**Tìm hiểu và triển khai ảo hóa với KVM và QEMU trên Linux**

# **1️⃣ Ý tưởng sơ bộ và mục tiêu**

Mục tiêu của đề tài là tìm hiểu cơ chế hoạt động của ảo hóa mức hệ điều hành (kernel-level virtualization) thông qua KVM (Kernel-based Virtual Machine) và QEMU (Quick Emulator).

* KVM là một module tích hợp trong nhân Linux cho phép biến Linux thành một hypervisor loại 1 (bare-metal).
* QEMU đóng vai trò giả lập phần cứng và cung cấp giao diện quản lý máy ảo (VMs) thông qua KVM.

//QEMU là một trình giám sát máy ảo được lưu trữ: nó mô phỏng bộ xử lý của máy thông qua dịch mã nhị phân động và cung cấp một bộ các mô hình phần cứng và thiết bị khác nhau cho máy, cho phép nó chạy nhiều hệ điều hành khách khác nhau. Nó cũng có thể được sử dụng với KVM để chạy máy ảo ở tốc độ gần như gốc (bằng cách tận dụng các phần mở rộng phần cứng như Intel VT-x). QEMU cũng có thể thực hiện mô phỏng cho các tiến trình cấp người dùng, cho phép các ứng dụng được biên dịch cho một kiến trúc chạy trên kiến trúc khác.[3]

**Kiến thức:**

➢ Hiểu khái niệm ảo hóa và các loại ảo hóa (Hypervisor loại 1 và loại 2). Nắm được vai trò của KVM (Kernel-based Virtual Machine) là một hypervisor loại 1 và QEMU (Quick Emulator) là một trình giả lập phần cứng.

➢ Nắm được kiến trúc của KVM và QEMU. Hiểu cách KVM sử dụng phần cứng để tăng tốc ảo hóa và QEMU được sử dụng để giả lập các thiết bị phần cứng còn lại.

➢ Hiểu cách quản lý tài nguyên ảo hóa như CPU, RAM, ổ đĩa và mạng để tối ưu hóa hiệu suất của máy ảo.

➢ Biết cách KVM và QEMU hỗ trợ triển khai IaaS (Infrastructure as a Service) và quản lý hạ tầng ảo hóa hiệu quả.

**Kỹ năng:**

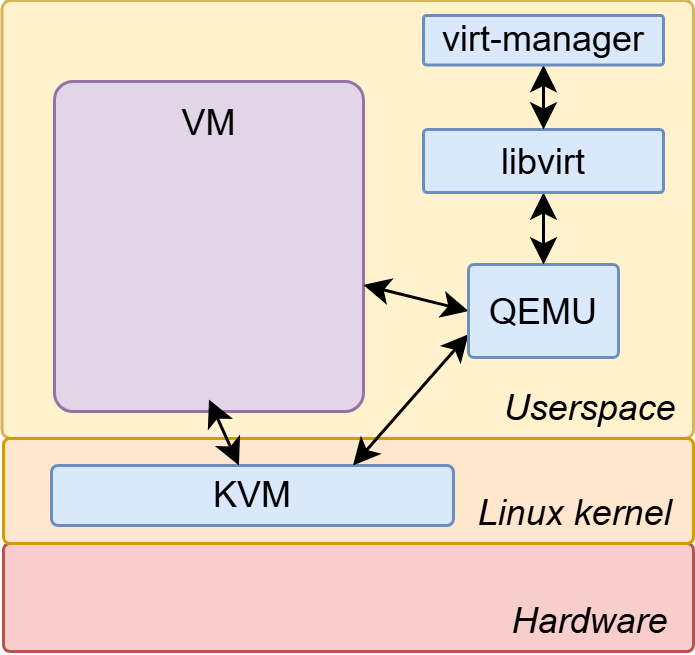
➢ Cài đặt và cấu hình KVM và QEMU trên một máy chủ Linux.

➢ Sử dụng lệnh qemu-system và các công cụ quản lý như virt-manager hoặc virsh để tạo và quản lý máy ảo.

