

Deployment Guide for SecureAl

Documentation History

DU-12302-001

Version	Date	Authors	Description of Change
1.0	7/25/2023	Rob Nertney	Initial Version for Early Access
2.0	8/30/2023	Rob Nertney	Minor fixes. EA2 Updates for Kata/CoCo and TDX installs
3.0	2/22/2024	Rob Nertney	GA Version Release
4.0	7/09/2024	Rob Nertney	Updating instructions from MVP Intel stack to more upstreamable flows.
5.0	2/25/2025	Rob Nertney	Multi GPU integration; updating Intel paths for patched solutions.
6.0	5/15/2025	Rob Nertney	Unified AMD/Intel guides. Updated for upstream 25.04 hosts.
6.1	6/17/2025	Rob Nertney	Minor Links Update

Table of Contents

Using This Guide	4
Document Structure	5
Supported Combinations of Hardware and Software	5
Hardware IT Administrator	6
Selecting Hardware	6
Setting Up the Hardware and Configuring Your System	7
Host OS Administrator	8
Setting Up the Host OS	8
Preparing the Host (AMD)	9
Preparing the Host (Intel)	11
Preparing to Launch a Guest Virtual Machine with KVM	13
(Optional) Setting Up the Guest VM	15
Virtual Machine Administrator	25
Virtual Machine User	29
Validating Your Configuration	
Installing the Attestation SDK	30
Conclusion	

Using This Guide

This guide is the most distilled set of instructions required to configure a system for SecureAI with NVIDIA[®] Hopper[™] GPUs. 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 in either Confidential Compute (CC) or Protected PCIe (PPCIe) modes. These personas are rough definitions of individuals who might have different responsibilities in the overall confidential system. The overall flow of using a confidential system is illustrated in <u>Figure 1</u>.

Figure 1. Overall Workflow



You can see that not every persona 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 your persona and goals, 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's 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 following 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 of Hardware and Software

Due to the nascency of the SecureAI 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 combinations of software and hardware, outlined in the following table:

CPU Vendor	SecureAl Mode	Host OS (Kernel)	Guest OS (Kernel)	HGX FW Bundle	NVIDIA Software
AMD	Confidential Computing (Single GPU)	Ubuntu	Ubuntu 25.04 Ubuntu 24.04 (LTS) >=1.5.0 >=1.5.0	>=1.5.0	CUDA 12.4+ w/ Driver r550.xx.yy
Genoa	Protected PCle (Multi GPU)	25.04		CUDA 12.8+ w/ Driver r570.xx.yy	
Intel	Confidential Computing (Single GPU)	Ubuntu		>=1.5.0	CUDA 12.4+ w/ Driver r550.xx.yy
Emerald Rapids	Protected PCle (Multi GPU)	kobuk-ppa		1.7.0	CUDA 12.8+ w/ Driver r570.xx.yy

Hardware IT Administrator

Figure 4. The Hardware IT Administrator Persona



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

Selecting Hardware

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

- CPU requirements:
 - Intel with TDX support (for example, Emeralds Rapids SKU 8570)
 - AMD with SEV-SNP support (Milan 7xx3, Genoa 9xx4)

- GPU requirements:
 - NVIDIA Hopper GPUs
- Other Recommendations enabled:
 - Your motherboard can be configured with Secure Boot and TDX/SNP.

To set up your system, you must configure the motherboard's BIOS to enable the options that support SecureAI mode. NVIDIA has tested the motherboards from the vendors listed in the following section with SecureAI and provided the BIOS menu-flows so that you can easily set them.

Setting Up the Hardware and Configuring Your System

Example BIOS (AMD)

```
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
```

Example BIOS (Intel)

```
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 Confidential Computing, you are now ready to begin configuring the host operating system and the hypervisor.

Host OS Administrator

Figure 3. The Host OS Administrator Persona



The Host OS Administrator is the persona that has received a system with its BIOS/UEFI configured so that it is *racked and stacked* with the SecureAI 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 Architect**, **Cloud Administrator**, or **Advanced On-Premises User**.

Setting Up the Host OS

This section provides information about setting up the host OS.

• The NVIDIA SecureAl solutions primarily reside in the guest VM, so the ability to configure and optimize the host OS and hypervisor is outside the scope of this guide.

To start the Confidential VMs, NVIDIA recommends the following:

For maximum performance on Intel CPUs:

- Set before VM boot up:
 - Use https://github.com/intel/pepc?tab=readme-ov-file#standalone-version
 - CPU Frequency Governor should be set to "performance"
 - CPU CSTATES C1E,C6 should be set to "disabled"
- After VM full boot up:
 - VM/QEMU process vCPU pinning using taskset
- The Intel TDX Module is the firmware code that should be kept up to date.

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

