

Phiên 2 — Snapshot và lab-reset.sh

Làm cho việc phá hỏng hệ thống trở nên rẻ. Snapshot để quay lại trong vài giây, script để dựng lại từ số không — và biết khi nào cần cái nào.

THỜI LƯỢNG	THUỘC	MỐC	ĐẦU VÀO
2 giờ, không lẩn giờ	Phase 0 — Dựng lab	M-L0	Phiên 1 đã xong
ARTIFACT			
lab-reset.sh			

Cả lộ trình từ Phase 1 trở đi đứng trên một giả định: phá hệ thống thoải mái. Giả định đó chỉ đúng nếu chi phí quay lại gần bằng không. Phiên này dựng đúng hai thứ làm nên chi phí đó — và quan trọng hơn, phân biệt được lúc nào dùng cái nào, vì lẫn lộn chúng là lỗi thiết kế mà rất nhiều người mắc suốt nhiều năm.

CÁCH DÙNG TRANG NÀY

Cơ chế đọc trước khi gõ. Việc cần làm thì tự ghép lệnh. Khối *Hiện lệnh đầy đủ* mở ra sau khi đã thử — trừ Block B, nơi **không có lời giải** vì chính cái script là sản phẩm của phiên.

MỤC 00

Trước khi bấm giờ

Phiên này giả định Phiên 1 đã xong. Ba dòng dưới đây kiểm đúng phần Phiên 2 sẽ dựng tới — chạy hết trước khi tính giờ.

- ☐ `virsh domstate <vm>` trả về một trạng thái, không phải lỗi. Lỗi nghĩa là domain chưa được define — quay lại Phiên 1.
- ☐ `virsh domblklist <vm>` → đĩa là file `.qcow2`. Nếu ở Phiên 1 lỡ tạo disk `raw` thì **internal snapshot không chạy được**, và cách sửa nằm ở việc tạo lại đĩa chứ không ở cờ của `virsh`.
- ☐ `qemu-img info --backing-chain` trên file đĩa → thấy **đúng hai** khối `image:`. Một khối nghĩa là VM đang trỏ thẳng vào cloud image gốc, và mọi thứ ở Block A sẽ làm bản nó.
- ☐ Còn ít nhất **gấp đôi RAM của VM** chỗ trống trên ổ chứa storage pool. Lý do nằm ở mục 01 — internal snapshot của VM đang chạy nuốt đúng chừng đó.
- ☐ Ghi lại `virsh --version` và `qemu-img --version` vào note. Gần như mọi bẫy của phiên này đều gắn với version.

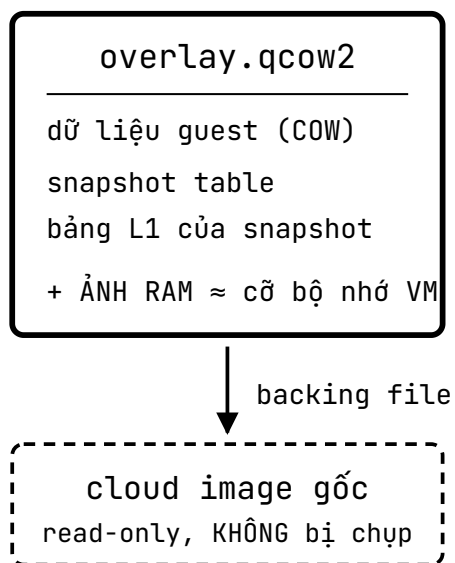
MỤC 01

Hai kiểu snapshot, và cái gì nằm ở đâu

Chữ "snapshot" trong libvirt gộp hai cơ chế khác nhau đến mức gần như không liên quan. Nhầm hai cái này là nguồn của phần lớn tai nạn — kể cả tai nạn mất dữ liệu.

Internal và external khác nhau ở đâu

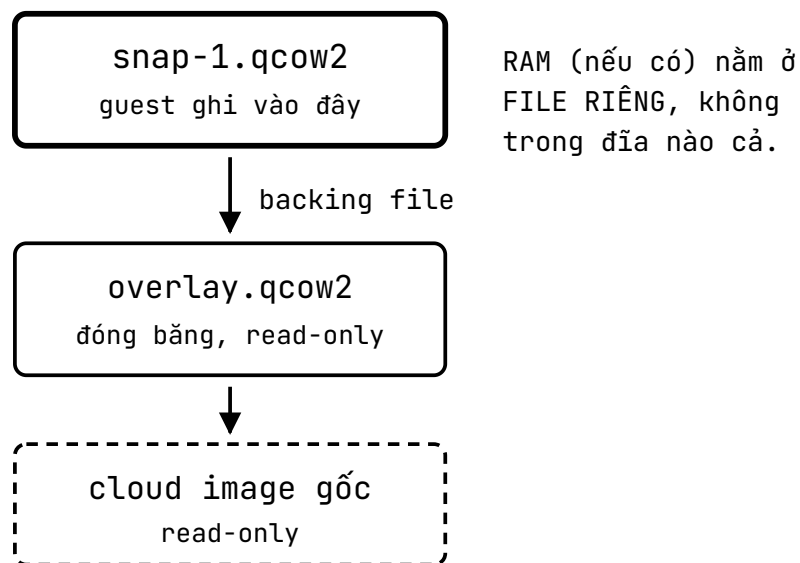
INTERNAL – MỘT FILE PHÌNH RA



Chuỗi vẫn sâu 2. File trên cùng to lên.

Chụp lúc VM chạy $\Rightarrow$ guest bị dừng trong lúc dump RAM.

EXTERNAL – CHUỖI DÀI RA



Mỗi snapshot +1 tầng. Xóa = merge thật, tốn thời gian.
Đây là kiểu duy nhất Red Hat còn hỗ trợ.

Cùng gọi là "snapshot" nhưng hai cấu trúc trên đĩa hoàn toàn khác nhau – và vì thế cái giá khi xóa cũng khác nhau hoàn toàn.