➢ Tạo và quản lý các máy ảo (VM), bao gồm cài đặt hệ điều hành, cấu hình tài nguyên (CPU, RAM, ổ đĩa) và thiết lập mạng.

➢ Quản lý snapshot, backup và live migration của các máy ảo.

➢ Triển khai thử nghiệm một hạ tầng ảo hóa đơn giản để phục vụ mục tiêu học tập và thực hành.



# **2️⃣**Môi trường triển khai

* Nền tảng: Google Cloud Platform (GCP)
* VM Instance (máy ảo chạy trong GCP)
* Hệ điều hành: Ubuntu Server 22.04 LTS
* Cấu hình đề xuất:

# **3️⃣ Content**

## 1. Giới thiệu chung về ảo hóa và hai công nghệ KVM, QEMU

1.1. Khái niệm về ảo hóa

1.2. Lý do cần ảo hóa và lợi ích trong công nghệ thông tin

1.3. Phân loại các công nghệ ảo hóa

 • Hypervisor loại 1 (bare-metal)

 • Hypervisor loại 2 (hosted)

 • Vị trí và vai trò của KVM trong phân loại hypervisor

1.4. Tổng quan về KVM và QEMU

 • Chức năng, vai trò của KVM

 • Chức năng, vai trò của QEMU

 • Mối quan hệ và cách phối hợp giữa KVM và QEMU

## 2. Cơ chế hoạt động của KVM và QEMU

2.1. Kiến trúc KVM trong nhân Linux

 • Cách KVM sử dụng công nghệ ảo hóa phần cứng (Intel VT-x, AMD-V)

2.2. Cơ chế hoạt động của QEMU

 • Vai trò trong việc giả lập thiết bị và quản lý tài nguyên ảo

2.3. Mối liên kết giữa KVM và QEMU trong quá trình tăng tốc ảo hóa

2.4. Quy trình tạo, khởi tạo và quản lý máy ảo thông qua KVM/QEMU

## 3. Môi trường và yêu cầu triển khai ảo hóa

3.1. Yêu cầu phần cứng

 • CPU hỗ trợ ảo hóa (Intel VT-x / AMD-V)

 • RAM và dung lượng ổ đĩa

3.2. Yêu cầu phần mềm

 • Hệ điều hành Linux (Ubuntu, Fedora, CentOS, v.v.)

 • Các gói phần mềm: qemu-kvm, libvirt, virt-manager, bridge-utils...

3.3. Thiết lập người dùng và phân quyền (user groups, libvirt)

3.4. Kiểm tra khả năng hỗ trợ ảo hóa của hệ thống (lệnh lscpu, kvm-ok)

## 4. Các bước triển khai thực tế

4.1. Cài đặt và cấu hình KVM, QEMU, và Libvirt

4.2. Quản lý máy ảo bằng Virt-Manager (giao diện đồ họa)

4.3. Quản lý máy ảo bằng dòng lệnh virsh

4.4. Cấu hình mạng và lưu trữ cho máy ảo

4.5. Thực hành tạo máy ảo cài hệ điều hành (ví dụ: Ubuntu, CentOS)

4.6. Các tính năng nâng cao:

 • Snapshot (chụp trạng thái hệ thống)

 • Live Migration (chuyển máy ảo không gián đoạn)

 • Clone máy ảo

## 5. Các bài toán và tình huống ứng dụng

5.1. Ứng dụng KVM/QEMU trong môi trường phòng lab và doanh nghiệp

5.2. Bài toán tối ưu tài nguyên máy chủ

5.3. So sánh hiệu năng KVM/QEMU với các công nghệ khác (VMware, VirtualBox, Xen, v.v.)

5.4. Thực nghiệm đánh giá hiệu suất (CPU, RAM, Disk I/O)

## 6. Tổng kết và đánh giá

6.1. Ưu điểm và hạn chế của ảo hóa với KVM/QEMU

6.2. Khó khăn, thách thức trong quá trình triển khai

6.3. Hướng phát triển và ứng dụng tương lai của ảo hóa KVM/QEMU trong hạ tầng điện toán đám mây

## Gợi ý công cụ, tài liệu sử dụng:

* Công cụ: virt-manager (giao diện đồ họa), virsh (dòng lệnh), qemu-img (quản lý ổ đĩa ảo).
* Tài liệu tham khảo: tài liệu KVM-QEMU trên GitHub, hướng dẫn của các trang chuyên môn như funix.edu.vn, wiki.nhanhoa.com, tài liệu PDF chính thức QEMU.
* Thực hành trên một máy Linux thật hoặc máy ảo hỗ trợ VT-x/AMD-V.