```
# Ensure /shared has read/write permissions for the user via chmod
$ sudo mkdir /shared
$ cd /shared/
$ sudo chmod -R 777 /shared
```

Installing the Prerequisite Host Packages

Install a supported host OS by following its 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.

Preparing the Host (AMD)

AMD currently supports SecureAI solutions out of the box, starting with Ubuntu 25.04 Server. Ensure it is installed and updated:

Packages to support the build
\$ sudo apt update
\$ sudo apt upgrade
\$ sudo reboot #if required

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 ipp2-2367 6.14.0-15-generic #15-Ubuntu SMP PREEMPT_DYNAMIC Sun Apr 6
15:05:05 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
```

The dates and hashes above might vary slightly. The important thing is to ensure your kernel is 6.14+

Validate the kernel that was configured with the proper CC options. This is done by

reviewing the /boot/config-6.14.0-15-generic \$ grep CONFIG_CRYPTO_EC /boot/config-6.14.0-15-generic CONFIG_CRYPTO_ECC=y CONFIG_CRYPTO_ECDH=y CONFIG_CRYPTO_ECDSA=m CONFIG_CRYPTO_ECRDSA=m CONFIG_CRYPTO_ECB=y CONFIG_CRYPTO_ECHAINIV=m 2. 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
     0.000000] SEV-SNP: RMP table physical range [0x000000088900000 -
0x00000000a8effff1
    6.072556] ccp 0000:45:00.1: sev enabled
[
ſ
    6.195348] ccp 0000:45:00.1: SEV firmware updated from 1.49.3 to 1.55.21
    7.793012] ccp 0000:45:00.1: SEV API:1.55 build:21
[
[
    7.793024] ccp 0000:45:00.1: SEV-SNP API:1.55 build:21
    7.806923] kvm_amd: SEV enabled (ASIDs 100 - 509)
[
    7.806926] kvm_amd: SEV-ES enabled (ASIDs 1 - 99)
[
ſ
    7.806929] kvm_amd: SEV-SNP enabled (ASIDs 1 - 99
```

(**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 -
0x000000000a8efffff
[ 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 <u>https://developer.amd.com/sev</u> and through the linux-firmware project.

The following steps document the firmware upgrade process for the latest SEV-SNP firmware on <u>https://developer.amd.com/sev</u> 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 -
0x000000075bffff
[ 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
```

After performing this step, you may proceed to the next section: **Hardware IT Administrator**

Preparing the Host (Intel)

Intel currently supports SecureAI on Ubuntu 25.04 Server with a "Kobuk" PPA. Their main host setup instructions can be found at <u>https://github.com/canonical/tdx/</u>. The following section within this document will go through the reduced set of steps; please route any questions/comments on the flow to the aforementioned Canonical GitHub page.

Prepare System & Download the Required Files

Update your system:

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

if prompted:
\$ sudo reboot

Downloading the GitHub Packages

Clone the required GitHub Repositories:

```
## Mainline nvtrust
$ cd /shared/
$ git clone <u>https://github.com/NVIDIA/gpu-admin-tools</u>
$ wget <u>https://github.com/canonical/tdx/archive/refs/tags/3.3.zip</u>
$ unzip 3.3.zip
$ mv tdx-3.3 tdx
$ cd /shared/tdx/
```

Configure the Host

You may customize the setup of the host and the resulting VM by editing the configuration file setup-tdx-config.

To complete the configuration of the host, run the following:

```
$ sudo ./setup-tdx-host.sh
$ sudo reboot
```

Validating the Host Detects TDX

To check that your kernel is the new TDX-aware version, and that your configuration options were correctly applied, run the following commands.

```
$ sudo dmesg | grep -i tdx
[sudo] password for user:
[ 10.162072] virt/tdx: BIOS enabled: private KeyID range [64, 128)
[ 10.162074] virt/tdx: Disable ACPI S3. Turn off TDX in the BIOS to use ACPI S3.
[ 21.678799] virt/tdx: TDX module 1.5.06.00, build number 744, build date 0134d817
[ 26.540654] virt/tdx: 8405028 KB allocated for PAMT
[ 26.540658] virt/tdx: module initialized
```



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 GPUs in confidential modes.