	INTERNAL	EXTERNAL
Nằm ở đâu	bên trong chính file qcow2, ở <i>snapshot table</i>	file mới; file cũ thành backing, read-only
Đòi format	bắt buộc qcow2	đích phải là qcow2, nguồn thì không bắt buộc
Tạo	gần như tức thời (đổi bảng L1)	tức thời (chỉ tạo file mới)
Xoá	thao tác metadata, nhanh	copy dữ liệu thật qua blockcommit, lâu theo lượng đã ghi
Số file	không đổi	+1 mỗi snapshot
RAM lưu ở đâu	trong chính file đĩa	file memory riêng
Nhược điểm lớn	guest bị dừng trong lúc dump RAM; file phình	chuỗi dài ra, dọn dẹp phức tạp

BA ĐIỀU NÊN BIẾT TRƯỚC KHI GỖ

RAM thật sự nằm trong file qcow2. Spec qcow2 mô tả một trường *"Size of the VM state in bytes"* trong mỗi entry của snapshot table, và nói rõ phần VM state được lưu *"giống như dữ liệu đĩa của guest, chỉ khác là ở các offset lớn hơn kích thước đĩa ảo"*. VM 4 GiB RAM $\Rightarrow$ mỗi internal snapshot "sống" cộng thêm chừng đó vào file.

Snapshot chỉ chụp tăng trên cùng. Bảng L1, snapshot table, refcount table đều là cấu trúc *nội bộ của một file qcow2* – không có mục nào trỏ tới backing file. Cloud image gốc không nằm trong snapshot. Hệ quả cần nhớ: revert ngấm giả định backing file *không đổi*. Ai đó sửa hay xoá cloud image gốc thì snapshot vẫn "tồn tại" mà dữ liệu guest sai – không có thông báo lỗi nào cả.

Chụp lúc VM chạy thì guest bị dừng. QAPI của QEMU ghi thẳng: *"execution of the guest CPUs may be stopped during the time it takes to save the snapshot"*. Đó là nhược điểm kỹ thuật, không phải bug.

Không cờ nào thì ra kiểu gì

Câu hỏi này man page trả lời lấp lửng, nhưng source thì dứt khoát. Hàm quyết định trả về "dùng external" khi và chỉ khi một trong ba điều kiện đúng, còn lại mặc định internal:

TÌNH HUỐNG	KẾT QUẢ
VM đang chạy, không cờ nào	internal full system – đĩa + toàn bộ RAM, tất cả trong qcow2
VM đã tắt, không cờ nào	internal disk-only (không có RAM để lưu)
--disk-only	external, không kèm RAM
--memspec ..., snapshot=external	external, có kèm RAM ở file riêng
--diskspec ..., snapshot=external khi VM đang chạy	vẫn internal – vòng kiểm tra disk chỉ chạy khi VM đã tắt

Dòng cuối bảng trên là chỗ dễ mất cả buổi. Muốn external trên VM *đang chạy*, phải dùng `--disk-only` hoặc `--memspec`; khai `--diskspec ..., snapshot=external` một mình sẽ im lặng cho ra internal. Cách kiểm chắc chắn sau khi tạo: `virsh snapshot-dumpxml <vm> <tên> | grep snapshot=`, hoặc nhìn xem có file mới sinh ra bên cạnh không.

Chuyện "internal đã bị khai tử"

CHỖ RẤT NHIỀU TÀI LIỆU NÓI SAI

Câu "internal snapshot đã bị deprecated" xuất hiện khắp nơi và **chỉ đúng một nửa**. Cần tách ba tầng:

- **QEMU upstream:** *không* deprecate. `savevm`/`loadvm`/internal snapshot không nằm trong danh sách deprecated hay removed. Ngược lại, QEMU còn thêm API mới cho nó (`snapshot-save` từ QEMU 6.0).
- **libvirt upstream:** *không* có cảnh báo deprecation nào, và còn đang sửa lỗi cho internal ở 10.9.0, 10.10.0, 11.0.0.
- **Red Hat:** đây mới là nơi deprecate. Deprecated từ RHEL 8; RHEL 9.4 thay bằng external; RHEL 10 nói thẳng internal là deprecated và "*Do not use internal snapshots in production environments*".

Nói cách khác: đó là **chính sách hỗ trợ của vendor**, không phải cái chết kỹ thuật ở upstream. Phân biệt được hai chuyện này là một kỹ năng dùng lại được ở rất nhiều nơi khác.

Vì vậy phiên này làm **cả hai**, và làm external trước. Không phải để đủ bài, mà vì hai lý do thực tế: đường bạn sẽ gặp ở môi trường doanh nghiệp là external, và cái giá khi xoá của hai kiểu khác nhau đến mức chỉ làm một kiểu thì không thấy được.

Một tin tốt: libvirt trên Ubuntu 24.04 **đã đủ** để làm external hoàn chỉnh. Hỗ trợ xoá external có từ libvirt 9.0.0 (01/2023), revert external từ 9.9.0 (11/2023) — cả hai đều sớm hơn bản 10.0.0 mà noble ship. Kiểm tại chỗ: `virsh capabilities | grep -i externalSnapshot`.

Snapshot và revert

Tạo và đọc

Việc cần làm:

1. **Đặt dấu vết trước.** Trong guest, tạo một thứ để kiểm chứng bằng đúng một lệnh — một file với nội dung biết trước là đủ. Ghi lại lệnh kiểm vào note; bạn sẽ dùng nó ba lần.
2. **Tạo một internal snapshot lúc VM đang chạy**, đặt tên và mô tả rõ ràng. Rồi tắt VM, tạo thêm một cái nữa. So thời gian tạo của hai lần.
3. **Đối chiếu hai góc nhìn vào cùng một thứ:** `virsh snapshot-list --tree` và `qemu-img snapshot -l` trên file đĩa. Hai lệnh này có nói cùng một chuyện không? Cái nào biết nhiều hơn, và điều đó cho biết internal snapshot nằm ở đâu?

4. Nhìn cột `VM SIZE` của `qemu-img snapshot -l` ở hai snapshot vừa tạo. Đây là bằng chứng trực quan nhất cho chuyện RAM nằm trong file đĩa – một cái khác 0, một cái bằng 0.
5. Tạo thêm một external snapshot và quan sát khác biệt: có file mới sinh ra không, `virsh domblklist` giờ trở vào đâu, `qemu-img info --backing-chain` sâu mấy tầng.

