

Phiên 1 — Dụng VM lab & serial console

Một VM Ubuntu Server chạy từ cloud image, login được, và serial console cho thấy kernel log từ dòng đầu tiên. Kèm cơ chế bên dưới từng bước và những chỗ tài liệu phổ biến nói sai.

THỜI LƯỢNG	HOST	GUEST	MỐC
2 giờ, không lẩn giờ	Ubuntu Server, bare metal	Ubuntu 24.04 noble	M-L0 (tuần 1)
ARTIFACT			
Note tái tạo được lab			

Phiên này chỉ có ba việc: làm host đủ điều kiện chạy KVM, đẻ ra một VM từ cloud image, và nhìn được kernel log của nó qua cổng serial. Ba việc đó đơn giản tới mức có thể xong trong 20 phút bằng cách copy lệnh từ blog. Chỗ còn lại của hai tiếng là để hiểu vì sao mỗi lệnh tồn tại — vì phiên B1 sẽ cố tình phá initramfs, và lúc đó serial console là màn hình duy nhất còn nói chuyện với mình.

CÁCH DÙNG TRANG NÀY

Mỗi block có ba lớp: **cơ chế** (đọc trước khi gõ), **việc cần làm** (tự ghép lệnh), và một khối *Hiện lệnh đầy đủ* gộp lại. Mở spoiler **sau** khi đã thử, hoặc khi kẹt quá 5 phút. Mở ngay từ đầu thì phiên này biến thành copy-paste và bạn sẽ không nhớ gì vào tuần sau.

MỤC 00

Trước khi bấm giờ

Danh sách này là phần rút gọn của cấu hình tối thiểu. Chạy hết trước khi tính giờ phiên — nếu một trong số này hỏng, cả phiên hỏng theo và bạn sẽ debug BIOS thay vì học libvirt.

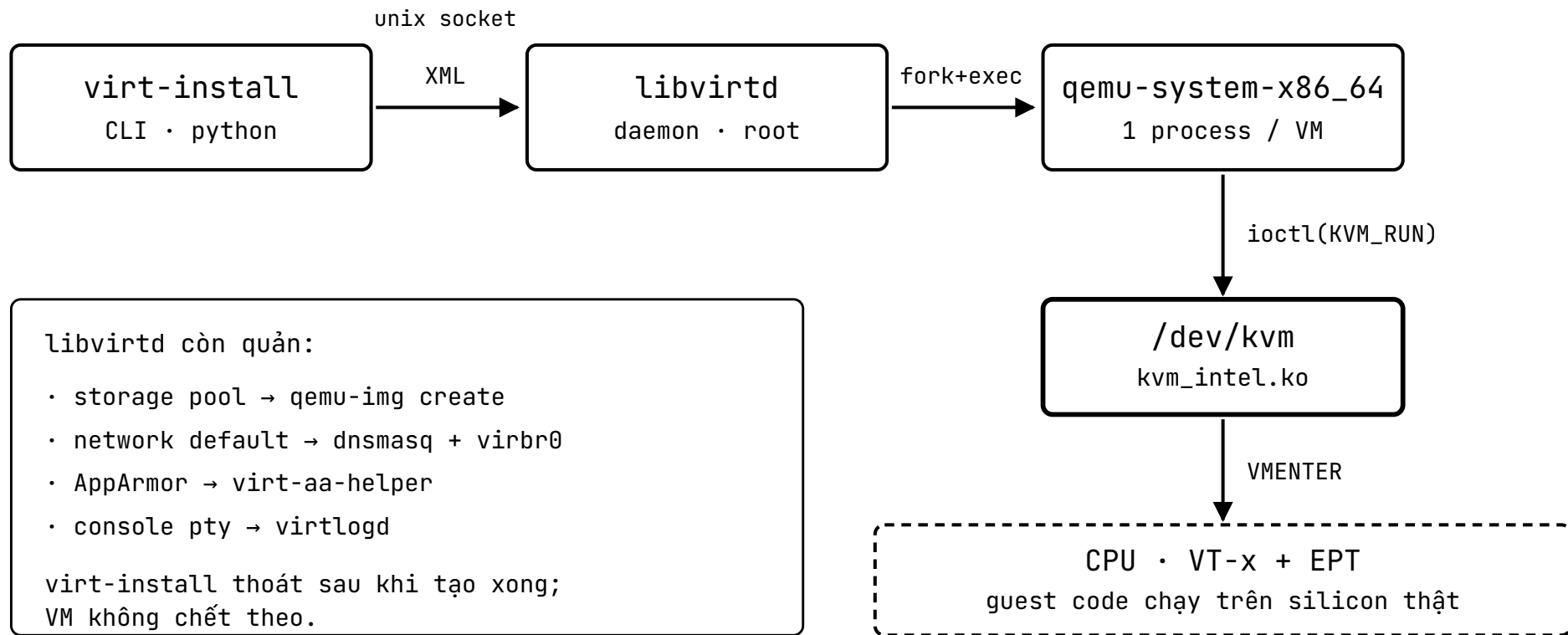
- ☐ Vào được host lab, và **mở `tmux` ngay** trước cả khi cài gói. Rớt mạng giữa lúc VM đang boot mà không có tmux là mất phiên console.
- ☐ `ls -l /dev/kvm` → file tồn tại. Không có nghĩa là VT-x chưa bật trong BIOS, hoặc module `kvm_intel` chưa load.
- ☐ `lscpu | grep -iE 'vmx|svm'` → thấy cờ virtualization. Nên có thêm `ept` (Intel) hoặc `npt` (AMD) — thiếu nó thì MMU virtualization chạy bằng shadow page table, chậm hơn nhiều.
- ☐ Đang là **user thường**, không phải root. Toàn bộ nội dung học của Block A nằm ở tầng quyền; làm bằng root là bỏ qua nó.
- ☐ `df -h /var/lib/libvirt` → còn ít nhất 10 GB. Cloud image ~600 MB, overlay lớn dần theo lượng ghi.

CHI TIẾT NHỎ, HẬU QUẢ LỚN

`/dev/kvm` tồn tại chưa đủ — phải **mở được**. Quyền mặc định là `root:kvm 0660`. Đây là lý do bước "thêm user vào group `kvm`" ở Block A không phải thủ tục hành chính: không có nó, QEMU rơi về chế độ TCG (emulation thuần phần mềm) hoặc từ chối chạy. Kiểm nhanh bằng `test -r /dev/kvm && test -w /dev/kvm && echo ok`.

Ngăn xếp ảo hoá: ai làm gì

Gõ `virt-install` xong thấy VM chạy, dễ tưởng `virt-install` là thứ chạy VM. Không phải. Có bốn thành phần tách bạch, và mỗi cái hỏng theo một kiểu riêng — biết ai làm gì là biết đọc thông báo lỗi.



Bốn tầng, ba ranh giới. Lỗi *permission denied* gần như luôn nằm ở ranh giới thứ nhất (client → daemon) hoặc thứ ba (QEMU → file trên đĩa), không bao giờ ở tầng CPU.

KVM	Module kernel (kvm.ko + kvm_intel.ko) phôi ra /dev/kvm. Nó <i>không</i> emulate thiết bị – chỉ cho phép chạy code guest ở chế độ non-root VMX và bắt các lần thoát (VM exit).	Không có /dev/kvm; VM chạy nhưng chậm thảm hại (rơi về TCG).
QEMU	Một process user-space cho mỗi VM. Emulate chipset, disk, NIC, serial port; mỗi vCPU là một thread gọi ioctl(KVM_RUN) trong vòng lặp.	Permission denied khi mở file disk; guest không thấy thiết bị nào đó.
libvirt	Daemon + thư viện quản lý. Giữ định nghĩa VM dạng XML, sinh dòng lệnh QEMU, quản storage pool, virtual network, và nhân bảo mật. Không phải hypervisor.	virsh không kết nối được; VM "biến mất" (thường là sai URI).
virt-install	Script Python dịch tham số CLI thành XML rồi đẩy qua libvirt. Chạy xong là thoát.	Lỗi cú pháp, thiếu --osinfo, seed ISO không sinh được.

THÚ VỊ

Cái tên `kvm_intel` đáng chú ý: KVM biến kernel Linux thành hypervisor *type-1-ish* mà không cần viết lại scheduler. Một VM là một process bình thường, các vCPU là các thread bình thường – nên `top` trên host thấy VM như mọi tiến trình khác, `nice` ăn được, cgroup giới hạn được. Đó cũng là lý do Phase 2 (memory/OOM) và Phase 4 (cgroup) áp thẳng lên VM chứ không cần kiến thức riêng.

Chuẩn bị host

Mục tiêu block: ba dòng nghiệm thu ở cuối đều đúng. Không phải "cài xong gói".

