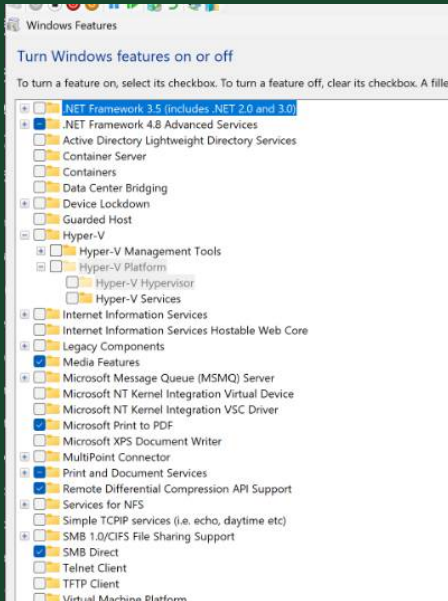


Hyper-V greyed out VM



the hyper-v hypervisor is greyed out. this is win 11 pro virtual machine.

Got it. The reason **Hyper-V Hypervisor is greyed out** in your screenshot is because you are trying to enable Hyper-V **inside a Windows 11 Pro virtual machine**.

Here's why:

- **Hyper-V requires hardware virtualization (Intel VT-x or AMD-V + SLAT)** directly from the CPU.
- When you are already running Windows 11 Pro as a **guest VM**, the VM does not get direct access to the CPU virtualization extensions, unless your host hypervisor explicitly supports **nested virtualization**.
- Since the guest VM doesn't see those virtualization features, Windows disables the **Hyper-V Hypervisor** option (hence it's greyed out).

How to fix / enable it:

1. Check if your host hypervisor supports nested virtualization:

- **VMware Workstation/ESXi** → Enable nested virtualization in VM settings.
- **Hyper-V host** → Run:

```
powershell  
  
Set-VMProcessor -VMName "YourVMName" -ExposeVirtualizationExtensions $true
```

- **VirtualBox** → Enable "Nested VT-x/AMD-V" in the VM's System → Processor settings.

2. Reboot the guest VM after enabling nested virtualization. Then the **Hyper-V Hypervisor** checkbox will no longer be greyed out.

Bottom line:

- On a **physical Windows 11 Pro machine** → You can enable Hyper-V directly.
- On a **Windows 11 Pro VM** → You need **nested virtualization** enabled from the host; otherwise Hyper-V inside the VM will remain unavailable.

👉 Which hypervisor are you running your Windows 11 Pro VM on (VirtualBox, VMware, Hyper-V, etc.)? That will change the exact steps I give you.



how do i list all the vms

Since you said this is **Windows 11 Pro running as a VM**, the way to list all the VMs depends on which hypervisor you're actually using to run it.

Here are the common ones:

♦ If your host is Hyper-V (Windows built-in hypervisor):

Open **PowerShell (as Admin)** and run:

```
powershell
```

```
Get-VM
```

That will show you all VMs, their state (Running/Off), and resource allocation.

♦ If your host is VirtualBox:

Run this in your host machine (not inside the VM):

```
powershell
```

```
VBoxManage list vms
```

or from CMD:

```
cmd
```

```
"C:\Program Files\Oracle\VirtualBox\VBoxManage.exe" list vms
```

♦ If your host is VMware Workstation / ESXi:

- Workstation/Player (Windows/Linux host):

```
cmd
```

```
vmrun list
```

- ESXi host:

```
bash
```

```
vim-cmd vmsvc/getallvms
```

♦ If your host is AWS EC2 / Cloud:

If your “Windows 11 Pro” VM is inside EC2, you don’t “list VMs” locally—you’d list EC2 instances with:

```
bash  
  
aws ec2 describe-instances
```

⚠ Important: You mentioned earlier you’re using **VirtualBox on Windows 11 Pro with 20 different OS VMs**.
If this Win11 Pro VM is running **inside VirtualBox**, then run:

```
cmd  
  
VBoxManage list vms
```

on the **physical host machine** (not inside the Win11 guest).