CÂU HỎI TỰ TRẢ LỜI TRONG BLOCK

Sau `snapshot-revert`, những thứ nằm ngoài VM – file trên host, entry trong `known_hosts`, DHCP lease của libvirt – có bị revert theo không? Câu trả lời quyết định cách viết script ở Block B.

Phá rồi revert

Việc cần làm: chọn một thứ hỏng rõ ràng nhưng chưa đến mức không boot – đổi nội dung config của một service đang chạy, hoặc xóa dấu vết ở bước 1. Đừng phá tới mức không boot ở phiên này; cái đó để dành cho phiên boot process. Rồi `snapshot-revert` và kiểm dấu vết bằng đúng lệnh đã ghi.

Trạng thái VM sau khi revert *không* phải lúc nào cũng là "đang chạy" – nó quay về đúng trạng thái lúc chụp:

TRẠNG THÁI LƯU TRONG SNAPSHOT	KHÔNG CỬ	--RUNNING	--PAUSED
running	running	running	paused
paused	paused	running	paused
shut off / disk-only	shut off – không tự khởi động	boot lên running	boot lên paused

CHỖ MAN PAGE KHÔNG CÒN KHỚP CODE

Man page của `snapshot-revert` liệt kê ba trường hợp cần `--force`. Đọc source libvirt hiện tại thì `--force` chỉ bypass được **đúng một**: VM có managed save image và snapshot không kèm memory state.

Trường hợp "snapshot thiếu thông tin rollback" `return -1` vô điều kiện, trước cả khối kiểm tra `force` – nên `--force` không cứu được. Trường hợp VNC/Spice thì đã biến mất khỏi code từ libvirt 7.10.0, khi VM process luôn được restart.

Vì vậy: nếu virsh đòi `--force`, hãy đọc *thông báo lỗi* trước khi thêm cờ. Nó đang nói về một tình huống cụ thể, và trong libvirt hiện đại chỉ có một tình huống mà cờ đó thật sự thay đổi kết quả. Thêm cờ bừa để "cho nó chạy" là đang bịt mắt mình.

Xoá, và dung lượng đi đâu

Việc cần làm: trước khi xóa snapshot, chạy cả hai lệnh `ls -l` và `du -h` trên file qcow2 và ghi lại hai con số. Xóa snapshot. Chạy lại. Hai con số đó hành xử khác nhau, và giải thích được vì sao là phần thưởng thật của block này.

■ Ý CẦN CÓ · VÌ SAO HAI CON SỐ KHÁC NHAU

Xoá internal snapshot **không** làm file ngắn lại. Cơ chế qcow2 là refcount: refcount về 0 nghĩa là cluster được đánh dấu *trống* và sẽ được tái sử dụng cho lần cấp phát sau. qcow2 không có cơ chế truncate cuối file, nên `ls -l` (apparent size) đứng yên.

Nhưng qcow2 bật mặc định option `pass-discard-snapshot` — *"Generate discard requests when snapshot related space is freed"*. Trên ext4/xfs, QEMU punch hole những cluster vừa giải phóng, nên số block thật sự cấp phát giảm ngay: `du -h` và dòng `disk size` của `qemu-img info` đều tụt xuống.

Muốn file thật sự nhỏ lại thì phải viết lại nó: `qemu-img convert -O qcow2 cũ.qcow2 mới.qcow2`.

Nghiệm thu Block A

- ☐ Tạo snapshot → phá VM → revert → kiểm dấu vết bằng một lệnh, kết quả đúng như trước khi phá.
- ☐ Nói được internal snapshot nằm ở đâu về mặt vật lý, **có bằng chứng từ** `qemu-img` chứ không phải từ trí nhớ.
- ☐ Có trong note: một internal và một external snapshot, kèm khác biệt quan sát được giữa hai cái (số file, độ sâu chuỗi, thời gian xóa).

■ HIỆN LỆNH ĐẦY ĐỦ · BLOCK A

INTERNAL SNAPSHOT

HOST LAB

```
# dấu vết, chạy trong guest
$ echo "truoc-snapshot-$(date +%s)" | sudo tee /etc/lab-marker
$ cat /etc/lab-marker          # đây là lệnh kiểm, ghi vào note

# ===== trên host =====
# VM đang chạy, không chờ nào ⇒ internal FULL SYSTEM (đĩa + RAM)
$ time virsh snapshot-create-as lab01 snap-running \
    --description "co RAM, VM đang chạy"

$ virsh shutdown lab01 && sleep 15
$ time virsh snapshot-create-as lab01 snap-off \
    --description "khong RAM, VM đã tắt"

$ virsh snapshot-list lab01 --tree
$ virsh snapshot-info lab01 snap-running

# góc nhìn từ tầng dưới — cột VM SIZE là ảnh RAM
$ sudo qemu-img snapshot -l /var/lib/libvirt/images/lab01.qcow2
# snap-running: VM SIZE khác 0 | snap-off: 0 B
```

EXTERNAL SNAPSHOT

HOST LAB

```
$ virsh start lab01 && sleep 20

# --disk-only ép external. Thiếu nó thì ra internal, dù có --diskspec.
$ virsh snapshot-create-as lab01 snap-ext \
    --description "external, không RAM" \
    --disk-only --atomic

$ virsh domblklist lab01          # đĩa giờ trở vào file MỚI
$ ls -l /var/lib/libvirt/images/ # có thêm lab01.snap-ext
$ sudo qemu-img info --backing-chain /var/lib/libvirt/images/lab01.snap-ext
```

```
# đếm số dòng 'image:' ⇒ độ sâu chuỗi, giờ là 3
$ sudo qemu-img info --backing-chain --output=json \
    /var/lib/libvirt/images/lab01.snap-ext | jq 'length'
```