Ba vai trò, ba gói

Mọi distro đều có cùng ba vai trò; chỉ tên gói khác nhau. Nhìn theo vai trò thì chuyển distro không phải học lại:

VAI TRÒ	UBUNTU	ARCH
Hypervisor userspace	qemu-system-x86	qemu-desktop / qemu-base
Daemon quản lý	libvirt-daemon-system	libvirt
CLI client	virtinst, libvirt-clients	virt-install, libvirt
DHCP/DNS cho NAT network	kéo theo dnsmasq-base	dnsmasq (phải cài tay)
Cơ sở dữ liệu OS metadata	osinfo-db	osinfo-db

BÃY · OSINFO-DB QUÁ CŨ

`osinfo-db` trong pocket `noble` gốc là bản `0.20231215` – build tháng 12/2023, tức **trước khi 24.04 ra đời**, nên không thể chứa entry `ubuntu24.04`. Bản vá nằm ở `noble-updates` (`0.20250606`). Nếu `--osinfo ubuntu24.04` báo unknown OS, thủ phạm là đây chứ không phải bạn gõ sai. Kiểm bằng `osinfo-query os | grep 24.04`.

Daemon: monolithic hay modular

CHỖ RẤT NHIỀU TÀI LIỆU NÓI SAI

Vô số blog – và một phần tài liệu chính hãng đọc vội – nói Ubuntu 24.04 trở đi dùng **libvirt modular daemon**. Sai. Debian và Ubuntu – kiểm tới tận 26.04 (resolute) – *không build* các daemon con: gói `libvirt-daemon` chỉ chứa `libvirtd`, `virtlogd`, `virtlockd`. Không có binary `virtqemud`, không có unit `virtqemud.socket`, không có manpage `virtqemud(8)`. Trên host Ubuntu, `systemctl status libvirtd` **chính là** cả câu chuyện.

Modular daemon là mặc định ở **Fedora 35+**, **RHEL 9+**, **openSUSE** và **Arch**. Nghĩa là khác biệt giữa hai họ distro có *thật* và đáng học – chỉ ngược chiều với những gì phần lớn tài liệu ghi.

Vẫn nên hiểu kiến trúc modular – nó là mặc định ở nửa còn lại của thế giới distro, và mình hoạ một pattern systemd đáng học:

- libvirt 5.7.0 (09/2019) tách `libvirtd` nguyên khối thành các daemon theo driver: `virtqemud`, `virtnetworkd`, `virtstoraged`, `virtnodedevd`, `virtsecret`, `virtnwfilterd`, `virtinterfaced`, cộng `virtproxyd` cho truy cập từ xa.
- Mỗi daemon system-mode phối **ba socket**: `virtqemud-sock` (đọc-ghi), `-sock-ro` (chỉ đọc), `-admin-sock`. Tương ứng ba unit `.socket`.
- Chúng chạy ở chế độ **socket activation**: systemd giữ socket, daemon chỉ được khởi động khi có client kết nối, và tự tắt sau một khoảng nhàn rỗi. Hệ quả trực tiếp: `systemctl restart virtqemud` gần như vô nghĩa – thứ cần enable/start là `.socket`, không phải `.service`. Và phần lớn option socket trong `virtqemud.conf` hết tác dụng vì systemd mới là bên tạo socket.
- Session mode chỉ có **hai** socket, đặt trong `$XDG_RUNTIME_DIR/libvirt/`. Không có socket read-only, vì client và daemon cùng UID nên chế độ chỉ-đọc không mang lại lợi ích bảo mật nào.

Cách phân biệt host đang ở chế độ nào – dùng đúng lệnh này khi so sánh hai máy:

KIỂM CHẾ ĐỘ DAEMON	HOST BẤT KỲ
<pre>\$ systemctl is-active virtqemud.socket # modular, ngay bây giờ \$ systemctl is-active libvirtd.socket # monolithic, ngay bây giờ \$ systemctl is-enabled virtqemud.socket # modular, ở lần boot sau \$ systemctl list-units 'virt*' 'libvirt*' \$ ls -l /run/libvirt/*-sock*</pre>	

Group, quyền, và cái URI hay giết người

Hai group cần có: `libvirt` (được nói chuyện với socket của daemon) và `kvm` (được mở `/dev/kvm`).

VÌ SAO PHẢI LOGOUT/LOGIN LẠI

Danh sách supplementary group được kernel gán vào **credential của process** lúc `setgroups()` chạy – tức là lúc đăng nhập. Shell đang mở đã mang bản chụp cũ, và mọi process con thừa kế bản chụp đó. Sửa `/etc/group` không hồi tố. Kiểm chứng ngay: `id` (đọc từ credential process) so với `getent group libvirt` (đọc từ database). Hai cái lệch nhau chính là bằng chứng. Cần lách một lần: `exec sg libvirt newgrp kvm` hoặc mở lại ssh – và với ssh, **đóng cả tmux server** mới thật sự sạch, vì tmux server cũ vẫn giữ credential cũ.

`qemu:///system` vs `qemu:///session`

Đây là bẫy tốn thời gian nhất của cả block. Hai URI là **hai thế giới VM tách rời** – không nhìn thấy nhau, không dùng chung storage, không dùng chung network.

	QEMU:///SYSTEM	QEMU:///SESSION
Daemon	libvirtd chạy bằng root	một instance riêng, cùng UID với bạn
Socket	/run/libvirt/libvirt-sock	\$XDG_RUNTIME_DIR/libvirt/...
QEMU chạy dưới	libvirt-qemu:kvm	chính user của bạn
Config driver	/etc/libvirt/qemu.conf	~/.config/libvirt/qemu.conf
Storage mặc định	/var/lib/libvirt/images	~/.local/share/libvirt/images
Network	virtual network thật: virbr0 + NAT + dnsmasq	chỉ userspace: SLIRP hoặc passt – không có virbr0
AppArmor sVirt	có, profile riêng mỗi VM	không

Khi truyền URI rỗng, libvirt chọn theo thứ tự: **(1)** biến môi trường `LIBVIRT_DEFAULT_URI` → **(2)** `uri_default` trong `~/.config/libvirt/libvirt.conf` (root đọc `/etc/libvirt/libvirt.conf`) → **(3)** dò lần lượt các driver cho tới khi kết nối được. Bước (3) là chỗ user thường hay rơi vào `session` mà không biết.

Chốt một URI ngay từ đầu phiên – khuyến nghị `system`, vì cả lộ trình về sau (snapshot, network, cgroup) đều cần nó – và đặt nó ở chỗ bền, không phải gõ tay mỗi shell.

GHI CHÚ THIẾT KẾ

Session mode không phải phiên bản "kém hơn" – nó tồn tại cho desktop virtualization, nơi không muốn daemon root. Điểm yếu cố hữu là mạng: không có quyền tạo bridge nên phải dùng ngăn xếp TCP/IP trong userspace. SLIRP là bản cũ và đã ngừng bảo trì; libvirt 9.0.0 (01/2023) thêm hỗ trợ **passt**, hiện là lựa chọn được khuyến nghị cho VM rootless. Không dùng cho lab này, nhưng đáng nhớ tên.

Network `default`

libvirt cài kèm một virtual network tên `default`: bridge `virbr0` ở `192.168.122.1/24`, NAT ra ngoài, và **một instance dnsmasq riêng cho mỗi network** làm DHCP + DNS. Bridge này cố ý *không* gắn card vật lý nào – nó không phải bridge L2 nối vào LAN, mà là một mạng nội bộ được NAT.

- Định nghĩa persistent nằm ở `/etc/libvirt/qemu/networks/default.xml`; autostart thể hiện bằng **symlink** trong `.../networks/autostart/`. Đừng sửa tay – dùng `virsh net-edit`.
- libvirt lái dnsmasq bằng `--dhcp-script` trỏ vào `libvirt_leaseshelper` để tự giữ lease database: `/var/lib/libvirt/dnsmasq/default.leases`.
- Trên **Ubuntu**, network này thường đã `active` + `autostart` sẵn sau khi cài – nhưng phải xác nhận, đừng giả định. Trên **Arch** thì thường chưa, phải `net-start` và `net-autostart` bằng tay.

Nghiệm thu Block A

- ☐ `/dev/kvm` tồn tại và user hiện tại đọc-ghi được.
- ☐ `virsh uri` trả về `qemu:///system` – không phải `session`.
- ☐ `virsh net-list --all` cho thấy `default` đang `active` và `autostart`.