# 

# 

# Reddit:)))

<https://www.reddit.com/r/linuxquestions/comments/1ixj4o8/can_someone_explain_qemu_to_me_really_fast/?tl=vi&rdt=40905>

Tớ xài QEMU cũng kha khá, nhưng mà xa mới tới trình độ chuyên gia được. Chắc chắn là nó còn nhiều chức năng tớ chưa đụng tới, nhưng đây là cái hiểu cơ bản (chắc là chưa đầy đủ) của tớ về nó.

QEMU về cơ bản là một trình giả lập CPU, được dùng chung với một hypervisor như KVM.

Nó nằm giữa CPU vật lý của máy chủ và máy ảo. CPU có một bộ lệnh. Có những lệnh cơ bản mà CPU nào cũng có, những lệnh phổ biến mà hầu hết CPU có, những lệnh đặc biệt chỉ có ở AMD hoặc Intel, và đôi khi bạn sẽ có những lệnh đặc biệt dựa trên chính model CPU. Vì vậy, QEMU có thể biết các lệnh của CPU vật lý, biết bạn muốn máy ảo nghĩ nó có gì, và dịch. Nó cũng có thể chỉ cần chuyển CPU thực tế cho VM, tớ tin rằng đây là cấu hình mặc định của nó.

Tại sao bạn lại muốn làm điều này? Giả sử bạn có một nền tảng ảo hóa như OpenStack để chạy một đống VM trên một đống máy chủ vật lý, tất cả đều đang chạy cùng một CPU. OpenStack có Nova, nó giống như một trình bao bọc trên KVM, giúp tất cả các máy chủ hoạt động cùng nhau. Bạn có thể làm những điều hay ho như di chuyển trực tiếp một VM từ máy chủ này sang máy chủ khác... Giả sử CPU trên cả hai máy chủ cung cấp cùng một bộ lệnh mà VM mong đợi.

Mọi thứ đang chạy tốt trong một năm, khi bạn cần sửa một máy chủ, bạn chỉ cần di chuyển trực tiếp tất cả các VM ra và làm bất cứ điều gì bạn muốn với nó. Sau đó, bạn mua thêm một đống máy chủ tính toán để thêm vào cụm của mình, nhưng CPU mới hơn và có các lệnh mới hơn. Bạn muốn có thể di chuyển trực tiếp giữa các thế hệ máy chủ khác nhau, vậy bạn phải làm gì?

Đing đing đing. QEMU! Bạn có thể sử dụng nó để trình bày cho các VM với các lệnh CPU ít phổ biến nhất cho VM và bây giờ nó không gặp vấn đề gì khi di chuyển giữa chúng, với cái giá là không bao giờ sử dụng các lệnh mới hơn trên các máy chủ mới. QEMU có các công cụ trợ giúp giúp bạn tìm thấy điều này và chúng có các cấu hình cài đặt sẵn dựa trên các model CPU.

Tớ nghĩ tất cả những điều này sẽ không liên quan gì nếu bạn chỉ muốn sử dụng VM trên PC ở nhà, nhưng tớ có thể thấy nó hữu ích để phát triển phần mềm trong VM bằng cách sử dụng các cấu hình CPU khác nhau để bạn không cần phải có CPU thực tế.