Do you want me to give you the exact command for **VirtualBox on Windows 11 Pro host** since that matches what you told me before?



i am using hyper-v in my host os which is win 11 pro

Perfect — thanks for confirming. Since your **host OS is Windows 11 Pro with Hyper-V enabled**, you can list all the VMs directly with PowerShell.

✓ To list all VMs in Hyper-V:

Open **PowerShell as Administrator** on the host and run:

```
powershell  
  
Get-VM
```

This will show you:

- VM Name
- State (Running, Off, Paused)
- CPU usage
- Assigned memory
- Uptime
- Integration services version

✓ To get more details about a specific VM:

```
powershell

Get-VM -Name "YourVMName" | Format-List *
```

✓ To list VMs with networking info:

```
powershell

Get-VMNetworkAdapter
```

⚠ Remember: since your Windows 11 Pro **guest VM** is running inside Hyper-V, the **Hyper-V Hypervisor** option inside that guest will stay **greyed out** until you enable **nested virtualization** for it from the host.

Would you like me to give you the exact command to enable **nested virtualization** for that Win11 VM on your Hyper-V host?



below are the details of my vm where i want to implement nested virtualisation -

```
PS C:\WINDOWS\system32> Get-VM -Name "0.1.win-11-pro-template" | Format-List *
```

```
ParentCheckpointId      : 3e12ad84-e99b-47e2-a76a-40269c676da6
ParentCheckpointName     : Automatic Checkpoint - 0.1.win-11-pro-template - (01-10-2025 - 10:55:52)
VMName                  : 0.1.win-11-pro-template
VMId                    : d6b0bfa6-a878-4102-b624-ceefa81adf42
CheckpointFileLocation  : C:\ProgramData\Microsoft\Windows\Hyper-V
ConfigurationLocation   : C:\ProgramData\Microsoft\Windows\Hyper-V
GuestStatePath          : C:\ProgramData\Microsoft\Windows\Hyper-V\Virtual Machines\D6B0BFA6-A878-4102-B624-CEEFA81ADF42.vmgs
SmartPagingFileInUse    : False
SmartPagingFilePath     : C:\ProgramData\Microsoft\Windows\Hyper-V
SnapshotFileLocation    : C:\ProgramData\Microsoft\Windows\Hyper-V
AutomaticStartAction    : StartIfRunning
AutomaticStartDelay     : 0
AutomaticStopAction     : Save
AutomaticCriticalErrorAction : Pause
AutomaticCriticalErrorActionTimeout : 30
AutomaticCheckpointsEnabled : True
CPUUsage                : 0
MemoryAssigned          : 0
MemoryDemand            : 0
MemoryStatus            :
NumaAligned              :
NumaNodesCount          : 1
NumaSocketCount         : 1
Heartbeat               :
IntegrationServicesState :
IntegrationServicesVersion : 0.0
Uptime                  : 00:00:00
OperationalStatus       : {Ok}
PrimaryOperationalStatus : Ok
SecondaryOperationalStatus :
StatusDescriptions      : {Operating normally}
PrimaryStatusDescription : Operating normally
```

```

SecondaryStatusDescription      :
Status                          : Operating normally
ReplicationHealth                : NotApplicable
ReplicationMode                  : None
ReplicationState                 : Disabled
ResourceMeteringEnabled          : False
CheckpointType                   : Standard
EnhancedSessionTransportType     : VMBus
Groups                           : {}
Version                          : 12.0
VirtualMachineType               : RealizedVirtualMachine
VirtualMachineSubType            : Generation2
GuestStateIsolationType          : Disabled
Notes                            :
State                            : Off
ComPorts                         : {COM 1, COM 2}
ComPort1                         : VMComPort (Name = 'COM 1', VMName = '0.1.win-11-pro-template') [Id =
                                'Microsoft:D6B0BFA6-A878-4102-B624-CEEFA81ADF42\8E3A359F-559A-4B6A-98A9-1690A6100ED7\0', VMId = 'd6b0bfa6-a878-
                                4102-b624-ceefa81adf42']
ComPort2                         : VMComPort (Name = 'COM 2', VMName = '0.1.win-11-pro-template') [Id =
                                'Microsoft:D6B0BFA6-A878-4102-B624-CEEFA81ADF42\8E3A359F-559A-4B6A-98A9-1690A6100ED7\1', VMId = 'd6b0bfa6-a878-
                                4102-b624-ceefa81adf42']
DVDDrives                       : {}
FibreChannelHostBusAdapters     : {}
FloppyDrive                      :
HardDrives                      : {Hard Drive on SCSI controller number 0 at location 0}
RemoteFxAdapter                  :
VMIntegrationService              : {Guest Service Interface, Heartbeat, Key-Value Pair Exchange, Shutdown...}
DynamicMemoryEnabled              : True
MemoryMaximum                    : 1099511627776
MemoryMinimum                    : 536870912
MemoryStartup                    : 8589934592
ProcessorCount                   : 11
BatteryPassthroughEnabled         : True
Generation                      : 2