■ HIỆN LỆNH ĐẦY ĐỦ · BLOCK A (UBUNTU)

CHUẨN BỊ HOST

HOST LAB · TRONG TMUX

```
# 1. gói – ba vai trò + osinfo-db mới
$ sudo apt update
$ sudo apt install -y qemu-system-x86 libvirt-daemon-system \
    libvirt-clients virtinst osinfo-db qemu-utils

# 2. group: libvirt cho socket, kvm cho /dev/kvm
$ sudo usermod -aG libvirt,kvm "$USER"
$ getent group libvirt kvm          # database đã có tên mình
$ id                               # process thì CHƯA – cần login lại

# thoát hẳn: kill tmux server rồi ssh lại
$ tmux kill-server; exit
$ ssh <host-lab> && tmux new -s lab
$ id                               # giờ mới thấy libvirt, kvm

# 3. chốt URI cho mọi shell về sau
$ mkdir -p ~/.config/libvirt
$ echo 'uri_default = "qemu:///system"' > ~/.config/libvirt/libvirt.conf
$ virsh uri                        # ⇒ qemu:///system

# 4. daemon: Ubuntu là monolithic, không có virtqemud
$ systemctl is-active libvirtd.socket
$ systemctl list-units 'libvirt*' 'virt*'
$ systemctl is-active virtqemud.socket # ⇒ not-found. Đúng như dự đoán.

# 5. network default
$ virsh net-list --all
# nếu chưa active:
$ virsh net-start default && virsh net-autostart default
$ ip -br addr show virbr0          # ⇒ 192.168.122.1/24
$ ls -l /etc/libvirt/qemu/networks/autostart/

# 6. nghiệm thu
$ test -r /dev/kvm && test -w /dev/kvm && echo "kvm ok"
$ virsh uri
$ virsh net-list --all
$ osinfo-query os | grep -i '24\.'04'
```

Nhánh Arch – chỉ khác ở bước 1, 4, 5:

KHÁC BIỆT ARCH

HOST ARCH

```
$ sudo pacman -S qemu-desktop libvirt virt-install dnsmasq osinfo-db

# Arch dùng modular daemon → enable .socket, không phải .service
$ sudo systemctl enable --now virtqemud.socket virtnetworkd.socket \
    virtstoraged.socket virtnodedevd.socket
```

```
# network default hầu như luôn phải bật tay
$ sudo virsh net-start default
$ sudo virsh net-autostart default
```

MỤC 02

Cloud image & cloud-init: cơ chế

Toàn bộ triết lý "VM rẻ, phá thoải mái" của Phase 0 đứng trên hai thứ này. Đọc kỹ mục này thì Block B chỉ còn là gõ lệnh; đọc lướt thì sẽ mất 20 phút debug một VM treo ở màn hình login.

Vì sao cloud image, không phải ISO

Cài từ ISO nghĩa là chạy installer: phân vùng, cài gói, cấu hình – 10–20 phút, và *tương tác*. Cloud image là một đĩa đã cài sẵn: boot phát chạy luôn, khoảng 30 giây. Mục tiêu P2 là tái tạo lab dưới 2 phút – con số đó chỉ khả thi với cloud image.

Cái giá phải trả: image giống hệt nhau cho mọi người tải về, nên nó **không có user nào đăng nhập được**. Ubuntu cloud image có sẵn user `ubuntu` nhưng `lock_passwd: True` và không có khoá SSH nào. Phần "cá nhân hoá" được đẩy sang lần boot đầu tiên – và đó chính là việc của cloud-init.

HAI KÊNH PHÁT HÀNH, ĐỪNG NHẦM

cloud-images.ubuntu.com/noble/current/ là **daily build**, tên file `noble-server-cloudimg-amd64.img`. .../releases/noble/release/ là **release build**, tên file `ubuntu-24.04-server-cloudimg-amd64.img`. Lab nên dùng release build: nó cố định, còn daily thay đổi mỗi ngày và bạn sẽ không tái tạo được kết quả cũ. Đuôi `.img` gây hiểu nhầm – nội dung thật sự là **qcow2**. Xác nhận bằng `qemu-img info`, đừng tin phần mở rộng.

cloud-init lấy cấu hình từ đâu: datasource NoCloud

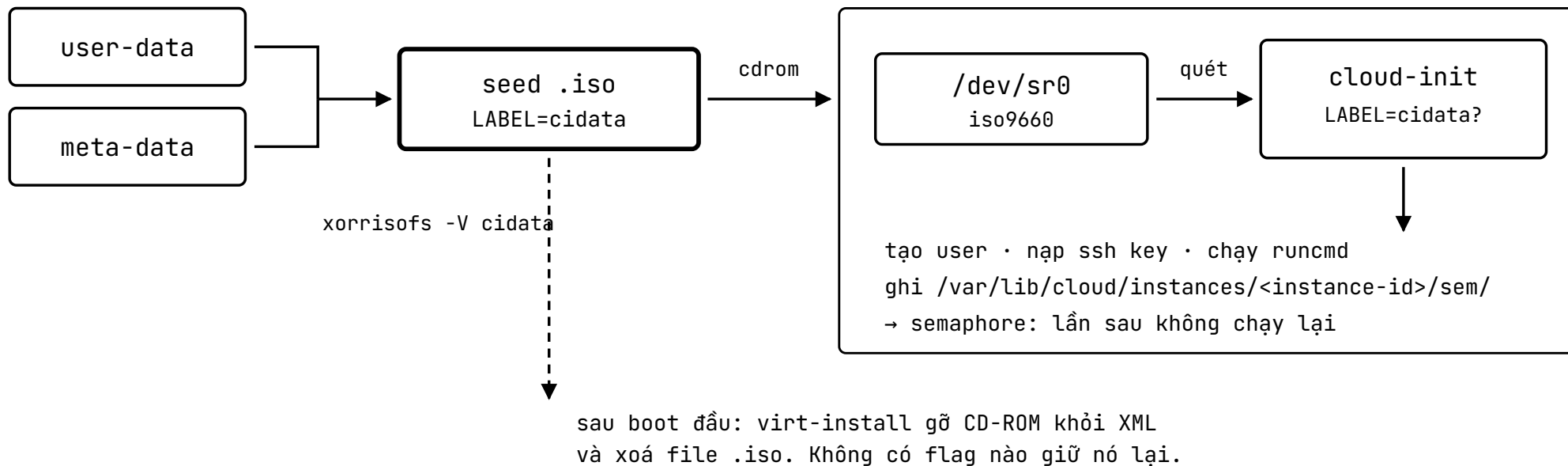
cloud-init hỗ trợ nhiều *datasource*: EC2 metadata service, Azure IMDS, OpenStack... Lab local dùng **NoCloud**, và cơ chế của nó thô sơ đến mức dễ hiểu:

1. Trên host, hai file văn bản `user-data` và `meta-data` được đóng thành một **ảnh đĩa iso9660 (hoặc vfat) có nhãn volume là cidata**.
2. Ảnh đĩa đó được gắn vào VM như một CD-ROM.
3. Trong guest, cloud-init quét mọi block device kiểu `vfat` hoặc `iso9660`, tìm cái có `LABEL=cidata` (khớp không phân biệt hoa thường), mount lên và đọc hai file.

Không có API, không có mạng, không có magic. Chỉ là một cái CD chứa file text. Xác minh trong guest bằng `lsblk -f` và `blkid` ở lần boot đầu.

TRÊN HOST

TRONG GUEST · BOOT ĐẦU



Đường đi của user-data: file text trên host → ảnh iso9660 nhãn `cidata` → CD-ROM của guest → cloud-init đọc ở boot đầu → semaphore chặn lần sau.

Vì sao chỉ có tác dụng ở lần boot đầu tiên

Hai cơ chế xếp chồng, và cả hai đều phải hiểu vì mỗi cái hỏng theo một kiểu:

Một – semaphore theo instance. Mỗi module cloud-init có một *frequency*. Sau khi chạy, module để lại file dấu trong `/var/lib/cloud/instances/<instance-id>/sem/`. Boot sau thấy dấu thì bỏ qua.

FREQUENCY	CHẠY KHI NÀO	MODULE TIÊU BIỂU
PER_INSTANCE	một lần cho mỗi instance-id	users_groups, ssh, set_passwords, runcmd, package_update...
PER_ALWAYS	mỗi lần boot	bootcmd, growpart, resizefs, final_message
PER_ONCE	đúng một lần cho cả đời máy, kể cả khi instance-id đổi	scripts_per_once

Hai – virt-install gỡ hẳn seed ISO. Khi dùng `--cloud-init`, virt-install dựng hai bản XML: bản đầu có CD-ROM seed, bản sau không. Sau khi guest tắt/reboot lần đầu, nó define lại bằng bản sau và **xoá luôn file ISO trên host**. Không có option nào giữ lại. Nên kể cả nếu bạn đổi `instance-id`, đĩa dữ liệu đã biến mất.

BẦY SÂU · INSTANCE-ID LUÔN LÀ "NOCLOUD"

`virt-install --cloud-init` sinh file `meta-data` **rỗng** nếu bạn không tự truyền. NoCloud khi đó rơi về mặc định `instance-id: nocloud` – một hằng số. Hệ quả: nếu bạn tái dùng cùng một đĩa và đưa seed mới vào, cloud-init coi đó là *cùng một instance* và bỏ qua toàn bộ module PER_INSTANCE. Sửa user-data rồi reboot là vô nghĩa.

Quy trình đúng khi sai user-data: `virsh destroy` + `virsh undefine --remove-all-storage`, rồi tạo lại từ đầu. Chính vòng lặp xoá-tạo này là thứ P2 sẽ script hoá thành `lab-reset.sh` – nên hãy để ý mình gỡ những gì.

Cách chủ động hơn: tự viết `meta-data` có `instance-id` khác nhau mỗi lần dựng. Vài dòng, nhưng nó biến "xoá VM tạo lại" từ bắt buộc thành lựa chọn.