PHÁ, REVERT, XOÁ

HOST LAB → GUEST

```
# phá: xoá dấu vết trong guest
$ sudo rm /etc/lab-marker && cat /etc/lab-marker    # No such file

# ===== host =====
$ virsh snapshot-revert lab01 snap-running
$ virsh domstate lab01    # quay về đúng trạng thái lúc chụp
# trong guest: cat /etc/lab-marker ⇒ nội dung cũ trở lại

# đo dung lượng TRƯỚC khi xoá
$ ls -l /var/lib/libvirt/images/lab01.qcow2    # apparent size
$ du -h /var/lib/libvirt/images/lab01.qcow2    # block thật cấp phát
$ sudo qemu-img info /var/lib/libvirt/images/lab01.qcow2 | grep -E 'virtual|disk si

$ virsh snapshot-delete lab01 snap-running

# đo lại – ls đứng yên, du tụt
$ ls -l /var/lib/libvirt/images/lab01.qcow2
$ du -h /var/lib/libvirt/images/lab01.qcow2

# xoá external: libvirt chạy blockcommit ngầm, CHẬM hơn hẳn
$ time virsh snapshot-delete lab01 snap-ext
$ virsh domblklist lab01    # đã pivot về file cũ
```

MỤC 02

Revert và rebuild giải hai bài toán khác nhau

Đây là chỗ lẫn lộn phổ biến nhất, và cũng là lý do phiên này có Block B chứ không dừng ở Block A.

	REVERT	REBUILD
Về đâu	một trạng thái đã biết	trạng thái sạch tuyệt đối
Chi phí	giây	phút
Thừa kế	mọi thứ có trong ảnh chụp – kể cả rác mình không biết mình đã tạo	không thừa kế gì
Chúng mình được điều gì	không gì cả	note của mình đúng và đủ
Dùng khi	hằng ngày, sau mỗi thí nghiệm	khi không còn tin trạng thái hiện tại

Dòng thứ tư là lý do thật sự của Block B. Một script dựng lại lab từ số không là **bài kiểm tra cho ghi chép của chính mình**: nếu note thiếu một bước, script sẽ không chạy. Snapshot không bao giờ phát hiện ra điều đó, vì nó mang theo cả những thứ bạn làm mà quên ghi.

lab-reset.sh

BLOCK NÀY KHÔNG CÓ LỜI GIẢI

Script chính là artifact của phiên và là điều kiện của mốc M-L0. Dưới đây có *tính chất phải đạt*, *cơ chế* của từng chỗ khó, và một số *mảnh lệnh vận hành rời* – nhưng không có script hoàn chỉnh, ở đây hay ở bất kỳ spoiler nào. Tự ghép là phần đang được luyện.

Idempotent theo trạng thái đầu vào

Script phải chạy được từ **ba** điểm xuất phát: VM đang chạy, VM đã tắt nhưng còn define, VM không tồn tại. Cách làm đúng là *hỏi trước rồi mới hành động*, không phải nhét `|| true` vào mọi dòng – `|| true` nuốt luôn cả lỗi thật, và ba tháng nữa bạn sẽ không hiểu vì sao script "chạy xong" mà chẳng có gì xảy ra.

Công cụ để hỏi là `virsh domstate`. Ba điều cần biết trước khi dùng nó trong điều kiện rẽ nhánh:

- Chuỗi trả về có **khoảng trắng**: `shut off`, `in shutdown`, `no state`. Không quote khi so sánh là gây.
- Domain **chưa define** không cho ra chuỗi nào – `virsh` báo lỗi và thoát khác 0. Đây là trạng thái thứ tư trên thực tế, phải bắt bằng exit code chứ không bằng chuỗi.
- `--reason` làm đổi format output. Nếu script có parse thì đừng bật.

Danh sách chuỗi đầy đủ: `running`, `idle`, `paused`, `in shutdown`, `shut off`, `crashed`, `pmsuspended`, `no state`.

set -euo pipefail, và chỗ nó không cứu

Dòng đầu script nên có, nhưng hiểu sai nó còn nguy hiểm hơn không có.

`pipefail` **giải quyết gì**. Mặc định, exit status của một pipeline là status của lệnh *cuối cùng*. Nghĩa là `qemu-img info khong-ton-tai.qcow2 | grep 'virtual size'` không làm script chết dù `qemu-img` đã fail – status của nó bị nuốt. Script đi tiếp trên dữ liệu rỗng, và lỗi lộ ra ở chỗ khác, muộn hơn nhiều.

BẦY · SET -E CÓ MỘT DANH SÁCH NGOẠI LỆ

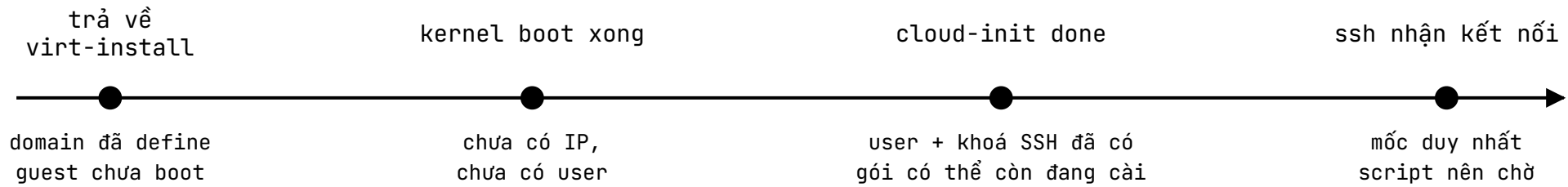
Bash manual liệt kê rõ: shell **không** thoát khi lệnh fail nằm trong danh sách ngay sau `while`/`until`, trong phần test của `if`, trong một chuỗi `&&` hoặc `||` (trừ lệnh cuối cùng), trong pipeline mà không phải lệnh cuối, hoặc khi status bị đảo bằng `!`.

Nói ngắn: `set -e` **không kích hoạt khi status đang được dùng làm điều kiện**. Đây vừa là thứ làm mẫu `if ! virsh domstate ...; then` an toàn, vừa là cái bẫy – một hàm được gọi trong `if` sẽ mang "miễn nhiệm `set -e`" cho *toàn bộ* thân hàm, nên mọi lệnh fail bên trong đó đều không làm script dừng.