 \equiv

Configuring the GPU for Confidential Compute Mode

The NVIDIA GPUs can be toggled into and out of CC modes only 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 one of the following:
 - ∎ on
 - ∎ off
 - devtools

Refer to our <u>whitepaper</u> for more information about what the modes represent. NVIDIA has provided the following script to help facilitate this call:

```
$ cd /shared/gpu-admin-tools
```

You may choose the mode in which you want your GPUs (and optional NVLink Switches) to be configured. Remember that these modes are mutually exclusive per-GPU, and you may modify this script to individually select theGPUs for which you want to change modes. To individually select a GPU, replace --gpu=\$i with --gpu-bdf=xx:00.0 where xx refers to the PCIe BDF of the GPU that you want to select.

To set all GPUs into Confidential Computing Mode:

Ensure all NVIDIA devices are not in PPCIe mode:

```
$ for i in $(seq 0 $(($(lspci -nn | grep -c "10de") - 1))); do sudo python3
./nvidia_gpu_tools.py --set-ppcie-mode=off --reset-after-ppcie-mode-switch --gpu=$i;
done
```

Set all NVIDIA GPUs into CC mode:

```
$ for i in $(seq 0 $(($(lspci -nn | grep -c "10de") - 1))); do sudo python3
./nvidia_gpu_tools.py --set-cc-mode=on --reset-after-cc-mode-switch --gpu=$i; done
```

NOTE: The following errors can be ignored:

2025-02-26,22:12:11.043 ERROR Configuring CC not supported on NvSwitch 0000:07:00.0 NVSwitch_gen3 0x22a3 BAR0 0xa6000000

To set all GPUs into Protected PCIe (Multi-GPU) Mode:

Ensure all NVIDIA devices are not in CC mode:

```
$ for i in $(seq 0 $(($(lspci -nn | grep -c "10de") - 1))); do sudo python3
./nvidia_gpu_tools.py --set-cc-mode=off --reset-after-cc-mode-switch --gpu=$i; done
```

To set the GPUs and NVSwitches into Protected PCIe mode:

```
$ for i in $(seq 0 $(($(lspci -nn | grep -c "10de") - 1))); do sudo python3
./nvidia_gpu_tools.py --set-ppcie-mode=on --reset-after-ppcie-mode-switch --gpu=$i;
done
```

To set all GPUs into Normal Operation:

```
$ for i in $(seq 0 $(($(lspci -nn | grep -c "10de") - 1))); do sudo python3
./nvidia_gpu_tools.py --set-cc-mode=off --reset-after-cc-mode-switch --gpu=$i; done
$ for i in $(seq 0 $(($(lspci -nn | grep -c "10de") - 1))); do sudo python3
./nvidia_gpu_tools.py --set-ppcie-mode=off --reset-after-ppcie-mode-switch --gpu=$i;
done
```

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



Warning: You must complete the previous step for **every** GPU and switch you pass into the CVM. This configuration is persistent across reboots and power cycles, so you **must** revert these changes by running the previous commands again with --set-<mode>-mode=off.

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

(Optional) Setting Up the Guest VM

The deployment of SecureAI solutions can vary, and some companies might want to use existing virtual machines. Here are the instructions to create a VM that can be used with SecureAI.

Create VM (Intel)

Intel calls Confidential Virtual Machines as "Trust Domains" (TD's). They provide a script to

create the TD/CVM:

\$ cd /shared/tdx/guest-tools/image/ \$ sudo ./create-td-image.sh -v 24.04

To prepare your TD/CVM:

Above, we see a single GPU with a BDF of b8:00.0. To pass this GPU to the CVM perform

the following:

```
$ sudo guest-tools/run_td
--image=guest-tools/image/tdx-guest-ubuntu-24.04-generic.qcow2 --gpus=0000:b8:00.0
```

To enable PPCIe mode by passing all 8x GPUs and 4x switches, comma-delimit the --gpus= flag.