```

```

IsClustered           : False
ParentSnapshotId      : 3e12ad84-e99b-47e2-a76a-40269c676da6
ParentSnapshotName     : Automatic Checkpoint - 0.1.win-11-pro-template - (01-10-2025 - 10:55:52)
Path                  : C:\ProgramData\Microsoft\Windows\Hyper-V
SizeOfSystemFiles     : 73728
GuestControlledCacheTypes : False
LowMemoryMappedIoSpace : 134217728
HighMemoryMappedIoSpace : 536870912
HighMemoryMappedIoBaseAddress : 68182605824
LockOnDisconnect       : Off
CreationTime           : 07-06-2025 14:13:49
Id                     : d6b0bfa6-a878-4102-b624-ceefa81adf42
Name                   : 0.1.win-11-pro-template
NetworkAdapters        : {Network Adapter}
CimSession              : CimSession: .
ComputerName           : TECHDJIN-I9
IsDeleted              : False

```

how to enable it in the host OS

Nice — thanks for dumping the VM info.

Your VM is a **Generation 2 Windows 11 Pro VM** on **Hyper-V host (Windows 11 Pro)**.

Right now the **Hyper-V Hypervisor inside the guest is greyed out** because **nested virtualization** isn't enabled on this VM from the host.

Here's exactly how to enable it:

◆ Steps to Enable Nested Virtualization for your VM

1. Shut down the VM

Since your VM (0.1.win-11-pro-template) is currently **Off**, you're good.

2. Run this PowerShell command on the host (as Administrator):


```
powershell
```

```
Set-VMProcessor -VMName "0.1.win-11-pro-template" -ExposeVirtualizationExtensions $true
```

✅ This tells Hyper-V to pass Intel VT-x / AMD-V features into the guest so it can run its own hypervisor.

3. (Optional but Recommended) **Enable MAC spoofing** if you'll run nested VMs with networking:

```
powershell
```

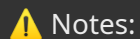
```
Get-VMNetworkAdapter -VMName "0.1.win-11-pro-template" | Set-VMNetworkAdapter -MacAddressSpoofing On
```

4. **Start the VM** again:

```
powershell
```

```
Start-VM -Name "0.1.win-11-pro-template"
```

5. Inside the guest Windows 11 Pro VM, open **Windows Features** again — now the **Hyper-V Hypervisor** checkbox will no longer be greyed out, and you can install Hyper-V inside the VM.



Notes:

- Only **Intel CPUs with VT-x + EPT** or **AMD CPUs with AMD-V + RVI** support this.
- Nested virtualization is supported only for **Generation 2 VMs** (your VM is Gen2, so you're fine).
- Performance takes a hit — don't expect production-level speed.