<https://www.reddit.com/r/linux/comments/7yszpz/is_qemukvm_underrated/?tl=vi>



<https://www.reddit.com/r/linuxquestions/comments/g17g8n/im_not_understanding_the_role_of_qemu_in_a_kvm/?tl=vi>

Từ trên xuống dưới trong stack:

GNOME Boxes/virt-manager/v.v… - Giao diện người dùng đồ họa đẹp  
libvirt - Quản lý ảo hóa  
QEMU - Công cụ máy ảo, mô phỏng phần cứng và cung cấp phần "máy ảo"  
KVM - Mô-đun kernel để bật khả năng ảo hóa phần cứng x86

Hy vọng với cái nhìn tổng quan này, bạn có thể thấy QEMU nằm ở đâu. QEMU là công cụ thực sự cho bạn một "VM". KVM tăng tốc nó, và libvirt cung cấp lớp quản lý mà các công cụ GUI ở tầng trên tương tác.

<https://www.reddit.com/r/linuxquestions/comments/9duxao/kvm_qemu_libvirt_virtmanager_how_do_these_all/?tl=vi>

<https://www.reddit.com/r/Ubuntu/comments/11eda3t/why_do_must_people_use_virtualbox_instead_of_qemu/?tl=vi>

<https://www.reddit.com/r/linux/comments/15xfl15/qemukvm_and_endless_tutorial_hell/?tl=vi>

<https://www.reddit.com/r/linux/comments/awub50/kvm_qemu_is_awesome/?tl=vi>

<https://www.reddit.com/r/linux/comments/1dw5zba/what_i_thought_that_qemu_was_faster/?tl=vi>

QEMU không dùng KVM chậm hơn VMware. Nhưng nó lại hữu dụng hơn nhiều cho các mục đích khác nhau.

Để hiểu điều này, cần phải hiểu rằng mã máy (mã nhị phân mà chương trình của bạn được biên dịch) chỉ là một ngôn ngữ máy tính khác.

Điều đó có nghĩa là mã máy (mã nhị phân) có thể được thông dịch và biên dịch lại thành các ngôn ngữ khác.

Và về cơ bản, đó là những gì đang xảy ra khi sử dụng mô phỏng máy đầy đủ.

Các trình mô phỏng máy đầy đủ cũ hơn như Bochs cung cấp phần cứng giả "thực sự", nhưng nó thông dịch mã máy ngay lập tức. Cũng giống như các ngôn ngữ thông dịch như BASIC hay Python được thông dịch ngay lập tức.

Vì vậy, mặc dù Bochs rất chính xác nhưng nó cũng rất chậm. Nó biến mã máy x86 thành một ngôn ngữ thông dịch.

QEMU sử dụng các kỹ thuật tương tự để "tăng tốc" mã như các ngôn ngữ lập trình "VM" như Java, C#, (hiện đại) Javascript, v.v.

Lấy Java làm ví dụ. Java là một kiến trúc máy. Giống như x86\_64, ARM hoặc RISC-V là một kiến trúc máy. Sự khác biệt là Java là kiến trúc cho một máy không thực sự tồn tại. (mặc dù có một thứ gọi là bộ xử lý Java vật lý được rút gọn)

Vì vậy, khi mọi người nói về "VM" C# hoặc Java... đó là nó: đó là một máy ảo. Và chúng sử dụng nhiều kỹ thuật lập trình như bytecode trung gian và biên dịch Just In Time (JIT) để làm cho mã của chúng nhanh hơn các ngôn ngữ thông dịch truyền thống.

Giờ đây, QEMU sử dụng các kỹ thuật như JIT để tăng tốc mã máy giống như Java làm với bytecode trung gian của nó. Điều này làm cho QEMU nhanh hơn nhiều so với các trình mô phỏng máy đầy đủ truyền thống.

Và trên hết, QEMU có các chế độ hoạt động khác nhau. Bạn có thể sử dụng QEMU để tạo một VM đầy đủ để làm bất cứ điều gì. Họ gọi đó là Mô phỏng toàn hệ thống.

Nhưng bạn cũng có thể sử dụng mô phỏng "Chế độ người dùng". Điều đó có nghĩa là bạn có thể thực thi một tệp nhị phân từ một kiến trúc khác trên máy cục bộ của mình.