The logs from the launch:

```
TD started by QEMU with PID: 13392.
To log in with the non-root user (default: tdx / password: 123456), as specified in
setup-tdx-config, use:
    $ ssh -p 10022 <username>@localhost
To log in as root (default password: 123456), use:
    $ ssh -p 10022 root@localhost
```

Create VM (AMD)

1. Run the following commands to obtain an ISO of a supported operating system. In this example, we are using Ubuntu 24.04.

```
$ cd /shared
#download an ISO of a supported OS here, for example:
$ wget https://releases.ubuntu.com/24.04.2/ubuntu-24.04.2-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
```

```
$ qemu-img create -f qcow2 /shared/ubuntu.qcow2 500G
```

Installing the Guest OS (AMD)

Create the file /shared/launch_vm.sh and add this information to it. #!/bin/bash

```
CORES=16
MEM=32
VDD_IMAGE=/shared/ubuntu.gcow2
FWDPORT=9899
CDROM=/shared/ubuntu-24.04.2-live-server-amd64.iso
doecho=false
docc=true
sev=""
while getopts "expc:" flag
do
 case ${flag} in
   e) doecho=true;;
   x) docc=false;;
   p) FWDPORT=${OPTARG};;
   c) sev=${OPTARG};;
 esac
done
NVIDIA_GPU=$(lspci -d 10de: | awk '/NVIDIA/{print $1}')
NVIDIA_PASSTHROUGH=$(lspci -n -s $NVIDIA_GPU | awk -F: '{print $4}' | awk '{print
$1}')
if [ "$doecho" = true ]; then
 echo 10de $NVIDIA_PASSTHROUGH > /sys/bus/pci/drivers/vfio-pci/new_id
fi
get_cbitpos() {
      modprobe cpuid
      EBX=$(dd if=/dev/cpu/0/cpuid ibs=16 count=32 skip=134217728 | tail -c 16 | od
-An -t u4 -j 4 -N 4 | sed -re 's|^ *||')
```

```
CBITPOS=$((EBX & 0x3f))
}
if [ "$docc" = true ]; then
 if [ -n "$sev" ]; then
       case "$sev" in
         sev|sev-es|sev-snp)
           SEV_MODE="$sev"
           USE_CC=true
                      get_cbitpos
           ;;
         *)
           echo "Error: unsupported SEV mode '$sev'."
                      echo "Use '-c' with valid options: sev, sev-es, sev-snp."
                      echo "Or use '-x' to boot without CC modes"
           exit 1
           ;;
       esac
     fi
fi
qemu-system-x86_64 \
  -bios /usr/share/ovmf/OVMF.fd \
 -nographic \
 ${USE_CC:+ -machine confidential-guest-support=sev0,vmport=off} \
 ${USE_CC:+$( [ "$SEV_MODE" = sev ] && \
       echo " -object
sev-guest,id=sev0,cbitpos=${CBITPOS},reduced-phys-bits=1,policy=0x1" )} \
  ${USE_CC:+$( [ "$SEV_MODE" = sev-es ] && \
       echo " -object
sev-guest,id=sev0.cbitpos=${CBITPOS},reduced-phys-bits=1,policy=0x5" )} \
  ${USE_CC:+$( [ "$SEV_MODE" = sev-snp ] && \
       echo " -object
sev-snp-guest,id=sev0,cbitpos=${CBITPOS},reduced-phys-bits=1,policy=0x30000" )} \
  -vga none \
 -enable-kvm -no-reboot \
 -cpu EPYC-v4 \
 -machine q35 -smp $CORES -m ${MEM}G,slots=2,maxmem=512G \
 -drive file=$VDD_IMAGE,if=none,id=disk0,format=qcow2 \
 -device virtio-scsi-pci,id=scsi0,disable-legacy=on,iommu_platform=true,romfile= \
 -device scsi-hd,drive=disk0 \
 -netdev user,id=vmnic,hostfwd=tcp::$FWDPORT-:22 \
 -cdrom $CDROM \
 -device virtio-net-pci,disable-legacy=on,iommu_platform=true,netdev=vmnic,romfile= \
 -device pcie-root-port,id=pci.1,bus=pcie.0 \
  -device vfio-pci,host=${NVIDIA_GPU},bus=pci.1,romfile=
```

Ensure that launch_vm.sh is executable: \$ chmod +x /shared/launch_vm.sh

Launch the VM with CC disabled: \$ sudo /shared/launch_vm.sh -x

Modifying GRUB to Print the Kernel Launch to the TTY (AMD)