Với `set -u`, hai dạng khai triển tham số đáng thuộc: `{VAR:-default}` cho giá trị mặc định, và `{VAR:?thông báo}` để bắt buộc phải có – dạng thứ hai rất hợp cho tham số đầu vào của script, vì nó tự dừng kèm thông báo thay vì chạy tiếp với chuỗi rỗng.

Thế nào là "VM đã sẵn sàng"

Đây là phần khó nhất của script, và cũng là phần dạy được nhiều nhất. Bốn mốc dưới đây **không** trùng nhau, và khoảng cách giữa chúng là hàng chục giây:



Khoảng cách giữa mốc 1 và mốc 4 là hàng chục giây, và thay đổi theo tải máy – nên mọi con số sleep cố định đều sai: quá ngắn thì script gãy, quá dài thì phí thời gian mỗi lần chạy.

Bốn mốc khác nhau. Script chờ nhằm mốc là script chạy được trên máy mình và gãy trên máy khác.

Hai công cụ để chờ, mỗi cái có cạm bẫy riêng:

Phía guest — `cloud-init status --wait`. Nó block tới khi trạng thái rời khỏi `not started`/`running`, rồi trả exit code: **0** = xong sạch, **1** = có lỗi cứng, **2** = xong nhưng có recoverable errors. Hai điều dễ vấp:

- Nó trả **0** cả khi **cloud-init bị disable** — trạng thái `disabled` không nằm trong tập lặp, nên lệnh về ngay. Script kiểu "chờ cloud-init xong rồi ssh" sẽ đi tiếp lập tức trên một máy đã tắt cloud-init.
- Dưới `set -e`, exit code 1 hoặc 2 sẽ **giết script**. Guest boot degraded là chuyện bình thường, nên cần tự phân loại exit code thay vì để `set -e` xử lý.

Phía host — `virsh domifaddr`. Ba nguồn, và cả ba đều có thể rỗng ngay sau khi boot vì những lý do khác nhau: `lease` rỗng khi guest chưa DHCP xong; `agent` rỗng khi guest chưa cài `qemu-guest-agent`; `arp` rỗng khi chưa có traffic L2 nào tới host. Đây là lý do lệnh này phải nằm trong vòng lặp có timeout, không phải gọi một phút.

CLOUD IMAGE KHÔNG CÓ SẴN QEMU-GUEST-AGENT

Kiểm manifest của cloud image noble: không có `qemu-guest-agent`. Muốn dùng `--source agent` thì phải đưa nó vào mục `packages:` của cloud-init — và khi đó lại rơi vào nghịch lý "phải chờ cloud-init xong mới có agent để hỏi xem cloud-init xong chưa". Với script reset, đường `lease` đơn giản hơn và đủ dùng.

Tính chất script phải đạt

- Idempotent** theo ba trạng thái đầu vào đã nêu ở trên.
- Dừng khi hỏng**: `set -euo pipefail` ngay dòng đầu, và hiểu rõ chỗ nó không cứu.
- Tách "image nào" khỏi "dụng thế nào"**: `release` là một biến ở đầu file, không rải rác trong URL và tên file. Mặc định noble.
- Cài sẵn bộ công cụ**: `strace ltrace perf bpftrace linux-tools-* build-essential`. Quyết định một chỗ — cài qua `packages:` của cloud-init, hay ssh vào chạy `apt` sau? Hai cách khác nhau về thời gian và về khả năng nhìn thấy lỗi. Chọn và **ghi lý do vào note**. Đọc mục Bẫy về `perf` trước khi tin là mình đã cài xong.
- Kết thúc khi VM thật sự sẵn sàng**, với điều kiện dừng rõ ràng và có timeout. Không `sleep 60`.
- In ra thứ dùng được ngay** ở dòng cuối: tên VM và IP, hoặc thẳng câu lệnh ssh.

Nghiem thu Block B

- ☐ Script chạy sạch từ **cả ba** trạng thái đầu vào, cả ba lần đều kết thúc bằng một VM ssh vào được.
- ☐ Trong VM vừa dựng: `strace -V`, `bpftrace --version`, và `perf --version` đều chạy được.
- ☐ Script không có dòng `sleep` nào với con số cố định làm điều kiện sẵn sàng.

■ MÀN LỆNH VẬN HÀNH RỜI · KHÔNG PHẢI SCRIPT

Đây là các lệnh đơn lẻ để tra cú pháp, không phải khung script. Việc ghép chúng lại — thứ tự, rẽ nhánh, vòng chờ, xử lý lỗi — chính là bài tập.

DỌN SẠCH MỘT VM

HOST LAB

```
# domstate: bắt cả trường hợp domain chưa tồn tại (exit khác 0)
$ virsh domstate lab01 2>/dev/null || echo undefined

# undefine: các cờ cần khi VM đã có snapshot / UEFI / managed save
$ virsh undefine lab01 --remove-all-storage --snapshots-metadata \
    --checkpoints-metadata --managed-save --nvram

# chờ IP từ lease — đây chỉ là LỆNH, vòng lặp và timeout tự viết
$ virsh domifaddr lab01 --source lease
$ virsh net-dhcp-leases default

# trong guest: chờ cloud-init, và đọc exit code thay vì để set -e xử lý
$ cloud-init status --wait; echo "exit=$?"
$ cloud-init status --format json | jq -r '.status, .extended_status'
```

BLOCK C · ~15 PHÚT

Đo và truy chỗ nghẽn

Việc cần làm:

- `time ./lab-reset.sh`, chạy **ba lần**, ghi cả ba số. Khác nhau nhiều thì tự hỏi vì sao — cache của apt? DHCP lease cũ? page cache của host?
- Chưa dưới 2 phút thì **định vị trước, tối ưu sau**. Chèn mốc thời gian vào các bước để biết thời gian đi đâu, đừng đoán. Thường có đúng một bước chiếm phần lớn.
- Ghi vào note: con số, bước tốn nhiều nhất, và **một ý tưởng cụ thể để rút ngắn** — chưa cần làm.

