Engineer

virtual machine

a whole computer run in software on a physical host

What it is

A virtual machine presents a whole computer through software: processors, memory, disks and network cards supplied by a virtualization layer on a physical host, with an operating system booting on them as if they were dedicated hardware. It is often shortened to VM. Some devices can be passed through from the host rather than emulated. NIST's guide to full virtualization, SP 800-125, defines it in one line as "A simulated environment created by virtualization." In the kind of virtualization that guide covers, one or more operating systems run, together with their programs, on virtual hardware, each in a VM of its own, where it is called a guest operating system.

Two words do most of the work around it. The real machine is the host and each virtual one is a guest. The program that divides the host between its guests is the hypervisor, also called the virtual machine monitor, which SP 800-125 describes as the component that manages the guest operating systems on a host and controls the flow of instructions between them and the physical hardware: processor, disks, memory and network cards. The same guide names the two things that make the arrangement useful. The hypervisor can split the host's resources so that each guest reaches its own allocated resources and any explicitly shared resources, and a guest can be wrapped up completely, which makes it portable from one host to another.

A virtual machine differs from a container in where the line is drawn. A container is a process on its host's kernel with a restricted view of the system. A virtual machine boots a kernel of its own on hardware that the hypervisor presents to it. That is why a Linux guest can run on a Windows or macOS host, and also why every guest needs an operating system of its own, installed, patched and given enough memory to run.

Use the concept in a decision

Put “What is a virtual machine?” to work in a real decision. We can help relate the concept to the system you are building or buying. Servers sized for what you actually use, moved into the cloud or back out of it, with the monthly bill worked out first.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Ran 20 Docker apps on Google Cloud and AWS — 20 containerised applications deployed and watched

Ran 20 Docker apps on Google Cloud and AWS Get in touch →

Why it matters

A server rented from a cloud provider is usually a virtual machine, and infrastructure as a service is built on them. The provider runs the hypervisors and the physical fleet; the customer gets machines that can be created, resized and destroyed through an API instead of being ordered, delivered and racked. The hardware stays someone else’s problem while the operating system becomes the customer’s.

Isolation is the second reason. The KVM project page says each virtual machine has private virtualized hardware, with its own network card, disk and graphics adapter. Guests share no kernel, so a kernel crash or a bad update inside one guest stays inside it, and it is the hypervisor, not the guests, that decides what reaches the physical hardware. Two customers, or two environments of one customer, can share a physical machine without sharing an operating system.

Third, with image-backed storage, a machine can become a set of files. A guest’s disk can be an image file on the host, which can be copied, kept as a template or used to start several identical machines. QEMU’s qcow2 format can record only the differences from a base image and can store snapshots of the disk inside the same file, so one template plus a thin layer per guest is a natural layout.

Fourth, emulation lets software built for one family of processors run on another. In QEMU’s system emulation, the guest operating system runs on a machine modelled in software, with its own processor, memory and devices. With an accelerator, such as KVM on Linux, Hypervisor Framework on macOS or Windows Hypervisor Platform on Windows, the host processor’s virtualization support does the heavy lifting. Without one, QEMU’s Tiny Code Generator translates the guest’s instructions in software, which is how the QEMU introduction’s own example boots an Arm64 machine with `-accel tcg` .

The cost is that every guest is a full computer to look after. Each carries its own operating system to patch, its own reserved memory and its own boot. Containers avoid that by sharing the host's kernel, which makes each one lighter and gives up a separate kernel as the wall between workloads. Choosing between them means choosing where that wall should stand.

What it looks like in practice

On x86 Linux the usual pair is KVM and QEMU. KVM is the kernel half: a loadable module, `kvm.ko`, with a processor-specific companion, `kvm-intel.ko` or `kvm-amd.ko`, included in mainline Linux since 2.6.20. It needs the processor's virtualization extensions, Intel VT or AMD-V. The KVM FAQ checks for their flags, `vmx` and `svm`, in `/proc/cpuinfo`, and its troubleshooting list includes confirming that `/dev/kvm` exists and that the user has the rights to open it:

```
grep -E -c '^flags.*(vmx|svm)' /proc/cpuinfo
ls -l /dev/kvm
```

A count above zero means the processor reports the extensions. QEMU is the user-space half, which builds the machine around KVM. The disk for a new guest can be a thin layer on a shared base image:

```
qemu-img create -f qcow2 -b debian-base.qcow2 -F qcow2 web1.qcow2
qemu-img info --backing-chain web1.qcow2
```

The qemu-img manual says an image created with a backing file records only the differences from it, and that the backing file is never modified unless those differences are committed back into it. The second command lists every file in the chain, which is worth reading before anything is moved or deleted. Starting the guest then looks like this:

```
qemu-system-x86_64 -accel kvm -m 2G -smp 2 \
    -drive file=web1.qcow2,if=virtio,format=qcow2 \
    -nographic
```

`-accel kvm` hands the processor work to KVM, through `/dev/kvm` by default. `-m` sets the guest's memory and `-smp` its number of processors. On `-drive`, `if=virtio` attaches the disk on the virtio interface, one of the interface types the manual lists, and `format=qcow2` names the format instead of leaving QEMU to detect it. `-nographic` turns off the graphical window, so QEMU becomes a plain command-line program with the guest's serial console in the terminal, shared with the QEMU monitor; Ctrl-a h shows how to switch between the two.