Do you want me to also give you the command to **verify that nested virtualization is enabled** after you apply this setting?



i want to fetch all the below vms nested virtualization states -

```
PS C:\WINDOWS\system32> get-vm
```

Name	State	CPUUsage(%)	MemoryAssigned(M)	Uptime	Status	Version
0.1.win-11-pro-template	Off	0	0	00:00:00	Operating normally	12.0
0.2.ubuntu-25	Off	0	0	00:00:00	Operating normally	12.0
0.4.debian-12-template	Off	0	0	00:00:00	Operating normally	12.0
1a.win-11-pro-ELK	Off	0	0	00:00:00	Operating normally	12.0
1b.win11-Azure-Labs	Off	0	0	00:00:00	Operating normally	12.0
1c.win11-wordpress	Off	0	0	00:00:00	Operating normally	12.0
1d.win11-MySQL-docker-k8s	Off	0	0	00:00:00	Operating normally	12.0
3a.ubuntu-24.04.3-MySQL	Off	0	0	00:00:00	Operating normally	12.0
4.debian-12-linux-practice	Off	0	0	00:00:00	Operating normally	12.0

Good — since you've got multiple VMs and you want to **check whether nested virtualization (ExposeVirtualizationExtensions) is enabled** for each of them, you can fetch it with PowerShell.

✓ Command to fetch nested virtualization state for all VMs:

```
powershell
```

```
Get-VM | Select-Object Name, State, @{Name="NestedVirtualization";Expression={(Get-VMProcessor $_.Name).ExposeVirtualizationExtensions}}
```

📌 Example output you'll get:

```
vbnet
```

Name	State	NestedVirtualization
0.1.win-11-pro-template	Off	True
0.2.ubuntu-25	Off	False
0.4.debian-12-template	Off	False
1a.win-11-pro-ELK	Off	True
1b.win11-Azure-Labs	Off	False
...		

- **True** → Nested virtualization is enabled.

- **False** → Nested virtualization is disabled.

⚡ If you later want to **enable nested virtualization for all VMs at once**, you can run:

```
powershell  
  
Get-VM | ForEach-Object { Set-VMProcessor -VMName $_.Name -ExposeVirtualizationExtensions $true }
```

👉 Do you want me to also include the **MAC spoofing state** for each VM in the same output? (it's usually needed when you plan to run guest VMs with networking inside these VMs).



ok now i have resolved the nested vm issue in my guest win 11 pro os.
wsl is working fine.
i have the Ubuntu os running in wsl now.
in context of above usecase, is it possible to install docker desktop in the virtual machine

Yes — now that you've enabled **nested virtualization** and got **WSL2 working** inside your Windows 11 Pro guest VM, you *can* install **Docker Desktop** inside that VM.

Here's the breakdown for your use case:

✅ Why it works now

- **Docker Desktop (Windows)** requires **WSL2 backend** or Hyper-V backend.
- You've already confirmed WSL2 is running fine in the guest → which means the nested virtualization setting is working.
- That's the main blocker — without it, Docker Desktop wouldn't even start.

⚡ What you can do

1. Inside your **Windows 11 Pro** guest VM:

- Download Docker Desktop from [Docker official site](#) ↗.
- Install it normally.
- During setup, ensure **WSL2 backend** is selected (avoid Hyper-V backend if you don't need it).

2. After install:

- Open **Docker Desktop** → Settings → Resources → WSL Integration.
- Enable integration for your **Ubuntu WSL instance**.

3. Test inside your Ubuntu WSL:

```
bash

docker --version
docker run hello-world
```

⚠ Things to keep in mind

- **Performance:** You're running Docker → on WSL → inside a VM → on Hyper-V. That's 2 layers of virtualization, so performance won't be as good as on bare metal.
- **Networking:** By default, Docker inside the guest VM will be reachable only from that VM. If you need to expose containers to your host or LAN, you'll have to configure **NAT/bridged networking** in Hyper-V.
- **Resources:** Give your guest VM plenty of **RAM + CPU** (you already allocated 8GB+ and 11 vCPUs, which is good).