1. Highlight 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 Serv	rver'
set gfxpay	load=keep	
linux initrd	/casper/vmlinuz /casper/initrd	
Minimum Emacs– completions. P a command–line	like screen editing ress Ctrl—x or F10 or ESC to discard	ng is supported. TAB lists) to boot, Ctrl–c or F2 for 1 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/vmlinuz console=ttyS0 ---

Figure 7. Printing the Local Console

		GNU GRUB	version 2.06				
setparams 'Try or Install Ubuntu Server'							
set	gfxpayload	=keep					
lin ini	ux /	<pre>casper/vmlinuz /casper/initrd</pre>	console=ttyS0				
Minimum complet a comma	Emacs-like ions. Press nd-line or l	screen editing Ctrl–x or F10 ESC to discard	g is supported. TAB lists to boot, Ctrl-c or F2 for 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 CPU SecureAI limitations, a reboot command **terminates** the VM. This behavior is **expected** for all subsequent reboots.

Finalizing the Guest OS (AMD)

After you complete the installation, and the reboot of the guest VM is requested, QEMU will terminate. Edit the VM launch commands and restart it via launch_vm.sh afterwards:

```
# Edit launch_vm.sh to remove the following line:
  -cdrom $CDROM \
# Save and quit
$ sudo /shared/launch_vm.sh -x
```

```
Log into your CVM.
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-57-generic x86_64)
 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
* Support:
                 https://ubuntu.com/pro
 System information as of Thu Apr 10 04:44:27 PM UTC 2025
 System load:
                          0.0
 Usage of /:
                          10.7% of 97.87GB
 Memory usage:
                         1%
 Swap usage:
                          0%
 Processes:
                          232
 Users logged in:
                          Ø
 IPv4 address for enp0s2: 10.0.2.15
 IPv6 address for enp0s2: fec0::5054:ff:fe12:3456
 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.
  https://ubuntu.com/engage/secure-kubernetes-at-the-edge
Expanded Security Maintenance for Applications is not enabled.
52 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
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.
user@guest:~$
```

Enabling LKCA on the Guest VM (AMD)

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

```
    Create a /etc/modprobe.d/nvidia-lkca.conf file and add this line to it:
install nvidia /sbin/modprobe ecdsa_generic; /sbin/modprobe ecdh; /sbin/modprobe
--ignore-install nvidia
```

```
2. Update the initramfs. sudo update-initramfs -u
```

Configure PCIe BAR Remapping (AMD)

OVMF sometimes has issues with being able to generate ACPI addresses for PCIe devices with high amounts of memory. We will work around this with a boot argument telling Linux to handle the remap:

3. Create a /etc/default/grub file and edit it by adding pci=realloc, nocrs to GRUB_CMDLINE_LINUX_DEFAULT

```
# If you change this file, run 'update-grub' afterwards to update
   # /boot/grub/grub.cfg.
    # For full documentation of the options in this file, see:
        info -f grub -n 'Simple configuration'
   #
    GRUB DEFAULT=0
    GRUB_TIMEOUT_STYLE=hidden
    GRUB TIMEOUT=0
    GRUB_DISTRIBUTOR=`( . /etc/os-release; echo ${NAME:-Ubuntu} ) 2>/dev/null || echh
    o Ubuntu`
    GRUB_CMDLINE_LINUX_DEFAULT="console=ttyS0 pci=realloc,nocrs"
    GRUB_CMDLINE_LINUX=""
   # If your computer has multiple operating systems installed, then you
   # probably want to run os-prober. However, if your computer is a host
    # for guest OSes installed via LVM or raw disk devices, running
    # os-prober can cause damage to those guest OSes as it mounts
    # filesystems to look for things.
   #GRUB_DISABLE_OS_PROBER=false
   # Uncomment to enable BadRAM filtering, modify to suit your needs
   # This works with Linux (no patch required) and with any kernel that obtains
   # the memory map information from GRUB (GNU Mach, kernel of FreeBSD ...)
4. Update GRUB and reboot.
```

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

 After you complete the configuration, and the reboot of the guest VM is requested, QEMU will terminate. Relaunch the VM via launch_vm.sh afterwards and remove the -x flag: this enables CC modes in the VM.

```
$ sudo /shared/launch_vm.sh -c {sev, sev-es, sev-snp}
```

Validating the Guest (Intel):