With the guest shut down, the same tool takes and restores snapshots of the disk. `-c` creates one, `-l` lists them and `-a` reverts the disk to a saved state:

```
qemu-img snapshot -c before-upgrade web1.qcow2
qemu-img snapshot -l web1.qcow2
qemu-img snapshot -a before-upgrade web1.qcow2
```

Sizing is a design decision rather than a form field. The portfolio entry on running the company's public services on one small server documents a September 2026 setup with one processor core, a 10 GB disk and 512 MB of memory, of which programs can use roughly 464 MB. That host serves seven domains over TLS and also runs the git forge, with backup automation implemented separately; memory is the limit every design choice there answers to. Whether it could take an onion service on Tor as well was decided by a recorded day of samples, in which the lowest reading of available memory was around 310 MB. Every guest's `-m` setting poses the same question, and a measurement answers it better than a bigger plan does.

The entry on managing thirty Ubuntu Linux VPS instances shows the other end of the scale. The setup of those instances had accumulated over the years instead of being designed, and their configurations had drifted apart. From then on they were run as a single fleet: one hardened network configuration on every instance, and restores that were tried out rather than taken on trust.

Where it goes wrong

Most trouble with virtual machines comes from forgetting that each one is a real computer to look after, and that an image-backed disk is also an ordinary file on another machine.

KVM is missing when the guest starts. In the x86 setup above, absent or firmware-disabled processor extensions prevent KVM acceleration and `/dev/kvm` may be absent, so a guest started with `-accel kvm` stops at once. The KVM FAQ warns that some manufacturers disable VT in the BIOS in a way that cannot be re-enabled. The fix is to run the flag check above, look for a firmware option that enables virtualization, and confirm that `/dev/kvm` exists and belongs to a group the user is in; the FAQ suggests a group called `kvm`.

A guest started without a memory size gets 128 MiB. That is QEMU's default for `-m`, and it applies whatever the host has free, so a guest can start, then fail in ways that look like bugs in the software inside it. The fix is to set `-m` and `-smp` every time and size them from measurement, the way the small-server entry settled its onion-service question with a day of measurements.

An image changed while the guest runs can be destroyed. The qemu-img manual is blunt: editing an image while a guest or another process has it in use can destroy that image. Taking a snapshot of a live guest's disk with `qemu-img snapshot -c` is exactly that. The fix is to shut the guest down before qemu-img touches its disk, and to make that a step in every runbook that does.

Snapshots stand in for backups. Snapshots made with qemu-img live inside the same image file, which `qemu-img info` shows by listing them alongside the disk. When the file is lost or damaged, every snapshot goes with it. The fix is to copy backups to storage on another machine and to test the restore, because a restore that has never been run proves nothing.

A base image changes or moves under its overlays. A thin image holds only differences, so it means nothing without the exact base it was made against. The manual warns that changing the backing file in unsafe mode corrupts the guest-visible content if the new file is the wrong one, and a relative backing path is resolved from the overlay's own directory, so moving one file without the other breaks the chain. The fix is to treat base images as read-only and give each new version a new name, to read `qemu-img info --backing-chain` before moving anything, and to run `qemu-img commit` only deliberately, since it writes into the base and empties the overlay.

The disk format is guessed from the image itself. Without `format=`, QEMU detects the format from the contents of the file, and the manual recommends `format=raw` to avoid interpreting an untrusted format header. An image

downloaded from elsewhere, or one a guest has written to, is untrusted in exactly that sense. The fix is to name the format on every `-drive` line.

Every guest drifts into a pet. Machines configured by hand drift apart one change at a time until no two are alike, which is where the VPS fleet above stood before it came under management. The fix is to build every guest from the same image and the same configuration, keep that configuration in version control, and manage the machines as one fleet with one baseline.

Further reading

- Guide to Security for Full Virtualization Technologies (NIST SP 800-125) — the VM definition, the hypervisor's job and how guests are kept apart.
- KVM — the project page: modules, processor extensions and mainline version.
- KVM FAQ — the vmx and svm check, `/dev/kvm` permissions and the BIOS setting.
- QEMU System Emulation introduction — the accelerators, the Tiny Code Generator and a full example command.
- QEMU invocation — every option used above: `-accel`, `-m`, `-smp`, `-drive`, `-nographic`.
- QEMU disk image utility — qcow2, backing files, snapshots and the warning about images in use.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a virtual machine?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/virtual-machine/