ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Lớp TN01 - Nhóm 8386

Bài tập nhóm tuần 1 - bài 3

Mạng máy tính (CO3094)

"Deep Dive into Docker Networking"

Giảng viên hướng dẫn: Hoàng Lê Hải Thanh

Sinh viên thực hiện: Lê Nguyễn Kim Khôi - 2311671
Bùi Ngọc Phúc - 2312665
Phạm Trần Minh Trí - 2313622
Lê Trọng Thiện - 2313233
Nguyễn Công Minh - 2312080

THÀNH PHỐ HỒ CHÍ MINH, 12/2025

Mục lục

1 Virtualization, Docker và Vai trò của Mạng Máy Tính **1**
1.1 Virtualization là gì? Docker khác gì máy ảo truyền thống 1
1.2 Vì sao containerization được sử dụng rộng rãi hiện nay 1
1.3 Kiến thức mạng giúp kỹ sư thiết kế và triển khai hệ thống thật như thế nào . . . 2
1.4 Các công ty và công nghệ sử dụng Docker rộng rãi 2

1 Virtualization, Docker và Vai trò của Mạng Máy Tính

1.1 Virtualization là gì? Docker khác gì máy ảo truyền thống

Virtualization (ảo hoá) là kỹ thuật cho phép chạy nhiều hệ điều hành và ứng dụng khác nhau trên cùng một máy vật lý, bằng cách chia sẻ tài nguyên thông qua một lớp phần mềm trung gian gọi là *hypervisor*. Hypervisor tạo ra các máy ảo (Virtual Machines – VMs), mỗi VM có hệ điều hành riêng, không gian tiến trình riêng và tài nguyên ảo hoá (CPU, RAM, ổ đĩa, thiết bị mạng) tách biệt với các VM khác. Mô hình này giúp tận dụng tốt hơn phần cứng vật lý, triển khai linh hoạt nhiều hệ thống mà không cần nhiều máy chủ thật.

Docker cũng là một công nghệ ảo hoá, nhưng hoạt động ở cấp độ hệ điều hành thay vì ảo hoá toàn bộ phần cứng như VM. Thay vì khởi động một hệ điều hành khách (guest OS) riêng, Docker sử dụng kernel của hệ điều hành host và cô lập ứng dụng bằng các cơ chế như *namespaces*, *cgroups* và *filesystem layers*. Mỗi *container* vì vậy nhẹ hơn rất nhiều so với một VM, khởi động gần như tức thì, và chia sẻ phần lớn thư viện hệ thống với host.

Điểm khác biệt quan trọng giữa Docker và VM có thể tóm tắt như sau. Thứ nhất, VM ảo hoá phần cứng rồi chạy một hệ điều hành đầy đủ, trong khi Docker ảo hoá môi trường chạy ứng dụng trên một hệ điều hành chung. Thứ hai, mỗi VM cần một image hệ điều hành lớn, khởi động lâu và tốn tài nguyên; còn container thường nhỏ, khởi chạy nhanh, phù hợp với mô hình triển khai linh hoạt và mở rộng theo nhu cầu. Thứ ba, mức độ cô lập của VM thường mạnh hơn vì mỗi VM có kernel riêng, trong khi container chia sẻ kernel với host nên cần chú ý hơn tới cấu hình bảo mật.

1.2 Vì sao containerization được sử dụng rộng rãi hiện nay

Containerization được sử dụng rộng rãi vì mang lại nhiều lợi ích rõ ràng so với cách triển khai ứng dụng truyền thống và cả so với VM. Một lợi ích quan trọng là *tính di động* (portability): ứng dụng và toàn bộ phụ thuộc được đóng gói trong một image, nên có thể chạy nhất quán trên máy cá nhân, máy chủ on-premise hoặc môi trường cloud khác nhau mà không lo xung đột môi trường. Điều này rất phù hợp với mô hình phát triển hiện đại, nơi developer, tester và production đều dùng cùng một image. Thứ hai, container giúp *tăng hiệu quả sử dụng tài nguyên* do chia sẻ kernel và một phần thư viện với host, cho phép chạy nhiều container trên cùng một máy vật lý hơn so với số lượng VM tương đương. Điều này giúp giảm chi phí phần cứng và chi phí cloud. Thứ ba, container hỗ trợ *triển khai nhanh* và *tự động hoá* trong các pipeline CI/CD: việc build image, push lên registry và deploy đến nhiều môi trường có thể được kịch bản hoá một cách dễ dàng, rút ngắn thời gian đưa tính năng mới ra thị trường.