CÒN MỘT ĐƯỜNG DỰ PHÒNG NỮA

`virt-install` cũng set SMBIOS system serial thành `ds=nocloud`. Đây là kênh thứ hai để cloud-init nhận ra datasource kể cả khi tra nhãn thất bại – `ds-identify` đọc chuỗi này từ DMI. Muốn thấy tận mắt trong guest: `cat /sys/class/dmi/id/product_serial`.

Năm giai đoạn boot của cloud-init

Biết tên giai đoạn là biết đọc log khi có sự cố – và `cloud-init analyze blame` sẽ nói cho bạn giai đoạn nào ăn hết thời gian boot.

GIAI ĐOẠN	SYSTEMD UNIT	LÀM GÌ
Detect	ds-identify	đoán datasource, dừng sớm nếu không có
Local	cloud-init-local.service	chạy <i>trước khi có mạng</i> , dựng cấu hình network

GIẢI ĐOẠN	SYSTEMD UNIT	LÀM GÌ
Network	cloud-init-network.service	đã có mạng: users_groups, ssh, set_passwords, mount, growpart
Config	cloud-config.service	apt, ntp, locale, runcmd
Final	cloud-final.service	cài gói, script, final_message

Log để đọc trong guest: `/var/log/cloud-init.log` (chi tiết) và `/var/log/cloud-init-output.log` (stdout/stderr của các lệnh). Trạng thái: `cloud-init status --long`.

BLOCK B · ~35 PHÚT

Tạo VM từ cloud image

Overlay qcow2: đừng ghi bản image gốc

Boot VM thẳng trên file `.img` vừa tải sẽ ghi vào nó — cloud-init chạy, log ghi, package cài. Mọi VM tạo sau từ file đó đều thừa kế rác, và bạn sẽ không hiểu vì sao "VM mới" lại có sẵn user cũ.

Đúng cách: image gốc chỉ-đọc, mỗi VM một **overlay** — file qcow2 riêng chỉ chứa phần *khác biệt*, trở ngược về gốc qua `backing_file`. Đọc block chưa từng ghi thì đi xuống backing; ghi thì copy-on-write lên overlay. Overlay của một VM vừa tạo chỉ vài trăm KB.

BẦY · THIẾU `-F` LÀ LỖI CỨNG

Từ **QEMU 6.1** (08/2021), tạo image có backing file mà không khai báo định dạng của nó là **lỗi**, không còn là cảnh báo (deprecated từ 5.1). Lý do là bảo mật: nếu QEMU phải tự đoán định dạng, một guest ghi được vào ảnh đĩa có thể khiến lần đoán sau ra kết quả khác — dữ liệu hỏng, hoặc tệ hơn là một đường thoát.

Cú pháp đúng: `qemu-img create -f qcow2 -F qcow2 -b BASE OVERLAY 20G`. Lưu ý tài liệu chính thức của `qemu-img` có *lỗi in*: phần synopsis ghi `-B BACKING_FMT` trong khi CLI thật nhận `-F`. Tin `qemu-img create --help`, đừng tin trang docs.

Với `virt-install` thì không gọi `qemu-img` trực tiếp — nó dựng XML storage volume và để libvirt gọi hộ:

HAI CÁCH, CÙNG KẾT QUẢ

HOST LAB

```
# cách thủ công — nhìn thấy rõ chuyện gì xảy ra
$ qemu-img create -f qcow2 -F qcow2 \
    -b /var/lib/libvirt/images/noble-base.img \
    /var/lib/libvirt/images/lab01.qcow2 20G
$ qemu-img info --backing-chain /var/lib/libvirt/images/lab01.qcow2

# cách qua virt-install — LUÔN khai backing_format
--disk size=20,backing_store=/var/lib/libvirt/images/noble-base.img,backing_format=qco
```

Vì sao "luôn khai": nếu thiếu `backing_format`, `virt-install` cố tự dò bằng cách tra file gốc trong các storage pool của libvirt. Không tìm thấy thì nó trả về `None` và XML thiếu hẳn phần định dạng. Khai tay thì không bao giờ phụ thuộc vào chuyện đó.

Viết `user-data`

File YAML, **dòng đầu bắt buộc** là `#cloud-config` – không phải comment, mà là chữ ký nhận dạng định dạng. Thiếu nó, `cloud-init` im lặng bỏ qua cả file.

Tối thiểu cho lab này: một user có sudo không cần password, và khoá SSH công khai của bạn. Password thì tùy – nhưng nếu định login qua `virsh console` thì phải có, vì console không dùng SSH key.

NÊN NHỚ

Trong `cloud-init`, khai `users:` mà không có mục `default` sẽ **thay thế** user mặc định `ubuntu` chứ không bổ sung vào. Muốn giữ cả hai thì phần tử đầu tiên của danh sách phải là chuỗi `default`. Đây là nguồn của rất nhiều câu hỏi "vì sao mất user ubuntu".

Chạy `virt-install`

Vài tham số đáng giải thích trước khi gõ:

- `--import` – nói rằng đĩa đã cài sẵn OS, không có install media. Thiếu nó, `virt-install` sẽ đòi `--location` hoặc `--cdrom`.
- `--osinfo ubuntu24.04` – **bắt buộc**. Từ `virt-manager` 4.0.0 (03/2022), thiếu tên OS là *lỗi cứng* với mọi guest HVM, không còn là cảnh báo. Thông báo lỗi nói thẳng lý do: không có nó thì không chọn được virtio, machine type q35, hay các mặc định hiện đại – VM sẽ chạy với đĩa SATA và NIC e1000 emulate.
- `--graphics none --console pty,target_type=serial` – bỏ màn hình đồ họa, chỉ để serial. Đúng thứ cần cho Block C.
- `--noautoconsole` – không tự nhảy vào console. Nên bật cho lần đầu để còn nhìn output của `virt-install`; Block C sẽ attach thủ công.

Lấy IP của VM

`virsh domifaddr` có ba nguồn dữ liệu khác nhau, và biết chọn đúng nguồn tiết kiệm rất nhiều bực bội:

NGUỒN	ĐỌC TỪ ĐÂU	GIỚI HẠN
<code>--source lease</code> (mặc định)	lease database của libvirt	chỉ thấy NIC nằm trên virtual network do libvirt cấp DHCP. IP tĩnh: không thấy.
<code>--source agent</code>	hỏi <code>qemu-guest-agent</code> trong guest	cần cài agent trong guest và có <code><channel></code> virtio trong XML. Đổi lại: thấy IP tĩnh, IPv6, nhiều địa chỉ trên một NIC.
<code>--source arp</code>	bảng neighbour của host	chỉ có sau khi guest đã nói chuyện với host ít nhất một lần.

Tương đương ở tầng thấp hơn: `virsh net-dhcp-leases default`, hoặc đọc thẳng `/var/lib/libvirt/dnsmasq/default.leases`.

Nghiem thu Block B

- ☐ `virsh list` thấy VM ở trạng thái `running`.
- ☐ Login được bằng user mình khai trong `user-data` (console hoặc SSH).
- ☐ Trả lời được: file `qcow2` nằm ở đâu, và quan hệ của nó với image gốc là gì.

■ HIỆN LỆNH ĐẦY ĐỦ · BLOCK B

TÀI VÀ BẢO VỆ IMAGE GỐCHOST LAB

```
$ cd /var/lib/libvirt/images
$ sudo wget -O noble-base.img \
    https://cloud-images.ubuntu.com/releases/noble/release/ubuntu-24.04-server-cloudimg-amd64.img

# xác minh – đừng bỏ qua, một image hỏng sẽ tốn cả phiên để đoán
$ wget -q https://cloud-images.ubuntu.com/releases/noble/release/SHA256SUMS
$ sha256sum noble-base.img
$ grep -i 'server-cloudimg-amd64.img' SHA256SUMS

$ qemu-img info noble-base.img    # file format: qcow2 (dù đuôi là .img)
$ sudo chmod 0444 noble-base.img  # chỉ-đọc: chống ghi bản
```

USER-DATA + META-DATAHOST LAB · ~/LAB/

```
$ mkdir -p ~/lab && cd ~/lab

$ cat > user-data <<'EOF'
#cloud-config
users:
  - default                                # giữ lại user 'ubuntu'
  - name: lab
    groups: [sudo, adm]
    shell: /bin/bash
    sudo: "ALL=(ALL) NOPASSWD:ALL"
    lock_passwd: false
    ssh_authorized_keys:
      - ssh-ed25519 AAAA...ĐÁNH_KHOÁ_CÔNG_KHAI_CỦA_BẠN_VÀO_ĐÂY

chpasswd:
  expire: false
  users:
    - name: lab
      password: "labpass"                # chỉ để login qua virsh console
      type: text

ssh_pwauth: false                        # SSH vẫn chỉ nhận key
package_update: true
EOF

# meta-data: instance-id khác nhau mỗi lần dựng
$ cat > meta-data <<EOF
```

```
instance-id: lab01-$(date +%s)
local-hostname: lab01
EOF
```