👉 Short answer: **Yes, you can install Docker Desktop inside your Windows 11 Pro VM now.**

It will run fine using the WSL2 backend since nested virtualization is enabled.

Do you want me to also show you how to make Docker containers inside the VM accessible directly from your **physical host** (so you don't need to remote into the VM every time)?



is wsl feature available in win 11 home edition

Yes  — **WSL is available in Windows 11 Home Edition.**

That's one of the nice changes compared to older Windows versions. Here's the deal:

WSL on Windows 11 Home

- **Windows 11 Home** supports both **WSL1** and **WSL2**.
 - You no longer need Pro or Enterprise editions just to run WSL2 (unlike Hyper-V).
 - WSL2 uses the built-in **Virtual Machine Platform** feature, which *is available* on Home edition.
-

How to install WSL on Win 11 Home

Run this command in **PowerShell (as Admin)**:

```
powershell  
  
wsl --install
```

That will:

- Enable required features (**VirtualMachinePlatform** + **Windows Subsystem for Linux**).
- Download and install the default Ubuntu distro.

After reboot, you'll be able to start Ubuntu with:

```
powershell  
  
wsl
```

📝 Difference vs Pro edition

- **Hyper-V** itself (full hypervisor with GUI, VM management, etc.) is **not available** on Home.
- But **WSL2 does not need Hyper-V** — it uses a lightweight subset (“Virtual Machine Platform”).

So Docker Desktop, for example, works fine on **Windows 11 Home** as long as you use the **WSL2 backend**.

👉 Do you want me to also show you the exact **optional features you need to enable** on Windows 11 Home for WSL2?



so with the wsl feature installed and ubuntu vm running on a guest vm, and also nested virtualization switched on, will i be able to install docker desktop in the guest vm?

Yes — in your exact setup (**Win11 Pro VM running on Hyper-V host, nested virtualization enabled, WSL2 installed in the guest, Ubuntu running inside WSL2**), you can install **Docker Desktop inside the guest VM** and it will work.

Here’s why and what to expect:

✅ Why it works

- **Docker Desktop requires WSL2 or Hyper-V backend.**
- You already confirmed **WSL2 is working in the guest** → that means the VM can run lightweight VMs inside it (nested virtualization is doing its job).
- Docker Desktop will hook into your WSL2 Ubuntu instance, just like it does on a bare-metal Windows install.

⚠️ Caveats

1. Performance penalty

- You’re stacking layers:
Physical hardware → Hyper-V host → Win11 Pro guest → WSL2 lightweight VM → Docker container

- It will run, but don't expect native-level performance.

2. Networking

- Containers inside the **guest VM** are reachable only inside that guest by default.
- If you want access from your physical host or LAN, you'll need Hyper-V **bridged networking** or port forwarding.

3. Resources

- Docker Desktop + WSL2 are memory-hungry. Your guest VM already has 8GB+ RAM, but if you plan to run real workloads, give it at least 12–16GB.

4. Not supported officially

- Docker Inc. only officially supports Docker Desktop on bare-metal Windows. Running it inside a VM works if nested virtualization is enabled, but it's "best effort," not guaranteed.

How you'd do it

1. In the **guest Win11 Pro VM**, download Docker Desktop.
2. During install, select **WSL2 backend** (not Hyper-V backend).
3. In Docker Desktop → Settings → Resources → WSL Integration → turn on integration for your **Ubuntu WSL instance**.
4. Inside WSL Ubuntu, test:

```
bash

docker --version
docker run hello-world
```

👉 So, short answer: **Yes, you'll be able to install Docker Desktop in the guest Win11 Pro VM and use it with WSL2 Ubuntu, since nested virtualization is already working.**

Do you want me to also show you how to configure **bridged networking** in Hyper-V so your containers inside the guest VM can be accessed directly from your physical host?



i want to export all my current powershell commands and output into a text file

Got it — you want to **capture all PowerShell commands and their outputs** into a text file.

