FIELD NOTES FROM A PRESALES SOLUTIONS ARCHITECT

Docker on a Windows VM Without Nested Virtualization — Process Isolation, Two Traps, and the Image That Was Linux All Along

A presales field note about a customer who wanted Docker on a Windows Server VM — on a platform with no nested virtualization — the two traps I hit proving it works, and the one check that made the whole question disappear.

Daanial Farrukh · 11 November 2026 · https://quabin.com/docker-windows-vm-no-nested-virtualization/

The request came in short and sure of itself.

"We need Docker on our Windows server. Our developers ship containers."

The server was a Windows Server VM on a KVM-based hyperconverged platform. Nested virtualization was not enabled on it.

"Which Docker? Docker Desktop?"

"Yes. That's what the developers use."

I asked one more question.

"Are the containers Windows containers or Linux containers?"

A pause.

"They're just containers."

That's the trap. "Docker on Windows" sounds like one thing. It is at least three different things, and on a VM without nested virtualization, two of them cannot start at all.

What Nested Virtualization Has to Do With Docker

Nested virtualization is a hypervisor letting a VM run its own hypervisor inside it. Without it, the guest cannot use Hyper-V, and anything that depends on Hyper-V fails.

Here is how the Docker options on Windows depend on it:

- **Docker Desktop** → needs WSL2 or Hyper-V → both need a hypervisor inside the VM → **impossible without nested virtualization.**
- **Windows containers in Hyper-V isolation** → each container gets a small utility VM → **impossible without nested virtualization.**

- **Windows containers in process isolation** → containers share the host's Windows kernel, like Linux containers share the Linux kernel → **works without nested virtualization.**

So the honest answer to "can we run Docker here?" is: Docker Engine, yes. Docker Desktop, no. Linux containers, no.

Where this shows up most:

- Windows VMs on KVM-based platforms and private clouds, where nested virtualization is off by default.
- Hosted Windows servers where the customer has no access to the hypervisor settings.
- Teams who build on laptops with Docker Desktop and assume the server behaves the same.

Why Nobody Sees It in Testing

On a developer laptop, Docker Desktop hides all of this. It quietly runs a Linux VM through WSL2 and puts Linux containers inside it. The developer types docker run and it just works. Nobody on the team has ever had to say which kernel their image needs.

Then the image meets a server that has no hypervisor to spare. The installer refuses, or WSL2 will not enable, and the first instinct is to blame the cloud.

The second instinct is to ask the cloud team to turn on nested virtualization. On a KVM platform that is not a console toggle. The VM must be powered off, its CPU mode set to host-passthrough, and the host node must have the KVM nested option on. In practice that is a support ticket to the platform vendor and a maintenance window — for one VM.

Before I asked anyone for that, I wanted to prove the cheaper path.

How to Run Windows Containers Without Nested Virtualization

I tested this on a Windows Server 2022 VM of our own on the same kind of platform, then rolled the whole install back. It was a proof, not a permanent install.

Step 1: Confirm there is no hypervisor available inside the VM.

Run this in PowerShell on the Windows VM:

```
Get-CimInstance Win32_Processor | Select-Object VirtualizationFirmwareEnabled,
VMMonitorModeExtensions
```

If both say False, the VM cannot run Hyper-V, WSL2 or Docker Desktop. Stop trying to install them. That is not a bug in the installer; it is the platform telling you the truth.

Step 2: Install Docker Engine, not Docker Desktop.

Microsoft publishes a script that installs the Containers feature and the Docker Engine (CE) as a Windows service. Download it:

```
Invoke-WebRequest -UseBasicParsing "https://raw.githubusercontent.com/microsoft/Windows-
Containers/Main/helpful_tools/Install-DockerCE/install-docker-ce.ps1" -o install-docker-ce.ps1
```

Then run it from an elevated PowerShell. This changes the server: it enables a Windows feature and reboots.

```
.\install-docker-ce.ps1
```

After the reboot, check the service:

```
Get-Service docker
```

Plain takeaway: on a server without nested virtualization, Docker Engine is the only Docker you can install.

Step 3: Check the host build before you pull anything.

Process isolation shares the host kernel, so the image must be built for the exact same Windows build as the host. An ltsc2022 host runs only ltsc2022 images. Older images normally fall back to Hyper-V isolation — which this VM does not have.

Read the host build:

```
(Get-ItemProperty 'HKLM:\SOFTWARE\Microsoft\Windows NT\CurrentVersion').CurrentBuild
```

Windows Server 2022 answers 20348. Your images must match that build.

Step 4: Run a test container in process isolation.

```
docker run --rm --isolation=process mcr.microsoft.com/windows/nanoserver:ltsc2022 cmd /c echo
hello
```

This step is where most tutorials stop and where mine starts. On my test VM, this did not print "hello". It failed with:

```
hcsshim::ProcessBaseLayer ... Access is denied
```

Step 5: Fix the Access Denied that is not about permissions.

The cause was Microsoft Defender **Controlled Folder Access**. It blocked dockerd.exe from writing image layers to C:\ProgramData\docker\windowsfilter. The folder permissions were fine. Defender was the one saying no.

Check whether Controlled Folder Access is on:

```
Get-MpPreference | Select-Object EnableControlledFolderAccess
```

If it is on, allow the Docker daemon by path. This changes Defender policy:

```
Add-MpPreference -ControlledFolderAccessAllowedApplications "C:\Program Files\Docker\dockerd.
exe"
```

Then restart the service:

```
Restart-Service docker
```

Run the Step 4 test again. Now it prints hello.