TẠO VM

HOST LAB

```
$ virt-install \
  --name lab01 \
  --osinfo ubuntu24.04 \
  --memory 2048 --vcpus 2 \
  --disk size=20,backing_store=/var/lib/libvirt/images/noble-base.img,backing_for
  --cloud-init user-data=$HOME/lab/user-data,meta-data=$HOME/lab/meta-data \
  --network network=default,model=virtio \
  --graphics none \
  --console pty,target_type=serial \
  --import \
  --noautoconsole

$ virsh list --all
$ virsh domblklist lab01          # đường dẫn qcow2 thật
$ sudo qemu-img info --backing-chain /var/lib/libvirt/images/lab01.qcow2

# lấy IP (chờ ~30s cho cloud-init xong)
$ virsh domifaddr lab01
$ virsh net-dhcp-leases default
$ ssh lab@192.168.122.X
```

Nếu user-data sai và cần làm lại — đây chính là vòng lặp mà P2 sẽ script hoá:

XOÁ SẠCH, LÀM LẠI

HOST LAB

```
$ virsh destroy lab01
$ virsh undefine lab01 --remove-all-storage
$ virsh list --all          # sạch chưa?
```

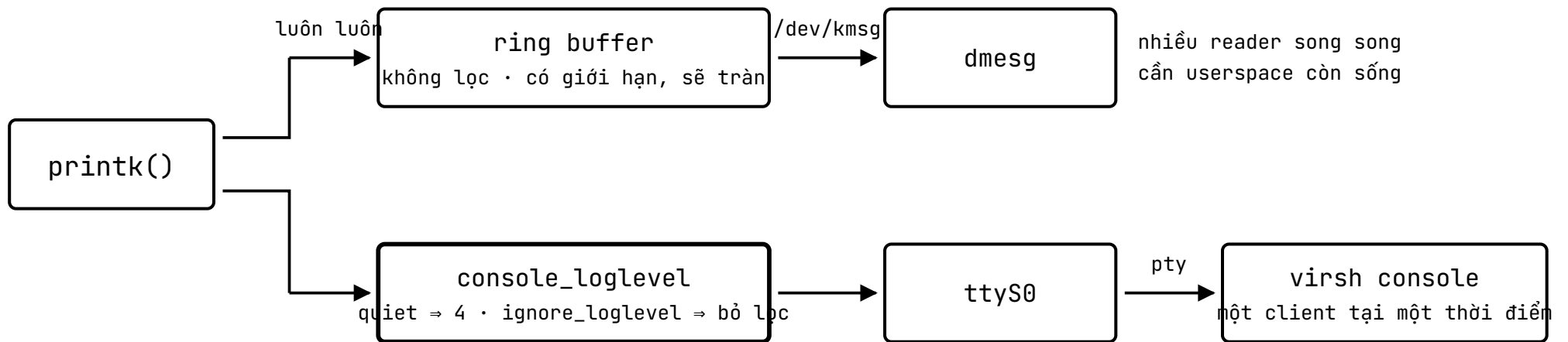
MỤC 03

Console & kernel log: cơ chế

Đây là phần có giá trị dài hạn nhất của cả phiên. Học nó lúc VM còn khoẻ, để lúc VM không boot được vẫn còn đường nhìn vào.

Ring buffer và console là hai vôi khác nhau

`dmesg` và serial console *cùng nguồn nhưng khác đích*. Mọi `printk()` luôn ghi vào **ring buffer** của kernel — không lọc. Song song đó, cùng lời gọi ấy ghi ra *từng console đã đăng ký* — nhưng chỉ khi vượt qua bộ lọc `loglevel`.



Panic => dmesg không chạy được nữa, nhưng byte đã ra khỏi cổng serial rồi. Đó là toàn bộ lý do serial console tồn tại.

Cùng một `printk()`, hai đích. Ring buffer nhận tất cả nhưng sẽ tràn và cần userspace để đọc; console nhận bản đã lọc nhưng là bản ghi nối tiếp, sống sót qua panic.

Cái nào thấy được gì mà cái kia không:

- **Chỉ `dmesg` thấy:** mọi thứ bị `console_loglevel` lọc bỏ — với `quiet` thì đó là toàn bộ mức 4–7. Và những dòng in *trước khi* console nào được đăng ký (đó chính là khoảng mù mà `earlycon` sinh ra để bịt).
- **Chỉ console thấy:** phần đã bị tràn khỏi ring buffer (buffer có kích thước cố định, `CONFIG_LOG_BUF_SHIFT`). Và — quan trọng nhất — **panic hoặc treo cứng:** lúc đó không còn userspace để chạy `dmesg`, nhưng byte đã đi ra cổng serial từ trước. Cộng thêm output của GRUB và firmware, thứ chưa bao giờ vào ring buffer.

CON SỐ CỤ THỂ

`cat /proc/sys/kernel/printk` trả về bốn số, ví dụ `4 4 1 7`: **hiện tại · mặc định · tối thiểu · mặc định lúc boot**. Số đầu là `console_loglevel` và sửa được ngay lập tức bằng `dmesg -n 7`. Tham số boot `quiet` không làm gì bí ẩn cả — nó chỉ tương đương `LogLevel=4` (giá trị lấy từ `CONFIG_CONSOLE_LOGLEVEL_QUIET`). Ring buffer không bị đung tới, nên `dmesg` sau một lần boot `quiet` vẫn đầy đủ.

console= nói với ai, và luật nhiều console

`console=ttyS0` là chỉ thị cho **kernel của guest**, viết ở dòng lệnh boot: “hãy in log ra cổng serial này”. Nó không liên quan gì tới host, tới libvirt, hay tới QEMU.

Cú pháp đầy đủ là `ttyS0,115200n8`, trong đó `bbbbpnf` = baud, parity, số bit dữ liệu, flow control. Mặc định của kernel là `9600n8`.

ĐÍNH CHÍNH MỘT HIỂU NHẦM PHỔ BIẾN

Ai cũng đọc `115200n8` thành “8N1”. Nhưng chuỗi này **không có trường stop bit** — `n` là parity=none và `8` là số bit dữ liệu. Con số 1 trong “8N1” đến từ quy ước nói chuyện của dân serial, không phải từ cú pháp kernel.

Khi có nhiều `console=` — và cloud image Ubuntu *luôn* có hai — luật là:

- Mỗi loại thiết bị chỉ xuất hiện một lần: **tất cả** đều nhận kernel message, và **cái cuối cùng** trở thành `/dev/console` cho userspace.
- Cùng một loại xuất hiện nhiều lần thì luật đổi hẳn: chỉ *cái đầu tiên* của mỗi loại nhận output, và `/dev/console` gắn với thiết bị *đăng ký sớm nhất* — mà thứ tự đăng ký phụ thuộc vào thứ tự kernel khởi tạo từng subsystem. Tài liệu kernel nêu thẳng ví dụ: hai dòng lệnh khác thứ tự vẫn cho kết quả giống hệt nhau, vì console đồ họa luôn được đăng ký trước serial.

Trong cloud image Ubuntu, dòng đó nằm ở đâu

Không phải `/etc/default/grub`. Ảnh cloud image được dựng bởi `livecd-rootfs`, và nó ghi một drop-in riêng:

TRONG GUEST

TRONG GUEST

```
$ cat /etc/default/grub.d/50-cloudimg-settings.cfg
GRUB_RECORDFAIL_TIMEOUT=0
GRUB_TIMEOUT=0
```

```
GRUB_CMDLINE_LINUX_DEFAULT="console=tty1 console=ttyS0"
GRUB_TERMINAL=console
```

Thứ tự là cố ý. `tty1` đứng trước để console đồ hoạ của hypervisor (vSphere, virt-manager) vẫn thấy log; `ttyS0` đứng sau để nó giành được `/dev/console`. Quyết định này có hồ sơ. Trên arm64 thì giá trị là `ttyAMA0` vì máy ảo QEMU virt dùng UART PrimeCell của ARM chứ không phải 8250.

Login prompt là chuyện hoàn toàn khác

Đây là chỗ hầu hết mọi người nhập nhầm. `console=` chỉ mang lại *output của kernel*. Dấu nhắc đăng nhập là một tiến trình userspace – `agetty` – do systemd sinh ra, và **không có symlink tĩnh nào trong image để bật nó**.

Thứ tạo ra nó là `systemd-getty-generator`, chạy ở mỗi lần boot: nó đọc `/sys/class/tty/console/active`, và với mỗi console có thể làm tty, nó sinh một symlink runtime trở vào `serial-getty@.service`. Nói cách khác: *chỉ cần thêm `console=ttyS0` vào cmdline là bạn có cả kernel log lẫn login prompt*, không phải enable gì thêm.

CHI TIẾT ĐẸP

`/sys/class/tty/console/active` liệt kê console theo **thứ tự ngược** với thứ tự đăng ký – hàm sinh ra nó trong `drivers/tty/tty_io.c` in danh sách bằng vòng `while (i--)`. Nên console *đăng ký sau cùng* – cũng chính là cái đang giữ `/dev/console` – hiện ra *đầu tiên*. Đó là cơ chế cụ thể đằng sau câu "cái `console=` cuối cùng mới là cái quan trọng". Trong guest, `cat /sys/class/tty/console/active` nên ra `ttyS0 tty1`.