- 1. After the CVM guest is launched, log in using SSH and check the dmesg log to confirm that the TDX hooks are detected.
- 2. Change the username to match the name that you configured in previous steps.
- 3. Log in
 \$ ssh -p10022 root@localhost
- 4. Check the kernel logs for TDX support.

```
$ sudo dmesg | grep -i tdx
[sudo] password for user:
[    0.000000] tdx: Guest detected
[    29.669467] process: using TDX aware idle routine
[    29.669467] Memory Encryption Features active: Intel TDX
[    46.864001] systemd[1]: Detected confidential virtualization tdx.
```

Validating the Guest (AMD):

Validate that the Guest is in CVM mode:

```
$ sudo dmesg | grep -i sev
[ 1.261594] Memory Encryption Features active: AMD SEV SEV-ES SEV-SNP
[ 1.376509] SEV: APIC: wakeup_secondary_cpu() replaced with
wakeup_cpu_via_vmgexit()
[ 1.482689] SEV: Using SNP CPUID table, 28 entries present.
[ 1.90332] SEV: SNP guest platform device initialized.
[ 6.028948] sev-guest sev-guest: Initialized SEV guest driver (using vmpck_id 0)
```

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.



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

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 of 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 instalation. 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. Intel's scripts will have prepared this guest completely; for AMD, please follow the below instructions:

Enabling LKCA on the Guest VM (AMD Only)

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

- Create the /etc/modprobe.d/nvidia-lkca.conf file and add this line to it: install nvidia /sbin/modprobe ecdsa_generic; /sbin/modprobe ecdh; /sbin/modprobe --ignore-install nvidia
- (Optional) If you are using PPCIe Mode, change the /etc/modprobe.d/nvidia-lkca.conf file as follows:

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

3. 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 the NVIDIA kernel drivers, and the source can be found on the <u>NVIDIA Linux Open GPU Kernel Module GitHub</u> project.

Hopper CC is enabled starting with CUDA 12.4 and is paired with the R550 driver, which can be downloaded as described below.

```
Multi-GPU Protected PCIe is enabled starting with CUDA 12.8 and is paired with 570 driver.
# In the guest:
```

```
# Obtain the NVIDIA keys to download the CUDA Toolkit
$ wget
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2404/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-12-8
# Install the driver
$ sudo apt install nvidia-driver-570-open nvidia-fabricmanager-570
```

(Optional) Starting Fabric Manager

If using PPCIe Multi-GPU mode, start fabric manager and confirm that it started by running the following commands:

```
$ systemctl enable nvidia-fabricmanager
```

```
$ systemctl start nvidia-fabricmanager
```

```
$ systemctl status nvidia-fabricmanager
```

Setting up the NVIDIA Driver to be in Persistence Mode

When the NVIDIA driver loads, the driver will automatically establish a secured SPDM session with the GPU. 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 SecureAI modes, this behavior causes the shared-secret and the shared keys that were established during the setup SPDM phase of the driver to be destroyed. 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 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.

Determine whether nvidia-persistenced is already running in persistence mode.
 \$ ps -aux | grep nvidia-persistenced

If you see output similar to the following example, nvidia-persistenced is already running in persistence mode:

nvidia-+ 797 0.0 0.0 5472 1852 ? Ss 17:23 0:00 /usr/bin/nvidia-persistenced --user nvidia-persistenced --uvm-persistence-mode --verbose

If you see only the grep command in the output similar to the following example, nvidia-persistenced is not already running:

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

If you see --no-persistence-mode in the output similar to the following example, nvidia-persistenced is already running but not in persistence mode: 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 nvidia-persistenced is not already running or is already running but not in persistence mode, 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

```
# In this line, change --no-persistence-mode to --uvm-persistence-mode:
    ExecStart=/usr/bin/nvidia-persistenced --user nvidia-persistenced
    --no-persistence-mode --verbose
```

```
# The result of this change:
    ExecStart=/usr/bin/nvidia-persistenced --user nvidia-persistenced
--uvm-persistence-mode --verbose
```

b. Tell systemd to reload its modules and enable the nvidia-persistenced.service, and reboot the guest VM.

```
# On the guest:
$ sudo systemctl daemon-reload
$ sudo systemctl enable nvidia-persistenced.service
$ sudo reboot
```

Validating State and Versions

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

```
$ nvidia-smi conf-compute -f
CC status: ON
$ nvidia-smi conf-compute -d
DevTools Mode: OFF
$ nvidia-smi conf-compute -mgm
Multi-GPU Mode: Protected PCIe
```

Ensure that the firmware on the H100 GPU meets or exceeds the minimum version required for the mode that you're using:

- GH100 CC Mode: 96.00.5E.00.00
- GH100 PPCIe Mode: 96.00.BC.00.01
- LS10 PPCIe Mode: 96.10.69.00.01

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

: 96.00.5E.00.25

If you have an earlier version of the VBIOS, contact your OEM for further instructions.