GHI KÈM ĐIỀU KIỆN ĐO, KHÔNG THÌ CON SỐ VÔ NGHĨA

`time` bao gồm cả thời gian tải gói qua mạng, nên nó phụ thuộc đường truyền và mirror. Ghi kèm: loại ổ chứa storage pool, có phải lần chạy đầu sau khi xóa cache không, và mạng lúc đó thế nào. Thiếu những thứ này thì lần sau so với chính mình cũng vô nghĩa.

Bây đã biết

`perf` cài rồi mà vẫn không chạy

Triệu chứng: `apt install linux-tools-common` thành công, `man perf` hoạt động, nhưng `perf --version` in ra một cảnh báo và không làm gì.

Cơ chế: `linux-tools-common` là package kiến trúc `all` – nó *không thể* chứa binary phụ thuộc CPU. Filelist cho thấy nó chỉ có wrapper script ở `/usr/bin/perf` cộng toàn bộ manpage. Binary thật nằm ở `/usr/lib/linux-tools/<kernel>/perf`, thuộc một package gắn với đúng ABI kernel. Wrapper đọc `uname -r` rồi đi tìm binary tương ứng; không thấy thì in:

THÔNG BÁO CỦA WRAPPER	TRONG GUEST
WARNING: perf not found for kernel 6.8.0-137 You may need to install the following packages for this specific kernel: <pre>linux-tools-6.8.0-137-generic linux-cloud-tools-6.8.0-137-generic</pre> You may also want to install one of the following packages to keep up to date: <pre>linux-tools-generic linux-cloud-tools-generic</pre>	

Cách sửa, và vì sao chọn nó: cloud image chạy meta `linux-image-virtual`, nên metapackage khớp là `linux-tools-virtual`. Cài `linux-tools-$(uname -r)` cũng chạy được *ngay bây giờ*, nhưng nó chỉ khớp kernel đang chạy – một lần `apt upgrade` lên ABI mới rồi reboot là `perf` gây lại, vì gói versioned cũ ở lại còn cái mới không được kéo về. Metapackage mới là thứ tự đi theo kernel.

ĐỪNG TIN BẢN BUILD NÀO CŨNG GIỐNG NHAU

Build cloud image noble hiện tại đã cài sẵn `linux-tools-common` và gói versioned tương ứng, nên `perf` chạy được ngay sau boot. Nhưng bản GA 24.04 thì **không có bất kỳ `linux-tools-*` nào**, và không bản nào có metapackage. Nội dung image trôi theo từng build, nên đừng khẳng định – kiểm tại chỗ bằng `dpkg -l | grep linux-tools` và `ls /usr/lib/linux-tools/`. Đây cũng chính là lý do DoD bắt chạy thử `perf --version` chứ không chỉ tin `apt install` đã xong.

Nghĩ rằng `undefine --remove-all-storage` sẽ ăn mất cloud image gốc

CẢNH BÁO PHỔ BIẾN NÀY SAI

Man page nói thẳng: lệnh này *“only removes the top level image of a backing chain, any backing stores of the image are kept as they may be shared”*. Source cũng xác nhận – vòng lặp chỉ lấy source cấp trên cùng của mỗi `<disk>`, không đi vào `<backingStore>`. Thêm một lớp bảo vệ nữa: chỉ xóa được volume mà libvirt quản lý trong storage pool; ngoài pool thì nó in *“is not managed by libvirt. Remove it manually.”* và bỏ qua.

Rủi ro thật cần cảnh báo nằm ở chỗ khác: nó xóa overlay của bạn *cùng toàn bộ internal snapshot nằm bên trong*, không hoàn tác được. Và thứ thật sự phá được cloud image gốc là `virsh blockcommit` commit xuống tận đáy – trong lab external snapshot, luôn giới hạn bằng `-base` hoặc `--shallow`, đừng bao giờ để nó chạm tới cloud image.

Bảng tra nhanh

TRIỆU CHỨNG	KIỂM CÁI NÀY TRƯỚC
snapshot-create-as báo unsupported	Đĩa có phải qcow2 không: <code>qemu-img info</code> . Internal đòi qcow2. Nếu VM boot UEFI thì xem dòng dưới.
Báo <i>"internal snapshots of a VM with pflash based firmware are not supported"</i>	VM boot UEFI. Trên libvirt 10.0.0 đây là lỗi cứng; chỉ được nối từ libvirt 10.9.0 và cần NVRAM ở định dạng qcow2. Đường vòng: dùng external.
Xin external mà vẫn ra internal	Có <code>--disk-only</code> hoặc <code>--memspec</code> chưa? <code>--diskspec ... , snapshot=external</code> một mình không đủ khi VM đang chạy.
snapshot-revert xong VM tắt ngóm	Snapshot đó chụp lúc VM đang chạy hay đã tắt? Revert đưa về <i>đúng</i> trạng thái lúc chụp. Xem <code>snapshot-info</code> .
<code>virsh undefine</code> fail dù VM đã tắt	VM có snapshot (<code>--snapshots-metadata</code>), có managed save (<code>--managed-save</code>), hoặc boot UEFI (<code>--nvram/--keep-nvram</code>). Thiếu cờ là fail.
Script chạy tay từng dòng thì được, chạy cả file thì hỏng	Ba nghi phạm theo thứ tự: <code>set -e</code> giết ở một lệnh mình coi là "được phép fail"; biến môi trường chỉ có ở shell tương tác (<code>LIBVIRT_DEFAULT_URI</code> là số một); và script chạy nhanh hơn người nên đừng chỗ chưa kịp sẵn sàng.
VM dựng xong nhưng ssh báo <code>REMOTE HOST IDENTIFICATION HAS CHANGED</code>	Host key mới, <code>known_hosts</code> cũ. Đây là việc script nên xử lý, không phải việc sửa tay mỗi lần.
Chuỗi backing dài bất thường	<code>qemu-img info --backing-chain</code> , đếm số dòng <code>image:.</code> . Reset đúng cách luôn tạo overlay từ image gốc , không phải từ đĩa của VM trước.