There are a few solid ways to do this:

♦ 1. Using Start-Transcript (Recommended)

This records **everything** you type and all output to a file.

```
powershell  
  
Start-Transcript -Path "C:\Users\YourUser\Documents\ps-session.log" -Append
```

- Every command you run + its output will be logged.
- To stop recording:

```
powershell  
  
Stop-Transcript
```

♦ 2. Redirecting output to a file (basic)

If you're running a script and just want the output:

```
powershell  
  
.\yourscript.ps1 > C:\Users\YourUser\Documents\output.txt
```

If you also want errors:


```
powershell  
  
.\yourscript.ps1 *> C:\Users\YourUser\Documents\output.txt
```

But this only captures output, not the commands you typed interactively.

♦ 3. Export last command's output only

If you just want the last command's output:

```
powershell  
  
Get-VM | Out-File -FilePath "C:\Users\YourUser\Documents\vmlist.txt"
```

or nicer formatting:

```
powershell  
  
Get-VM | Format-Table | Out-String | Out-File "C:\Users\YourUser\Documents\vmlist.txt"
```

✅ For your use case (logging **everything you're doing while testing nested virtualization, WSL, Docker, etc.**), the best is:

```
powershell  
  
Start-Transcript -Path "C:\Users\YourUser\Documents\powershell-session.txt"
```

Then when you're done:

```
powershell  
  
Stop-Transcript
```

👉 Do you want me to give you a **one-liner** that starts PowerShell with transcript logging enabled automatically every time you launch it?



so the current usecase is as follows -
i already typed in 10 powershell commands.
each has its own successful output or error.
i want the entire thing - commands, output, errors logged into a text file

Got it 👍 — you already ran ~10 commands and now you want the **whole session history (commands + outputs + errors)** saved into a text file.

Here's the situation:

- **Start-Transcript** only logs **from the moment you start it onward** — it won't retroactively capture what you already ran.
- But since you've already typed the commands, we can use **command history** + **scrollback buffer** to save things.

✅ Option 1: Save all visible commands & outputs (scrollback buffer)

In your current PowerShell session, you can dump everything that's on screen + scrollback buffer into a file with:

```
powershell  
  
Get-Content (Get-PSReadLineOption).HistorySavePath | Out-File "C:\Users\YourUser\Documents\commands-only.txt"
```

⚠️ But note: this only gives you the **commands**, not the outputs.

To get both commands **and outputs** that are visible in the console right now, select everything in the PowerShell window → Right-click → **Mark** → drag → copy → paste into a file.

That's the only way to capture *past outputs* once they're gone.

✅ Option 2: Going forward (better way)

If you want **both commands + outputs + errors** from now on:

```
powershell  
  
Start-Transcript -Path "C:\Users\YourUser\Documents\full-session.log" -Append
```

→ run commands as usual

→ when done:

```
powershell  
  
Stop-Transcript
```

That file will contain **exactly what you see on screen** (commands, errors, output).

✓ Option 3: Run history of just commands

If you only care about the commands you ran (not output):

```
powershell  
  
Get-History | Format-List | Out-File "C:\Users\YourUser\Documents\history.txt"
```

⚡ Since you said you want *commands + outputs + errors for the session you already ran*:

- Past commands = can be retrieved with **Get-History**.
- Past outputs = only available in the scrollback buffer (unless you had **Start-Transcript** running).

👉 Do you want me to give you a **ready-to-run snippet** that combines **Get-History** + starts **Transcript** now, so you at least save what's possible from the past **and** log everything from this point forward automatically?