Verify the system state by ensuring that the following tasks are complete:

- Fabric manager is enabled.
- NVIDIA NVLink[®] is up.
- The NVLink P2P status has been enabled.

\$ nvidia-smi topo -m

	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	CPU	Affinity	NUMA
	LLY GPU	NUMA	ID								
GPUØ N/A	Х	NV18	0-15	0							
GPU1 N/A	NV18	Х	NV18	NV18	NV18	NV18	NV18	NV18	0-15	0	
GPU2	NV18	NV18	Х	NV18	NV18	NV18	NV18	NV18	0-15	0	N/A
GPU3	NV18	NV18	NV18	Х	NV18	NV18	NV18	NV18	0-15	0	N/A
GPU4	NV18	NV18	NV18	NV18	Х	NV18	NV18	NV18	0-15	0	N/A
GPU5	NV18	NV18	NV18	NV18	NV18	Х	NV18	NV18	0-15	0	N/A
GPU6	NV18	NV18	NV18	NV18	NV18	NV18	Х	NV18	0-15	0	N/A

GPU7	NV18	NV18	NV18	NV18	NV18	NV18	NV18	X	0-15	0	N/A
\$ nvi	dia-smi	topo -	p2p n								
	GPU0	GPU1	GPU2	GPU3	GPU4	GPU5	GPU6	GPU7	,		
GPU0	Х	ОК	OK	ОК	ОК	ОК	ОК	ОК			
GPU1	ОК	Х	OK	OK	ОК	OK	ОК	ОК			
GPU2	ОК	OK	Х	OK	ОК	OK	ОК	ОК			
GPU3	ОК	ОК	ОК	Х	ОК	ОК	ОК	ОК			
GPU4	ОК	OK	OK	OK	Х	OK	ОК	ОК			
GPU5	ОК	OK	OK	OK	ОК	Х	ОК	ОК			
GPU6	ОК	ОК	ОК	ОК	ОК	ОК	Х	ОК			
GPU7	ОК	ОК	ОК	ОК	ОК	ок	ок	х			

You have successfully configured the guest CVM to operate in the CC or with PPCIE mode with a secured H100s accelerator.

After the SecureAI capable driver package is installed, the guest user must ensure the following requirements are always maintained:

- Persistence mode for NVIDIA drivers **must** be enabled.
- After the driver exits, it will not load again until the next GPU reboot.
- The driver unload must be followed by a GPU device reset (PF-FLR) or a system reboot before loading again.

Virtual Machine User

Figure 9. Virtual Machine User



The Virtual Machine User might (or might not) be the administrator of the system (refer to <u>Virtual Machine Administrator</u> for more information). This persona assumes that the system is correctly configured for CC.



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 restriction prevents accidental usage before the confirmation of the GPU is complete.

Successfully passing attestation as root (see below) or running the command below will set the ready state.

nvidia-smi conf-compute -srs 1

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, skip this verification step.

08:57

0:05

\$ ps -aux | grep nvidia-persistenced root 2327 20.1 0.0 5312 1788 ? Ss nvidia-persistenced

- 2. If nothing is returned, run the following command to start nvidia-persistenced.\$ sudo nvidia-persistenced
- 3. Check the status of the GPU to ensure that it is configured in your desired SecureAI mode.

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

Installing the Attestation SDK

After validating the configuration, complete an attestation of the system (refer to the <u>Protected PCIe Verifier tool documentation</u>).

Conclusion

This guide provides information about the process that starts when the machine is racked and stacked, configuring the host and guest operating systems, and attaching an H100 GPU in a CVM. This flow can be modified to suit your specific needs.

To provide feedback, comments, or ask questions during the EA PPCI build and into the future, go to

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

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

Notice

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

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

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

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

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

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

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

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

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

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051 http://www.nvidia.com



CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

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

VESA DisplayPort

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

HDMI

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

Arm

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

OpenCL

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

Copyright

© 2023-2025 NVIDIA Corporation. All rights reserved.