Ngoài ra, containerization hỗ trợ rất tốt cho kiến trúc microservices, nơi một hệ thống lớn được chia thành nhiều dịch vụ nhỏ, độc lập, có thể scale riêng, cập nhật riêng và deploy riêng. Khả năng tạo, nhân bản, nâng cấp và huỷ container nhanh chóng giúp các đội ngũ DevOps vận hành hệ thống phức tạp với độ tin cậy cao hơn.

1.3 Kiến thức mạng giúp kỹ sư thiết kế và triển khai hệ thống thật như thế nào

Kiến thức mạng máy tính đóng vai trò cốt lõi khi kỹ sư sử dụng Docker để xây dựng và triển khai hệ thống thực tế. Mỗi container đều giao tiếp với các container khác, với host và với mạng bên ngoài thông qua các cơ chế như bridge networks, port mapping, overlay networks hay các giải pháp như VPN và reverse proxy. Nếu không hiểu các khái niệm cơ bản như địa chỉ IP, subnet, routing, NAT, DNS, firewall hay các giao thức như TCP/UDP và HTTP, việc thiết kế kiến trúc mạng cho hệ thống container sẽ trở nên rất khó khăn.

Cụ thể, khi triển khai một hệ thống microservices, kỹ sư cần quyết định container nào được expose ra Internet, container nào chỉ giao tiếp nội bộ, cách cấu hình port mapping, cách dùng reverse proxy hoặc API gateway và cách đặt rules firewall để giới hạn lưu lượng. Kiến thức về mạng giúp họ đảm bảo hệ thống vừa *kết nối tốt* (các dịch vụ nói chuyện với nhau đúng port, đúng IP, đúng giao thức) vừa *an toàn* (không mở thừa port, không để lộ dịch vụ nội bộ ra ngoài, hạn chế tấn công như scan port hay sniffing). Bên cạnh đó, hiểu về latency, throughput, packet loss và congestion cũng rất quan trọng để tối ưu hiệu năng. Trong hệ thống phân tán, nhiều container có thể chạy trên các máy khác nhau, liên lạc qua mạng LAN hoặc Internet. Kiến thức mạng cho phép kỹ sư đánh giá đường đi của gói tin, phát hiện nút cổ chai, và chọn giải pháp phù hợp (ví dụ: dùng overlay network, VPN, load balancer, hay message broker) để cân bằng giữa hiệu năng, độ tin cậy và bảo mật.

1.4 Các công ty và công nghệ sử dụng Docker rộng rãi

Trong thực tế, rất nhiều công ty công nghệ lớn đã và đang dựa vào Docker như một phần quan trọng của hạ tầng. Các nền tảng streaming như Netflix sử dụng container để chạy hàng loạt dịch vụ backend phục vụ nội dung cho hàng triệu người dùng, kết hợp với các công cụ orchestrator để tự động scale theo tải. Các công ty gọi xe như Uber triển khai hàng trăm microservices dưới dạng container để đảm bảo khả năng mở rộng linh hoạt theo nhu cầu di chuyển của người dùng.

Trong lĩnh vực thương mại điện tử, các nền tảng lớn như Shopify hay nhiều sàn thương mại điện tử khác sử dụng Docker để xử lý lượng truy cập tăng vọt vào các dịp cao điểm, đồng thời giảm rủi ro khi cập nhật tính năng mới. Trong mảng thanh toán và tài chính, các tổ chức như PayPal hay các ngân hàng sử dụng container để xây dựng pipeline CI/CD an toàn và tuân thủ quy định, cho phép phát hành bản cập nhật nhanh nhưng vẫn kiểm soát được môi trường chạy.

Bên cạnh các công ty cụ thể, nhiều công nghệ và nền tảng hiện đại cũng dựa mạnh vào Docker. Kubernetes, một hệ thống orchestration phổ biến, sử dụng container (thường là Docker hoặc container runtime tương thích) để quản lý việc deploy, scale và cập nhật ứng dụng trên cụm máy chủ. Các nền tảng CI/CD như GitLab CI, GitHub Actions, CircleCI hay các PaaS như Heroku, OpenShift, nhiều dịch vụ của các nhà cung cấp cloud lớn đều hỗ trợ hoặc dựa trên Docker images để cung cấp môi trường build và chạy ứng dụng tiêu chuẩn, dễ tái sử dụng và dễ mở rộng.