Phía libvirt: `virsh console` nối vào đâu

Trong domain XML, `<serial type='pty'>` khiến QEMU cấp một cặp pty trên host; QEMU giữ đầu master, còn đường dẫn slave (`/dev/pts/N`) được báo ngược về libvirt và xuất hiện trong `virsh dumpxml` của máy *đang chạy* – nó không có trong config persistent, nên đừng đi tìm ở đó.

Quan hệ giữa `<serial>` và `<console>` hơi rối vì lý do lịch sử. Điều cần nhớ:

- `<console type='pty'><target type='serial'>` **không phải thiết bị riêng** – nó là chính `<serial>` đầu tiên nhìn từ góc khác. Nên chỉ có tối đa một `<console>` kiểu `serial`.
- Serial *emulate* (`ttyS0`) khởi tạo **rất sớm** – đủ sớm để thấy GRUB và firmware. Console *paravirtualized* (`virtio`, hiện ra thành `/dev/hvc0` trong guest) khởi tạo muộn hơn nhiều và **không sống trong lúc GRUB chạy**.

Đó là lý do lab này chọn `ttyS0` chứ không phải `hvc0`: phiên B1 sẽ phá initramfs, và thứ cần nhìn nằm ở giai đoạn mà `virtio-console` còn chưa tồn tại.

BÃY · ĐỐI PHÍM THOÁT CONSOLE

`-e` / `--escape` là **option toàn cục của `virsh`**, phải đứng *trước* subcommand. `virsh -e ^X console lab01` đúng; `virsh console lab01 -e ^X` sai. Còn `--devname`, `--force`, `--safe` là option của subcommand `console` và **không** liên quan gì đến phím thoát: `--devname` chọn thiết bị nào, `--force` / `--safe` quản chuyện độc quyền phiên.

Chỉ một client được attach tại một thời điểm, vì hai lý do độc lập. libvirt *chủ động* khoá – mã trong `qemu_driver.c` trả về lỗi `Active console session exists for this domain`; và bên dưới, pty vốn dĩ chỉ giao mỗi byte cho đúng một reader, nên hai người đọc song song sẽ xé nhỏ output một cách ngẫu nhiên. Cái khoá của libvirt biến một race không xác định thành một thông báo lỗi sạch sẽ. Gặp lỗi đó sau khi rút mạng: `virsh console --force`.

BLOCK C • ~20 PHÚT

Serial console trong thực tế

Việc cần làm

1. Attach vào console bằng `virsh console`. Nếu màn hình trống, **nhấn Enter một lần** – thường chỉ là prompt chưa được vẽ lại.
2. Trong guest: `cat /proc/cmdline`, tìm `console=ttyS0`. Đối chiếu với `/sys/class/tty/console/active` – thứ tự hai file này *ngược nhau*, và bạn đã biết vì sao.
3. **Reboot VM trong khi vẫn đang attach.** Phải thấy kernel log chạy từ dòng `Linux version` ... trở đi. Đây là mục tiêu chính của block, không phải bước phụ.
4. Thoát bằng `Ctrl+]`, rồi kiểm chứng VM vẫn `running`.
5. Đối chiếu: `dmesg | head -40` trong guest so với những gì vừa thấy trên console. Chỗ khác nhau ở đâu, và mục 03 đã giải thích vì sao.

BẦY • HAI PHÍM THOÁT ĐỀ NHAU

Nếu host lab là máy khác thì bạn đang ở *ba* tầng: shell trên máy đang ngồi → ssh vào host lab → `virsh console` vào guest. Hai tầng trong đó có phím thoát riêng:

- `Ctrl+]` – thoát `virsh console`. Không cần đứng đầu dòng, bấm lúc nào cũng ăn.
- `~.` – ngắt phiên ssh. Chỉ được nhận diện **ngay sau một newline** – đây là lý do nó "lúc được lúc không": bạn đang gõ dở dòng. Bấm Enter trước đã.

Gõ nhầm `~.` khi đang trong console sẽ giết cả phiên ssh và **để lại một console session treo đang giữ khoá** – lần sau attach sẽ báo `Active console session exists` và phải `--force`. Thoát đúng thứ tự: `Ctrl+]` trước, rồi mới rời ssh.

Nếu ssh lồng nhiều tầng, escape phải nhân lên: `~.` cho một tầng lồng, `~~.` cho hai. Cơ chế: ssh ngoài cùng nuốt dấu `~` đầu tiên và truyền vào trong một dấu `~` nguyên văn. Thêm một lý do nữa để chỉ ssh một tầng và ở trong tmux.

Nghiem thu Block C

- ☐ Reboot trong `virsh console` → thấy kernel log từ dòng đầu.
- ☐ Thoát bằng `Ctrl+]` → `virsh list` vẫn thấy VM `running`.
- ☐ `/proc/cmdline` trong guest chứa `console=ttyS0`, và giải thích được bằng một câu dòng đó nghĩa gì.

ATTACH VÀ KHÁM PHÁ

HOST LAB → GUEST

```
# trước khi attach: xem libvirt đang phôi pty nào
$ virsh dumpxml lab01 | grep -A3 '<serial'
$ virsh ttyconsole lab01          # => /dev/pts/N trên host

$ virsh console lab01
# màn hình trống? Enter một lần.

# ===== từ đây trở xuống là TRONG GUEST =====
$ cat /proc/cmdline
$ cat /sys/class/tty/console/active      # ttyS0 tty1 – thứ tự ngược!
$ cat /etc/default/grub.d/50-cloudimg-settings.cfg
$ systemctl status serial-getty@ttyS0.service
$ cat /proc/sys/kernel/printk           # 4 4 1 7
$ cloud-init status --long
$ cat /sys/class/dmi/id/product_serial  # ds=nocloud

# reboot NGAY TRONG console – mục tiêu chính của block
$ sudo reboot
# => phải thấy: [    0.000000] Linux version 6.8.0-... chạy từ đầu

# thoát: Ctrl+] (KHÔNG phải ~.)

# ===== quay lại HOST =====
$ virsh list                            # lab01 vẫn running

# kệt vì phiên console cũ còn treo:
$ virsh console lab01 --force
```

Muốn thấy cả những dòng bị `quiet` lọc mất – thí nghiệm đáng làm một lần:

BỎ BỘ LỌC CONSOLE

TRONG GUEST

```
$ sudo dmesg -n 7                      # console_loglevel = 7
$ cat /proc/sys/kernel/printk          # số đầu đã đổi
# hoặc động, không cần đổi loglevel:
$ echo 1 | sudo tee /sys/module/printk/parameters/ignore_loglevel
```

MỤC 04

Bấy đã biết, xếp theo nơi phát sinh

AppArmor chặn QEMU – và nó không phải chuyện quyền POSIX

Triệu chứng: `Permission denied` với file qcow2 hoặc backing file, dù `ls -l` nhìn đúng hoàn toàn. Trên Ubuntu, nghi AppArmor *trước* khi nghi chmod. Nhưng có **hai** cơ chế khác nhau và phân biệt được chúng là kỹ năng thật:

	APPARMOR	DAC (QUYỀN POSIX)
Nguyên nhân	virt-aa-helper không được phép đọc đường dẫn đó, nên không sinh nổi profile cho VM	process QEMU chạy dưới libvirt-qemu:kvm, không đi xuyên được thư mục 0750
Dấu hiệu phân biệt	có dòng apparmor="DENIED" trong journalctl -k	không có dòng DENIED nào – chỉ có lỗi của libvirt
Cách sửa	chuyển file vào /var/lib/libvirt/images; hoặc thêm rule ở /etc/apparmor.d/local/...	chmod o+x cho từng thư mục trên đường dẫn

Cơ chế cụ thể, vì nó khá đẹp: mỗi VM ở `qemu:///system` được cấp một profile riêng, sinh ra bởi `virt-aa-helper` – một chương trình phụ mà libvirtd gọi thay vì tự động vào AppArmor. Helper đọc domain XML, đi qua từng disk / serial / hostdev, kiểm tra và chuẩn hoá đường dẫn, rồi ghi ra `/etc/apparmor.d/libvirt/libvirt-<uuid>.files`.

Nhưng **bản thân helper cũng bị AppArmor giới hạn**. Profile của nó cho phép đọc `@{HOME}/**` và `/var/lib/libvirt/images/**`, nhưng có dòng `deny @{HOME}/.* mrwkl` – nghĩa là mọi thứ nằm trong **thư mục ẩn** của home (`~/.local/share/libvirt/images/`, `~/.cache/`) đều bị chặn thẳng. Helper không stat được file, profile không sinh được, VM không khởi động.

CÁCH NÉ ĐƠN GIẢN NHẤT CHO LAB

Để image gốc và disk overlay trong **storage pool mặc định của libvirt**,

`/var/lib/libvirt/images`. Ba lý do cùng lúc: profile của `virt-aa-helper` cho phép đọc đường dẫn đó tường minh; quyền thư mục cho `libvirt-qemu` đi qua được; và `dynamic_ownership` cho libvirt tự đổi chủ sở hữu file mà không phải đánh nhau với home directory.

