NVIDIA Helm repository.

```
helm repo add nvidia https://helm.ngc.nvidia.com/nvidia \
   && helm repo update
```

3. Install the Operator.

```
helm install --wait --generate-name \
   -n gpu-operator --create-namespace \
   nvidia/gpu-operator \
   --set sandboxWorkloads.enabled=true \
   --set kataManager.enabled=true \
   --set ccManager.enabled=true \
   --set nfd.nodefeaturerules=true

NAME: gpu-operator
LAST DEPLOYED: Tue Jul 25 19:19:07 2023
NAMESPACE: gpu-operator
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

4. Verify that the Kata Manager, Confidential Computing Manager, and VFIO Manager operands are running.

```
$ kubectl get pods -n gpu-operator

NAME                                                      READY   STATUS     RESTARTS
AGE
gpu-operator-57bf5d5769-nb98z                             1/1     Running    0
6m21s
gpu-operator-node-feature-discovery-master-b44f595bf-5sjxg  1/1   Running    0
6m21s
```

```
gpu-operator-node-feature-discovery-worker-lwhdr             1/1     Running     0
6m21s
nvidia-cc-manager-yzbw7                                      1/1     Running     0
3m36s
nvidia-kata-manager-bw5mb                                    1/1     Running     0
3m36s
nvidia-sandbox-device-plugin-daemonset-cr4s6                 1/1     Running     0
2m37s
nvidia-sandbox-validator-9wjm4                               1/1     Running     0
2m37s
nvidia-vfio-manager-vg4wp                                    1/1     Running     0
3m36s
```

5. Verify that the kata-qemu-nvidia-gpu and kata-qemu-nvidia-gpu-snp runtime classes are available:

```
$ kubectl get runtimeclass

NAME                        HANDLER                     AGE
kata                        kata                        37m
kata-clh                    kata-clh                    37m
kata-clh-tdx                kata-clh-tdx                37m
kata-qemu                   kata-qemu                   37m
kata-qemu-nvidia-gpu        kata-qemu-nvidia-gpu        96s
kata-qemu-nvidia-gpu-snp    kata-qemu-nvidia-gpu-snp    96s
kata-qemu-sev               kata-qemu-sev               37m
kata-qemu-snp               kata-qemu-snp               37m
kata-qemu-tdx               kata-qemu-tdx               37m
nvidia                      nvidia                      97s
```

6. Confirm that the host uses the vfio-pci device driver for GPUs:

```
$ lspci -nnk -d 10de:

65:00.0 3D controller [0302]: NVIDIA Corporation xxxxxxx [xxx] [10de:xxxx] (rev xx)
        Subsystem: NVIDIA Corporation xxxxxxx [xxx] [10de:xxxx]
        Kernel driver in use: vfio-pci
        Kernel modules: nvidiafb, nouveau
```

7. Confirm that NVIDIA Kata Manager installed the kata-qemu-nvidia-gpu-snp runtime class files:

```
$ ls -1 /opt/nvidia-gpu-operator/artifacts/runtimeclasses/kata-qemu-nvidia-gpu-snp/

5.19.2.tar.gz
config-5.19.2-109-nvidia-gpu-sev
configuration-kata-qemu-nvidia-gpu-snp.toml
```

```
dpkg.sbom.list
kata-ubuntu-jammy-nvidia-gpu.initrd
vmlinuz-5.19.2-109-nvidia-gpu-sev
...
```

8. If you have not yet set the Confidential Compute mode of the GPU , set the CC mode to state you want.

```
# Option-1: Cluster-wide command:
$ kubectl patch clusterpolicy/cluster-policy \
       -p '{"spec": {"ccManager": {"defaultMode": "on"}}}'

# Option-2: Node-level command:
$ kubectl label node myServer nvidia.com/cc.mode=on --overwrite
```

Running a Sample CoCo Workload

Now, the host system is ready to launch and deploy Confidential Containers in the lightweight Kata VM infrastructure. These Containers are isolated from each other and the hypervisor/host.

1. Verify the GPUs available for passthrough.

```
$ kubectl get nodes -l nvidia.com/gpu.present -o json | \
  jq '.items[0].status.allocatable |
    with_entries(select(.key | startswith("nvidia.com/"))) |
    with_entries(select(.value != "0"))'

{
   "nvidia.com/GH100_H100_PCIE": "1"
}
```

2. Create the cuda-vectoradd-coco.yaml file.

```
apiVersion: v1
kind: Pod
metadata:
  name: cuda-vectoradd-coco
  annotations:
    cdi.k8s.io/gpu: "nvidia.com/pgpu=0"
spec:
  runtimeClassName: kata-qemu-nvidia-gpu-snp
  restartPolicy: OnFailure
  containers:
  - name: cuda-vectoradd
    image: "nvcr.io/nvidia/k8s/cuda-sample:vectoradd-cuda11.7.1-ubuntu20.04"
```

```
    resources:
      limits:
        "nvidia.com/GH100_H100_PCIE": 1
```

3. Create the pod.

```
$ kubectl apply -f cuda-vectoradd-coco.yaml
```

4. View the logs from the pod.

```
$ kubectl logs -n default cuda-vectoradd-coco

[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

You can create other pods with other job descriptions, other containers, and so on by using this process as a template.

Attestation in a Confidential Container

PPerforming attestation in a container that is managed by the GPU Operator is slightly different than the process described in , configuration of the possible attestation flows and claims should be reviewed. Part of automating the GPU Operator is that it will automatically install the Attestation SDK.

1. Determine the kubernetes node and the pod sandbox ID.

```
$ kubectl describe pod
```

2. Access the Kata runtime to log in to the VM.

```
$ kata-runtime exec <pod-sandbox-ID>
```

3. Check the CC status of the H100.

```
$ nvidia-smi conf-compute -f

CC status: ON
```

4. Run the attestation within the Container.
   For your convenience, SmallGPUTest.py is provided to run a basic attestation of the confidential container.

```
source /gpu-attestation/nv-venv/bin/activate
python3 /gpu-attestation/nv_attestation_sdk/tests/SmallGPUTest.py

[SmallGPUTest] node name : mySystem
[['LOCAL_GPU_CLAIMS', <Devices.GPU: 2>, <Environment.LOCAL: 2>, '', '', '']]
[SmallGPUTest] call attest() - expecting True
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
                     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 the expected state.
      GPU 0 verified successfully.
      attestation result: True
      claims list:: {'x-nv-gpu-availability': True, 'x-nv-gpu-attestation-report-available': ...
      True
      [SmallGPUTest] token : [["JWT", "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e..."],
         {"LOCAL_GPU_CLAIMS": "eyJhbGciOiJIUzI1NiIsInR5cCI..."}]
      [SmallGPUTest] call validate_token() - expecting True
      True
```

This shows a successful attestation from the claims configured.

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 both during the EA and into the future. 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!

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051

http://www.nvidia.com