<https://qemu-project.gitlab.io/qemu/user/index.html>

Vì vậy, QEMU thực sự hữu ích nếu bạn muốn phát triển các chương trình ARM trên máy tính để bàn X86 của mình, ví dụ. Nó có thể thực thi các chương trình ARM trực tiếp trong hệ điều hành của bạn. Ngoài ra, nó có thể mô phỏng một số máy và phần cứng ARM khác nhau. Và nó làm tất cả điều đó khá nhanh.

Vì vậy, nếu bạn muốn thực thi các tệp nhị phân x86 trên máy tính xách tay hoặc điện thoại ARM của mình, bạn có thể làm điều đó với QEMU.

Vâng, KVM với QEMU thì khác.

KVM biến nhân Linux thành một "Hypervisor Loại 1". Có nghĩa là bạn đang sử dụng nhân Linux làm hypervisor, giống như VMware vSphere sử dụng ESX làm hypervisor.

Và điều này có nghĩa là nó về cơ bản thêm "phân vùng" vào máy của bạn. Nó làm điều này kết hợp với các phần mở rộng ảo hóa trên CPU Intel và AMD. Điều này kích hoạt các tính năng bộ nhớ và CPU đặc biệt giúp có thể thực thi các nhân hệ điều hành theo cách lồng nhau mà không cần phải sửa đổi chúng.

Vì vậy, không giống như QEMU hoặc Bochs độc lập, không có việc thông dịch hoặc biên dịch lại nào đang diễn ra cả. Bạn đang thực thi mã gốc trực tiếp trên bộ xử lý của mình.

Vì vậy, về mặt thực thi mã thuần túy, phần mềm trong VM sẽ thực thi nhanh gần như bất kỳ phần mềm nào khác trên máy tính của bạn. Vì tất cả đều được thực thi 'tại chỗ'.

Có một số trừu tượng hóa bộ nhớ hơn, nhưng nó không thực sự khác nhiều so với loại 'Bộ nhớ ảo' hiện có đã được thực hiện cho các tiến trình thông thường.

Vì vậy, bất kỳ sự mất hiệu suất nào chủ yếu là do mô phỏng phần cứng. Những thứ như cung cấp ổ đĩa giả, cổng mạng giả, màn hình giả, v.v.

Và đó là nơi QEMU đến từ. QEMU cung cấp "phần cứng giả" được sử dụng với KVM để đánh lừa hệ điều hành khách cho rằng nó đang chạy trên một máy vật lý thực. Chỉ là với KVM, không cần phải biên dịch lại mã máy.

Và KVM không phải là người duy nhất sử dụng QEMU theo cách này.

Cả Xen và Virtualbox đều sử dụng mã QEMU cho các tính năng phần cứng của chúng.

<https://www.reddit.com/r/linuxquestions/comments/g17g8n/im_not_understanding_the_role_of_qemu_in_a_kvm/?tl=vi>

<https://voz.vn/t/cai-qemu-kvm-va-virt-manager-trong-ubuntu-linux-%C4%91e-tao-may-ao.1010462/>

<https://pve.proxmox.com/wiki/Qemu/KVM_Virtual_Machines>

<https://bitgrounds.tech/posts/kvm-qemu-libvirt-virtualization/>

<https://www.csd.uoc.gr/~hy428/vm-labs/qemu-internals-slides-apr26_2023.pdf>

<https://vmsplice.net/~stefan/qemu-kvm-architecture-2015.pdf>

<https://wiki.nhanhoa.com/kb/kvm-ao-hoa-kvm-phan-1/>

<https://casinesque.medium.com/first-approach-with-qemu-kvm-virtualization-on-linux-733ba2ffb739>

<https://bitgrounds.tech/posts/kvm-qemu-libvirt-virtualization/>

Câu hỏi rất hay — và câu trả lời là:  
 👉 **Không cần quá nhiều lệnh đâu**, trừ khi bạn muốn **triển khai thủ công hoàn toàn bằng terminal** (để hiểu sâu cách QEMU/KVM hoạt động).  
 Tùy bạn chọn hướng triển khai nào trong đề tài mà số lệnh sẽ khác nhau. Dưới đây mình chia ra 3 mức để bạn dễ hình dung 👇