ĐỌC DENIAL	HOST LAB
<pre>\$ journalctl -k -g DENIED --since "10 min ago" \$ dmesg grep -i apparmor \$ sudo aa-status grep libvirt \$ ls /etc/apparmor.d/libvirt/ # profile per-VM có sinh ra không</pre>	

Dòng denial ghi rõ `profile="..."`. Nếu là `libvirt-<uuid>` thì QEMU đang bị chặn; nếu là `/usr/lib/libvirt/virt-aa-helper` thì lỗi ở khâu *sinh* profile. Hai chỗ sửa khác nhau.

Bảng tra nhanh khi kẹt

TRIỆU CHỨNG	KIỂM CÁI NÀY TRƯỚC
virsh không thấy VM vừa tạo	virsh uri – đang ở session hay system? Đã id thấy group libvirt chưa (tức đã login lại chưa)?
VM boot rất lâu rồi mới ra login, hoặc login bị khoá	Triệu chứng kinh điển của cloud-init không tìm thấy datasource. Đường dẫn trong <code>--cloud-init</code> có tuyệt đối không? Trong guest: <code>cloud-init status --long</code> và <code>lsblk -f</code> tìm nhãn cidata.
Sửa user-data rồi reboot mà không đổi gì	Đúng như thiết kế. Seed ISO đã bị gỡ, và instance-id không đổi. Phải <code>undefine - -remove-all-storage</code> và tạo lại.
virsh console attach được nhưng trống	Enter một lần. Vẫn trống: <code>virsh dumpxml grep -A3 serial</code> có thiết bị không, và <code>/proc/cmdline</code> của guest có <code>console=ttyS0</code> không.
Active console session exists	Phiên trước chưa đóng sạch (thường do ~. nhầm). <code>virsh console --force</code> .

--osinfo ubuntu24.04 báo unknown OS	osinfo-db quá cũ. Cần bản từ noble-updates trở lên. <code>osinfo-query os grep 24.04</code> .
Permission denied với file qcow2	Có dòng <code>apparmor="DENIED"</code> không? Có → AppArmor. Không → quyền POSIX / thư mục không có bit x.
VM chạy nhưng chậm khủng khiếp	/dev/kvm mở được không? Không thì QEMU đã rơi về TCG. <code>virsh dumpxml grep -i domain</code> phải là <code>type='kvm'</code> , không phải <code>type='qemu'</code> .

Riêng nhánh Arch

- **Đừng động vào multipass.** Chỉ có qua AUR (`canonical-multipass`), comment section đầy lỗi build và lỗi OVMF khi launch. Quan trọng hơn: multipass không có tương đương `virsh snapshot`, thứ P2 cần.
- Network `default` thường **không tự bật**, và cần `dnsmasq` cài riêng.
- Arch dùng modular daemon → enable `.socket`, không phải `.service`. Bê nguyên lệnh `systemctl` từ Ubuntu sang là gãy ngay ở đây.
- Arch rolling nên `virsh --version` / `qemu-system-x86_64 --version` thường mới hơn Ubuntu LTS vài phiên bản. Ghi lại số — về sau gặp lỗi "option không tồn tại" thì đây là nghi phạm đầu tiên.

MỤC 05

Nghiệm thu & câu hỏi cuối phiên

Definition of Done

- ☐ `virsh list` cho thấy VM đang `running`; login thành công bằng user khai trong `user-data`.
- ☐ Reboot VM trong `virsh console` → thấy kernel log từ dòng đầu; thoát bằng `Ctrl+]`, VM vẫn chạy.
- ☐ `cat /proc/cmdline` trong guest chứa `console=ttyS0`, và note có một câu giải thích dòng đó nghĩa gì.
- ☐ Note có: **tên host đã làm**, chuỗi lệnh đầy đủ từ host trắng → VM chạy, nội dung `user-data`, và vị trí file qcow2.

Test tiêu chuẩn cho note: người-là-mình-tuần-sau đọc note làm lại được không cần Google. Nếu note của bạn có dòng nào kiểu "cài các gói cần thiết" thì nó chưa đạt.

Chưa xong mà hết giờ → dừng đúng chỗ đang dừng, ghi lại chỗ vướng; P2 mở màn bằng việc gỡ nó. **Không lẩn giở.**

Câu hỏi kiểm tra — đóng terminal, đóng trang này

Trả lời vào note, không nhìn lại tài liệu. Mở phần chấm điểm sau khi đã viết xong câu trả lời của mình.

- 1 Kể tên các thành phần tham gia từ lúc gõ `virt-install` đến lúc VM chạy (tối thiểu 3), và vai trò một câu mỗi cái.

■ Ý CẦN CÓ

`virt-install` (CLI, dịch tham số thành XML, chạy xong là thoát) → `libvirtd` (daemon, giữ định nghĩa XML, sinh dòng lệnh QEMU, quản pool/network/AppArmor) → `qemu-system-x86_64` (một process cho mỗi VM, emulate thiết bị, mỗi vCPU là một thread) → KVM (`/dev/kvm`, module kernel, cho phép chạy code guest trên VT-x). Điểm ăn thêm nếu nói được `libvirtd` còn gọi `qemu-img` hộ để tạo overlay và gọi `virt-aa-helper` để sinh profile.

- 2 Trong setup vừa làm, cloud-init lấy `user-data` từ đâu — cơ chế vật lý nào — và vì sao nó chỉ có tác dụng ở lần boot đầu tiên?

■ Ý CẦN CÓ

Một ảnh iso9660 có nhãn volume `cidata`, gắn vào VM như CD-ROM; cloud-init quét block device tìm nhãn đó rồi đọc file. "Chỉ lần đầu" có **hai** lý do, phải nêu cả hai: (a) semaphore trong `/var/lib/cloud/instances/<instance-id>/sem/` chặn module PER_INSTANCE chạy lại — và vì `virt-install` để `meta-data` rỗng nên instance-id luôn là hằng số `nocloud`; (b) `virt-install` gỡ hẵn CD-ROM khỏi domain XML và xóa file ISO sau boot đầu.

- 3 `console=ttyS0` là chỉ thị cho **ai**, nói điều gì — và nếu thiếu nó thì `virsh console` sẽ thấy gì?

■ Ý CẦN CÓ

Chỉ thị cho *kernel của guest*, ở dòng lệnh boot: in log ra cổng serial ttyS0. Không liên quan tới host hay libvirt. Thiếu nó: `virsh console` vẫn attach vào pty thành công nhưng **trống trơn** — không kernel log, và cũng **không login prompt**, vì `systemd-getty-generator` sinh `serial-getty@ttyS0` dựa trên chính danh sách kernel console đang active.

- ④ Trên một host Ubuntu, `systemctl restart libvirtd` có phải "cả câu chuyện" không? Trả lời có/không và giải thích. Nếu là Arch thì câu trả lời đổi thế nào?

■ Ý CẦN CÓ

Trên Ubuntu: **có** — Debian/Ubuntu không build daemon modular, chỉ có `libvirtd` (cộng `virtlogd`, `virtlockd`). Đây là chỗ phần lớn tài liệu ghi sai. Trên Arch: **không** — có `virtqemud`, `virtnetworkd`, `virtstoraged`... chạy socket-activated, nên thứ cần enable là các unit `.socket`, và restart `.service` gần như vô nghĩa vì daemon tự tắt khi rảnh rồi tự dậy khi có client.

- ⑤ Bạn thấy một dòng lỗi trong `dmesg` của guest mà console không hiển thị. Vì sao?

■ Ý CẦN CÓ

Vì `printk()` ghi vào ring buffer *không lọc*, nhưng chỉ ghi ra console nếu mức ưu tiên vượt `console_loglevel`. Cloud image boot với `quiet` $\equiv$ `loglevel=4`, nên mức 4–7 (warning trở xuống) vào buffer mà không ra cổng serial. Sửa tạm: `dmesg -n 7`. Chiều ngược lại cũng có: dòng in trước khi console đăng ký, hoặc sau khi buffer tràn.

Stretch — chỉ khi dư giờ, không tính vào DoD

- Entry trong SSH config để `ssh lab` vào thẳng guest. Nếu host lab là máy khác, dùng `ProxyJump` qua nó để ssh một phát tới guest từ máy đang ngồi.
- Ném trước P2: `virsh snapshot-create-as` một phát, revert, xem trạng thái có quay lại thật không. Rồi để dành phần chính cho phiên sau.
- Xóa VM hoàn toàn và dựng lại **chỉ bằng note của mình**, bấm giờ. Con số đo được chính là baseline cho mục tiêu "dưới 2 phút" của phiên sau. Ghi kèm điều kiện đo — loại ổ, tốc độ mạng — nếu không thì lần sau so với chính mình cũng vô nghĩa.
- Thêm `earlycon=uart8250,io,0x3f8,115200n8` vào cmdline của guest, reboot trong console và so xem thấy được thêm những dòng nào ở đầu.

MỤC 06

Nguồn & đọc thêm

Xếp theo mức độ nên đọc, không theo chủ đề. Nguyên tắc của lộ trình vẫn giữ: man page và tài liệu upstream là nguồn chính, blog là nguồn phụ.