MỤC 04

Nghiem thu & câu hỏi cuối phiên

Definition of Done

- ☐ Tạo snapshot → phá VM → revert → kiểm dấu vết bằng một lệnh, kết quả đúng như trước khi phá.
- ☐ Note giải thích được internal snapshot nằm ở đâu về mặt vật lý, có bằng chứng từ `qemu-img`.
- ☐ Đã tạo cả một internal và một external snapshot, và ghi được ít nhất hai khác biệt quan sát được.
- ☐ `lab-reset.sh` chạy sạch từ cả ba trạng thái đầu vào.
- ☐ Trong VM vừa dựng: `strace -V`, `bpftool --version`, `perf --version` đều chạy được.
- ☐ Ba con số `time` trong note, kèm một câu chỉ ra bước tốn nhiều thời gian nhất.
- ☐ **M-LO**: xóa VM hoàn toàn rồi dựng lại **chỉ bằng một lệnh**, không mở note ra tra.

Chưa xong mà hết giờ thì dừng đúng chỗ đang dừng và ghi lại chỗ vướng. Phiên sau cần VM chạy được, không cần script hoàn hảo – script còn dở thì hoàn thiện dần, đừng lẩn giở.

Câu hỏi kiểm tra — đóng terminal, đóng trang này

- ① Internal và external snapshot khác nhau ở chỗ nào về mặt file trên đĩa? Nêu một hệ quả thực tế của khác biệt đó mà bạn *quan sát được* trong phiên này.

■ Ý CẦN CÓ

Internal nằm trong chính file qcow2 (snapshot table + bảng L1 + ảnh RAM nếu chụp lúc chạy) — số file không đổi, file phình ra. External tạo file mới, file cũ thành backing read-only — chuỗi dài thêm một tầng. Hệ quả quan sát được, chấp nhận bất kỳ cái nào: thời gian xóa (internal gần như tức thì, external phải blockcommit nên lâu); số dòng `image:` trong `qemu-img info --backing-chain`; hay `domblklist` trở vào file khác sau khi tạo external.

- ② Có đúng là internal snapshot đã bị deprecated không? Trả lời và nói rõ *ai* deprecate.

■ Ý CẦN CÓ

Phải tách được ba tầng. QEMU upstream: không deprecate, còn thêm API mới cho nó. libvirt upstream: không có cảnh báo deprecation, còn đang sửa lỗi cho internal ở các bản gần đây. Red Hat: có — deprecated từ RHEL 8, RHEL 9.4 thay bằng external, RHEL 10 nói thẳng không dùng trong production. Đây là chính sách hỗ trợ của vendor, không phải cái chết kỹ thuật ở upstream. Điểm ăn thêm nếu nói được vì sao phân biệt hai chuyện này quan trọng.

- ③ `snapshot-revert` và chạy lại `lab-reset.sh` đều cho ra "VM sạch". Kể hai thứ khác nhau giữa hai kết quả, và một tình huống cụ thể mà revert không đủ.

■ Ý CẦN CÓ

Khác biệt: thời gian (giây vs phút); revert thừa kế mọi thứ có trong ảnh chụp kể cả rác không biết mình đã tạo, rebuild thì không; revert không đụng tới thứ nằm ngoài VM (file trên host, `known_hosts`, DHCP lease), rebuild thì đổi hết; và quan trọng nhất — chỉ rebuild mới chứng minh được note của mình đúng và đủ. Tình huống revert không đủ: khi không còn tin trạng thái đã chụp (đã cài linh tinh trước khi chụp), hoặc khi cần chứng minh quy trình dựng lại từ số không chạy được.

- ④ Script chờ VM "sẵn sàng" bằng cách nào, và vì sao cách đó đúng hơn `sleep`? Nếu VM boot chậm gấp đôi bình thường thì script làm gì?

■ Ý CẦN CÓ

Phải nêu được rằng "sẵn sàng" có nhiều mốc khác nhau — `virt-install` trả về ≠ `kernel boot xong` ≠ `cloud-init done` ≠ `ssh nhận kết nối` — và script phải chờ đúng mốc cuối. Cách chờ: vòng lặp poll một điều kiện quan sát được (`cloud-init status --wait` phía guest, hoặc `virsh domifaddr` rồi thử `ssh` phía host) kèm `timeout`. `sleep` cố định sai vì thời gian boot đổi theo tải máy: quá ngắn thì gãy, quá dài thì phí mỗi lần chạy. Boot chậm gấp đôi: vòng lặp vẫn chờ tới khi điều kiện đúng, chỉ dừng khi chạm `timeout`, và khi đó phải báo lỗi rõ ràng chứ không im lặng đi tiếp.

- ⑤ Bạn xóa một internal snapshot. `ls -l` cho thấy file không nhỏ đi. Vậy dung lượng có được giải phóng không?

■ Ý CẦN CÓ

Có, nhưng không ở chỗ `ls` nhìn. `qcow2` dùng `refcount`: cluster về `refcount 0` là trống và được tái sử dụng, nhưng file không bao giờ tự truncate nên `apparent size` đúng yên. Đồng thời option `pass-discard-snapshot` mặc định bật khiến QEMU punch hole xuống filesystem, nên số block cấp phát thật (`du -h`, hoặc dòng `disk size` của `qemu-img info`) giảm ngay. Muốn file thật sự nhỏ lại thì phải `qemu-img convert` viết lại.

Stretch — chỉ khi dư giờ, không tính vào DoD

- **Golden image:** thay vì cài công cụ ở mỗi lần reset, cài một lần vào một image riêng rồi để script tạo overlay từ image đó. Đây thường là cách duy nhất xuống sâu dưới 2 phút — đo lại và so với ba con số ban đầu. `virt-sysprep` là công cụ đáng biết ở đây, nhưng tự tay làm trước để hiểu nó dọn cái gì và vì sao cần dọn.
- **Release thành tham số:** `lab-reset.sh --release noble|resolute`. Chỗ dễ sai: đổi release thì `--osinfo` phải đổi theo, và hai image không nên dùng chung một tên VM.
- **Snapshot tree:** tạo A → sửa → B → revert về A → sửa khác → C. `snapshot-list --tree` lúc này hình gì? Đây là mô hình sẽ dùng khi muốn giữ nhiều trạng thái lab song song ở các phase sau.
- **Đo cái giá của external:** tạo một external snapshot, ghi vài trăm MB trong guest, rồi `time virsh snapshot-delete`. So với thời gian xóa internal. Con số này là lý do người ta vẫn dùng internal cho lab dù vendor không khuyến khích.