## **🧩 1️⃣ Mức cơ bản – Dành cho đồ án minh họa (ít lệnh nhất)**

Nếu bạn dùng **Proxmox**, **virt-manager** hoặc **GNOME Boxes**, thì:

* Gần như **không cần gõ lệnh QEMU/KVM**.
* Tất cả thao tác (tạo VM, cấp RAM, CPU, ổ đĩa, network) đều có giao diện đồ họa.
* Bạn chỉ cần dùng 2–3 lệnh để kiểm tra hoặc thao tác cơ bản.

📘 **Ví dụ các lệnh cần thiết:**

# Kiểm tra CPU có hỗ trợ ảo hóa không

egrep -c '(vmx|svm)' /proc/cpuinfo

# Kiểm tra KVM đã bật chưa

lsmod | grep kvm

# Kiểm tra dịch vụ libvirt hoặc QEMU đang chạy

systemctl status libvirtd

👉 Mức này **đủ cho đồ án triển khai**, vì bạn có thể chụp hình giao diện cấu hình VM trong Proxmox / virt-manager.

## **⚙️ 2️⃣ Mức trung bình – Dành cho ai muốn hiểu cơ chế QEMU/KVM**

Bạn dùng **QEMU và KVM trực tiếp** (không GUI).  
 Sẽ cần khoảng **10–15 lệnh**, chủ yếu để:

* Tạo đĩa ảo
* Gắn ISO cài hệ điều hành
* Cấu hình CPU, RAM, network

📘 **Ví dụ chuỗi lệnh:**

# Tạo đĩa ảo

qemu-img create -f qcow2 ubuntu.qcow2 20G

# Chạy máy ảo cài Ubuntu

qemu-system-x86\_64 -enable-kvm -m 2048 -boot d \

-cdrom ubuntu.iso -hda ubuntu.qcow2 \

-net nic -net user

# Sau khi cài xong có thể khởi động lại

qemu-system-x86\_64 -enable-kvm -m 2048 -hda ubuntu.qcow2

👉 Mức này phù hợp nếu bạn muốn **đi sâu về kernel / hypervisor**, vì nó cho thấy rõ cách QEMU gọi KVM để tăng tốc.

## **🧱 3️⃣ Mức nâng cao – Khi bạn quản lý nhiều VM**

Nếu bạn triển khai trên **cluster hoặc Proxmox CLI**, thì sẽ dùng thêm vài lệnh qm hoặc virsh.

📘 **Ví dụ:**

# Tạo VM mới (Proxmox CLI)

qm create 101 --name testvm --memory 2048 --net0 virtio,bridge=vmbr0

# Import disk

qm importdisk 101 ubuntu.qcow2 local-lvm

# Cấu hình boot và khởi động

qm set 101 --boot c --bootdisk scsi0

qm start 101

## **✅ Kết luận gọn:**

| **Mức độ** | **Công cụ** | **Số lệnh cần** | **Dùng khi** |
| --- | --- | --- | --- |
| **Cơ bản** | Proxmox / virt-manager / Boxes | 3–5 | Trình bày đồ án, demo ảo hóa |
| **Trung bình** | QEMU + KVM trực tiếp | 10–15 | Muốn hiểu cơ chế ảo hóa sâu |
| **Nâng cao** | CLI quản lý (qm, virsh) | 10+ | Quản lý nhiều VM hoặc cluster |

👉 Tóm lại:

* Nếu bạn làm **đồ án triển khai minh họa**, **dùng GUI là đủ** (Proxmox hoặc virt-manager).
* Nếu bạn muốn **thuyết trình “mức sâu”**, nên có thêm 2–3 lệnh QEMU minh họa — là quá ổn rồi.

Bạn muốn mình soạn giúp **danh sách ngắn gọn các lệnh cần thiết nhất cho đồ án (cả GUI và CLI)** để bạn in kèm vào phần phụ lục không?  
 (kiểu “phụ lục A – Các lệnh dùng trong triển khai QEMU/KVM”)