This error is not a bug. It is the verification. Without the test container, the first person to see "Access is denied" would have been a developer on go-live day, and the obvious next guess would have been

NTFS permissions — the wrong place to look.

Step 6: Check an image's build before you deploy it.

For any Windows image the team brings, read its OS build:

```
docker image inspect --format "{{.Os}} {{.OsVersion}}" <image>
```

You want windows and a version that starts with 10.0.20348 on a Server 2022 host. Anything else will not start in process isolation.

So the platform was proven. Docker Engine, Windows containers, process isolation, no nested virtualization needed. Then the customer sent their images, and none of it mattered.

The Check That Made the Whole Question Disappear

The customer shared two image files, each a little over 200 MB, exported as tar archives. Before loading them anywhere, I read what was inside.

An exported image is a plain tar file. You do not need Docker to look into it. On any machine with tar — Windows 10 and later include it — list the files:

```
tar -tf image.tar
```

Read the manifest. It names the config file of the image:

```
tar -xOf image.tar manifest.json
```

Then read that config file, using the name the manifest gave you:

```
tar -xOf image.tar <config-file-name>.json
```

Look for two fields: "os" and "architecture".

Both images said "os": "linux" and "architecture": "amd64". The base was Debian 12. The app was .NET 8 on ASP.NET, listening on port 8080, built with Buildah — the Podman tool chain. Not a single Windows layer.

So the Windows VM could never run them. Not with Docker Desktop, not with Docker Engine, not with nested virtualization turned on and a perfect host build. A Linux image needs a Linux kernel, and on that server the only way to get one would have been a Linux VM inside the Windows VM — which needs the nested virtualization the platform did not offer.

The question was never "how do we get Docker on Windows". It was "why is this app on Windows at all".

The Two Honest Options

Because the app was .NET 8, which is cross-platform, there were two real paths:

1. **Give them a Linux VM.** The images run as they are. No rebuild, no change on the developer side, no platform ticket. This is what I recommended.

2. **Rebuild the images for Windows.** The team could rebase on a Windows .NET image such as mcr.microsoft.com/dotnet/aspnet:8.0-nanoserver-ltsc2022, then ship headless ltsc2022 images that match the host build. Possible, but it is real work for the developers, and it ties them to the host's Windows build for every future update.

Turning on nested virtualization was not on the list. It would have made Docker Desktop possible, but it would have meant a vendor ticket, a VM power-off, and a container stack running a Linux VM inside a Windows VM inside a hypervisor — three layers to run one Linux app.

The Checklist I Run Before Saying Yes to "Docker on Windows"

1. **What OS is the image?** tar -xOf image.tar manifest.json, then read "os" in the config file. If it says linux, stop and offer a Linux VM.
2. **Does the VM have a hypervisor to spare?** Get-CimInstance Win32_Processor — if VMMonitorModeExtensions is False, Docker Desktop and Hyper-V isolation are off the table.
3. **Is it Docker Engine, not Docker Desktop?** Get-Service docker shows a running service.
4. **Does the image build match the host build?** Host CurrentBuild against docker image inspect --format "{{.OsVersion}}".
5. **Is Controlled Folder Access in the way?** Get-MpPreference — if it is on, allow dockerd.exe before go-live, not after.
6. **Does a test container run in process isolation?** The nanoserver echo hello test, every time, on the real server.
7. **Who rebuilds images when the host is patched to a new build?** Name the team, in writing.

What I Tell Every Customer Now

I stopped asking "do you need Docker?" Everyone says yes.

I ask: "What does os say in your image's config file?"

If the answer is "windows, built for the same build as the server", they're in good shape. If the answer is "we just use containers", we're not done yet.

I write about the questions I ask in real customer rooms — the ones that change architectures and surface the problems no one else noticed. If you found this useful, drop your email below and I'll send the next one when it's ready.

FAQs

Can I run Docker Desktop on a Windows VM without nested virtualization?

No. Docker Desktop needs WSL2 or Hyper-V, and both need a hypervisor inside the VM. Without nested virtualization the VM cannot provide one. Install Docker Engine instead and run Windows containers in process isolation.

What is process isolation for Windows containers?

Process isolation runs Windows containers directly on the host's Windows kernel, the same way Linux containers share the Linux kernel. It needs no hypervisor, so it works on a VM without nested virtualization. The catch is that the image must be built for the same Windows build as the host.

Why does docker run fail with "hcsshim::ProcessBaseLayer Access is denied"?

On the server I tested, Microsoft Defender Controlled Folder Access blocked dockerd.exe from writing image layers to C:\ProgramData\docker\windowsfilter. Allow dockerd.exe with Add-MpPreference -ControlledFolderAccessAllowedApplications, restart the Docker service and run the test container again.

How do I check if a Docker image is Linux or Windows without Docker?

An exported image is a tar file. Read manifest.json with tar to find the config file name, then read that config file and look at the "os" and "architecture" fields. If "os" is linux, the image cannot run as a Windows container.

Can a Windows Server 2022 host run older Windows container images?

Not in process isolation. An ltsc2022 host runs only ltsc2022 images that way. Older images normally fall back to Hyper-V isolation, which needs a hypervisor, so on a VM without nested virtualization they will not start. Rebuild them on an ltsc2022 base image.

How do I enable nested virtualization on a KVM-based platform?

The VM must be powered off, its CPU mode set to host-passthrough, and the KVM nested option must be on for the host node. On most hyperconverged platforms that is a request to the platform team or vendor, not a setting in the VM console. Check first whether the app needs it at all.

Read online: https://quabin.com/docker-windows-vm-no-nested-virtualization/