Nguồn & đọc thêm

Đọc trong hoặc ngay sau phiên

TÀI LIỆU CHÍNH CHỦ

RHEL 10 – Saving and restoring VM state by using snapshots https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/10/html/configuring_and_managing_linux_virtual_machines/saving-and-restoring-virtual-machine-state-by-using-snapshots

Mục "Support limitations": Red Hat chỉ hỗ trợ external, và nói thẳng internal deprecated trong RHEL 10.

TÀI LIỆU CHÍNH CHỦ

GNU Bash Manual – The Set Builtin https://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html

Đọc phần `-e` để thấy danh sách ngoại lệ, và `pipefail`. Đây là nguồn gốc của mọi hiểu nhầm về `set -e`.

TÀI LIỆU CHÍNH CHỦ

cloud-init – CLI reference (status) <https://docs.cloud-init.io/en/latest/reference/cli.html>

Ngữ nghĩa `--wait` và ba exit code 0/1/2. Đọc trước khi đưa lệnh này vào script chạy dưới `set -e`.

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`.

TÀI LIỆU CHÍNH CHỦ

libvirt NEWS – lịch sử tính năng theo version <https://libvirt.org/news.html>

Tra "external snapshot": delete có từ 9.0.0 (01/2023), revert từ 9.9.0 (11/2023). Cũng là nơi thấy internal snapshot vẫn đang được sửa ở 10.9.0–11.0.0.

SOURCE / PACKAGING

libvirt – src/qemu/qemu_snapshot.c https://raw.githubusercontent.com/libvirt/libvirt/master/src/qemu/qemu_snapshot.c

Nguồn duy nhất trả lời chính xác "không chờ nào thì ra internal hay external" và "`--force bypass` được đúng điều kiện nào".

TÀI LIỆU CHÍNH CHỦ

QEMU – qcow2 image format specification <https://www.qemu.org/docs/master/interop/qcow2.html>

Mục Snapshots: snapshot table, và trường "Size of the VM state" chứng minh RAM nằm trong file `qcow2`.

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

GNU Bash Manual – Shell Parameter Expansion https://www.gnu.org/software/bash/manual/html_node/Shell-Parameter-Expansion.html

Các dạng `${var:-default}` và `${var:?message}` – công cụ sống chung với `set -u`.

SOURCE / PACKAGING

Manifest của cloud image noble <https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-amd64.manifest>

Danh sách gói có sẵn trong image. Tra "linux-" và "qemu-guest-agent" để biết cái gì đã cài sẵn, cái gì chưa.

Để dành cho các phiên sau

TÀI LIỆU CHÍNH CHỦ

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

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

TÀI LIỆU CHÍNH CHỦ

libvirt wiki – QEMU guest agent https://wiki.libvirt.org/Qemu_guest_agent.html

Cần gì để agent chạy: package trong guest + channel virtio tên `org.qemu.guest_agent.0`.

Dẫn trong bài

TÀI LIỆU CHÍNH CHỦ

RHEL 9 – Considerations in adopting, ch. 26 Virtualization https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/considerations_in_adopting_rhel_9/assembly_virtualization_considerations-in-adopting-rhel-9

Câu chốt: RHEL 9.4 thay cơ chế internal đã deprecated bằng external.

TÀI LIỆU CHÍNH CHỦ

QEMU – Deprecated features <https://www.qemu.org/docs/master/about/deprecated.html>

Dùng để kiểm chứng điều NGƯỢC lại: `savevm/loadvm/internal` snapshot không có trong danh sách này.

SOURCE / PACKAGING

packages.ubuntu.com – filelist của linux-tools-common <https://packages.ubuntu.com/noble/all/linux-tools-common/filelist>

Bằng chứng gói này là arch "all" và chỉ chứa wrapper + manpage, không có binary perf thật.

SOURCE / PACKAGING

packages.ubuntu.com – linux-tools-virtual (noble) <https://packages.ubuntu.com/noble/linux-tools-virtual>

Metapackage bám ABI kernel của cloud image. Xem cột Depends để thấy nó trỏ tới gói versioned nào.

Hai mẫu đáng nhớ

VÌ SAO XOÁ SNAPSHOT LẠI LÀ MỘT BLOCK JOB

Với internal, xoá là thao tác metadata: bỏ một entry khỏi snapshot table, giảm refcount, xong. Với external, libvirt **không có cơ chế riêng** để xoá — nó chạy chính `blockcommit` ngầm bên dưới, tức là copy dữ liệu thật từ tầng trên xuống tầng dưới rồi mới pivot. Đó là lý do xoá external cần một job chạy nền, có thể theo dõi bằng `virsh blockjob`, và mất thời gian tỉ lệ với lượng dữ liệu đã ghi kể từ lúc chụp. Cùng một lệnh `snapshot-delete`, hai thể giới chi phí hoàn toàn khác nhau.

CÁI ĐĨA 20G ĐẾN TỪ ĐẦU

Overlay tạo với kích thước 20G, nhưng bản phân vùng bên trong được copy từ cloud image gốc chỉ vài GB. Guest kernel thấy đĩa 20G mà partition vẫn bé. Thứ và khoảng cách đó là hai module cloud-init chạy rất sớm ở stage `init`: `growpart` nới partition, rồi `resizefs` nới filesystem theo. Cả hai đều là `PER_ALWAYS` — chạy *mỗi lần boot*, không phải chỉ lần đầu. Hệ quả tiện lợi: sau này `qemu-img resize` overlay lên lớn hơn rồi reboot là filesystem tự nới tiếp, không cần `cloud-init clean` gì cả.

Mọi khẳng định kỹ thuật ở trên đã đối chiếu tài liệu và source upstream — libvirt, QEMU, cloud-init, GNU bash, packages.ubuntu.com — tính đến 22/08/2026. Ba chỗ lệch với tài liệu phổ biến (deprecation của internal snapshot, phạm vi thật của `--force`, và `undefine --remove-all-storage` có ăn backing file không) được đánh dấu riêng kèm bằng chứng.