Đọc trong hoặc ngay sau phiên

TÀI LIỆU CHÍNH CHỦ

How to run cloud-init locally https://docs.cloud-init.io/en/latest/howto/run_cloud_init_locally.html

Ví dụ chuẩn của chính upstream về virt-install --cloud-init. Lưu ý ví dụ vẫn dùng jammy; đổi sang noble và --osinfo ubuntu24.04.

TÀI LIỆU CHÍNH CHỦ

NoCloud datasource <https://docs.cloud-init.io/en/latest/reference/datasources/nocloud.html>

Nguồn gốc của nhãn cidata và ngữ nghĩa instance-id. Ngắn, đọc hết được.

TÀI LIỆU CHÍNH CHỦ

Connection URIs <https://libvirt.org/uri.html>

Luật chọn URI mặc định. Đọc một lần là hết bị qemu:///session cản.

KERNEL DOC

Linux Serial Console <https://docs.kernel.org/admin-guide/serial-console.html>

Trang gốc của luật nhiều console=, kèm ví dụ hai dòng lệnh khác thứ tự vẫn cho kết quả giống nhau.

TÀI LIỆU CHÍNH CHỦ

Ubuntu Server – libvirt <https://ubuntu.com/server/docs/how-to/virtualisation/libvirt/>

Tên gói và cách cài phía Ubuntu, kèm ghi chú về AppArmor.

Tra khi cần, không đọc tuyến tính

MAN PAGE

virsh(1) <https://libvirt.org/manpages/virsh.html>

Đọc mục console, ttyconsole, domifaddr, net-dhcp-leases, và option toàn cục -e.

MAN PAGE

virt-install(1) <https://manpages.ubuntu.com/manpages/noble/en/man1/virt-install.1.html>

Mục --cloud-init, --disk (backing_store/backing_format), và --osinfo với giải thích vì sao thiếu nó là lỗi cứng.

TÀI LIỆU CHÍNH CHỦ

Domain XML format <https://libvirt.org/formatdomain.html>

Mục Console/Serial: quan hệ giữa <serial> và <console>, vì sao emulate khởi tạo sớm hơn paravirtualized.

TÀI LIỆU CHÍNH CHỦ

libvirt Daemons <https://libvirt.org/daemons.html>

Kiến trúc modular, ba socket mỗi daemon, quy trình chuyển mono → modular.

TÀI LIỆU CHÍNH CHỦ

QEMU/KVM driver – Driver security architecture <https://libvirt.org/drvqemu.html>

Giải thích AppArmor + virt-aa-helper từ phía upstream.

KERNEL DOC

printk basics <https://docs.kernel.org/core-api/printk-basics.html>

Tám mức log, bốn số trong /proc/sys/kernel/printk, quan hệ ring buffer ↔ console.

TÀI LIỆU CHÍNH CHỦ

qemu-img <https://www.qemu.org/docs/master/tools/qemu-img.html>

Backing chain, commit, rebase -u. Nhớ rằng phần synopsis in nhầm -B; CLI thật là -F.

TÀI LIỆU CHÍNH CHỦ

QEMU – Removed features <https://www.qemu.org/docs/master/about/removed-features.html>

Mục "qemu-img backing file without format": vì sao khai backing format thành bắt buộc từ 6.1, và rủi ro bảo mật đằng sau.

MAN PAGE

ssh(1) – ESCAPE CHARACTERS <https://man.openbsd.org/ssh>

Danh sách đầy đủ 11 escape và luật "phải đứng ngay sau newline". ~? in ra bảng ngay trong phiên.

TÀI LIỆU CHÍNH CHỦ

Module reference – bảng frequency <https://docs.cloud-init.io/en/latest/reference/modules.html>

Cột frequency cho biết module nào PER_INSTANCE, nào PER_ALWAYS, nào PER_ONCE.

Để dành cho các phiên sau

TÀI LIỆU CHÍNH CHỦ

Boot stages <https://docs.cloud-init.io/en/latest/explanation/boot.html>

Năm giai đoạn và unit systemd tương ứng. Cần khi debug thời gian boot.

TÀI LIỆU CHÍNH CHỦ

Snapshot XML format <https://libvirt.org/formatsnapshot.html>

internal (một file) vs external (snapshot một file, delta file khác) – với đĩa đã có backing chain thì không tầm thường.

KERNEL DOC

Kernel parameters <https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html>

Tra earlycon, earlyprintk, keep_bootcon, ignore_loglevel – bộ đồ nghề cho phiên phá initramfs.

MAN PAGE

systemd-getty-generator(8) <https://www.freedesktop.org/software/systemd/man/latest/systemd-getty-generator.html>

Generator là khái niệm systemd đáng nắm sớm: chạy trước khi tính dependency graph, sinh unit động.

Dẫn trong bài

TÀI LIỆU CHÍNH CHỦ

Ubuntu cloud images – release build <https://cloud-images.ubuntu.com/releases/noble/release/>

Bản cố định, tên file ubuntu-24.04-server-cloudimg-amd64.img. Dùng bản này cho lab để tái tạo được kết quả cũ.

TÀI LIỆU CHÍNH CHỦ

Ubuntu cloud images – daily build <https://cloud-images.ubuntu.com/noble/current/>

Đổi mỗi ngày, tên file noble-server-cloudimg-amd64.img. Dùng dùng cho lab cần tái lập.

BUG · MAILING LIST

Ubuntu bug #1016695 – console order trong cloud image <https://bugs.launchpad.net/ubuntu-on-ec2/+bug/1016695>

Hồ sơ của quyết định đặt tty1 trước ttyS0: console đồ hoạ vẫn thấy log, còn ttyS0 giành được /dev/console.

SOURCE / PACKAGING

packages.ubuntu.com – filelist của libvirt-daemon <https://packages.ubuntu.com/noble/amd64/libvirt-daemon/filelist>

Bằng chứng gốc rằng Ubuntu không ship virtqemud: xem thẳng danh sách file của gói.

TÀI LIỆU CHÍNH CHỦ

Fedora Changes – LibvirtModularDaemons <https://fedoraproject.org/wiki/Changes/LibvirtModularDaemons>

Phía distro thật sự bật modular daemon, để đối chiếu.

TÀI LIỆU CHÍNH CHỦ

passt – user-mode networking <https://passt.top>

Thay thế SLIRP cho VM rootless; libvirt hỗ trợ từ 9.0.0. Không dùng ở lab này nhưng đáng nhớ tên.

Ba mẫu đáng nhớ

115200 KHÔNG PHẢI CON SỐ AI ĐÓ CHỌN

UART 8250/16550 lấy mẫu mỗi bit ở tốc độ **16×** baud rate, và thạch anh trên PC là **1.8432 MHz**. Vậy tốc độ tối đa là $1843200 / 16 = 115200$ chẵn chẵn, ở divisor bằng 1. Mọi tốc độ "chuẩn" phía dưới chỉ là con số đó chia cho số nguyên: 57600 ($\div 2$), 38400 ($\div 3$), 19200 ($\div 6$), 9600 ($\div 12$). Comment ngay trong `arch/x86/include/asm/serial.h` còn than rằng con 16550A thừa sức chạy 1.5 Mbit/s – *thạch anh* mới là trần, không phải con chip. Còn `ttys0` = cổng I/O `0x3F8`, IRQ 4, tức COM1 của BIOS.

`CTRL+J` LÀ DI SẢN CỦA TELNET

Man page của virsh nói thẳng: "By default, **telnet's** `^J` is used." Không phải lựa chọn tùy hứng của libvirt – đó là cái ký tự mà mọi sysadmin đã có sẵn phản xạ cơ bắp từ thời telnet. Về mặt mã, `CTRL+J` truyền byte **0x1D**, ký tự điều khiển ASCII *GS (Group Separator)* – gần như không ứng dụng terminal nào dùng tới, đúng thứ cần cho một phím thoát. QEMU thì chọn `CTRL+a c`, thừa kế từ `screen`/`tip` – hai dòng dõi khác nhau, và đó là lý do hai hệ sinh thái gõ khác cảm giác.

VÌ SAO ĐÚNG 8 MỨC LOG, KHÔNG PHẢI 10

Tám mức `KERN_EMERG` ... `KERN_DEBUG` không phải phát minh của kernel – chúng là **severity code của syslog BSD**, mượn nguyên để log kernel chảy chung đường ống với log userspace. ABI của `/dev/kmsg` nói rõ: tiền tố `<N>` có *3 bit thấp là syslog priority*. Ba bit → đúng tám giá trị. Trong cùng tài liệu đó còn một chi tiết chống giả mạo nhỏ mà chần chu: userspace ghi được vào buffer của kernel, nhưng **không thể** mạo danh facility `LOG_KERN (0)` – để nguồn gốc một dòng log luôn xác định được.

Mọi khẳng định kỹ thuật ở trên đã đối chiếu với tài liệu upstream – libvirt, cloud-init, QEMU, kernel.org, livecd-rootfs, OpenSSH – tính đến 22/08/2026. Chỗ nào lệch với tài liệu phổ biến thì được đánh dấu riêng kèm bằng chứng.