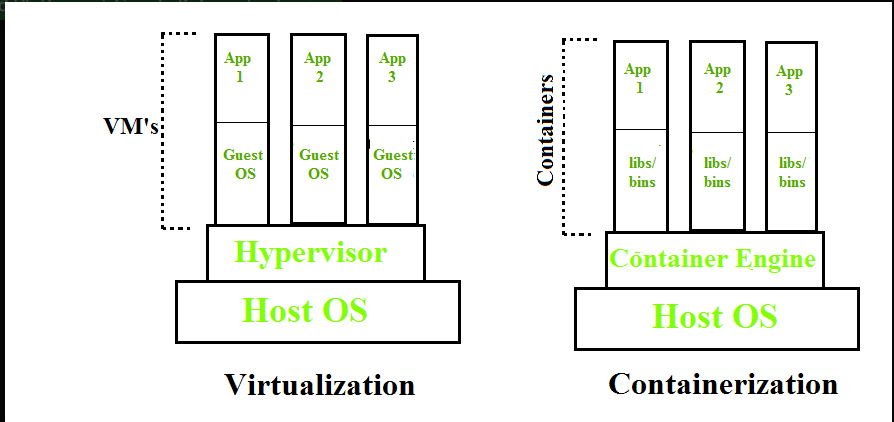
**Phần 12 : Container hóa bằng Docker**

**Container hóa**

Container hóa là ảo hóa dựa trên hệ điều hành tạo ra nhiều đơn vị ảo trong không gian người dùng, được gọi là Container. Container chia sẻ cùng một nhân máy chủ nhưng được cô lập với nhau thông qua các không gian tên riêng và cơ chế kiểm soát tài nguyên ở cấp độ hệ điều hành. Ảo hóa dựa trên container cung cấp một cấp độ trừu tượng khác về mặt ảo hóa và cô lập khi so sánh với các trình quản lý ảo. Các trình quản lý ảo sử dụng nhiều phần cứng dẫn đến chi phí chung về mặt ảo hóa phần cứng và trình điều khiển thiết bị ảo. Một hệ điều hành đầy đủ (ví dụ: Linux, Windows) chạy trên phần cứng ảo hóa này trong mỗi phiên bản máy ảo.

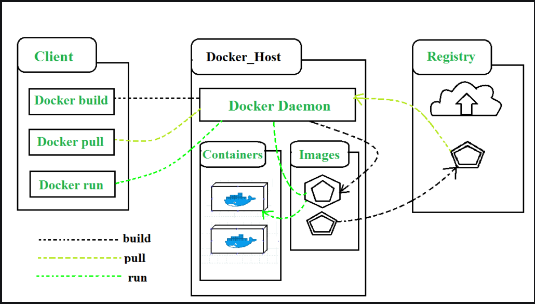
Nhưng ngược lại, container triển khai cô lập các quy trình ở cấp độ hệ điều hành, do đó tránh được chi phí chung đó. Các container này chạy trên cùng một nhân hệ điều hành được chia sẻ của máy chủ cơ sở và một hoặc nhiều quy trình có thể chạy trong mỗi container. Trong container, bạn không phải phân bổ trước bất kỳ RAM nào, nó được phân bổ động trong quá trình tạo container trong khi trong VM, trước tiên bạn cần phân bổ trước bộ nhớ rồi mới tạo máy ảo. Container hóa có khả năng sử dụng tài nguyên tốt hơn so với VM và quy trình khởi động ngắn. Đây là bước tiến tiếp theo của công nghệ ảo hóa.

Container có thể chạy hầu như ở mọi nơi, phát triển và triển khai cực kỳ dễ dàng: trên hệ điều hành Linux, Windows và Mac; trên máy ảo hoặc máy chủ trần, trên máy của nhà phát triển hoặc trong các trung tâm dữ liệu tại chỗ; và tất nhiên, trên đám mây công cộng. Container ảo hóa CPU, bộ nhớ, lưu trữ và tài nguyên mạng ở cấp độ hệ điều hành, cung cấp cho nhà phát triển chế độ xem hộp cát về hệ điều hành được cô lập hợp lý khỏi các ứng dụng khác. Docker là định dạng container nguồn mở phổ biến nhất hiện có và được hỗ trợ trên Google Cloud Platform và Google Kubernetes Engine.



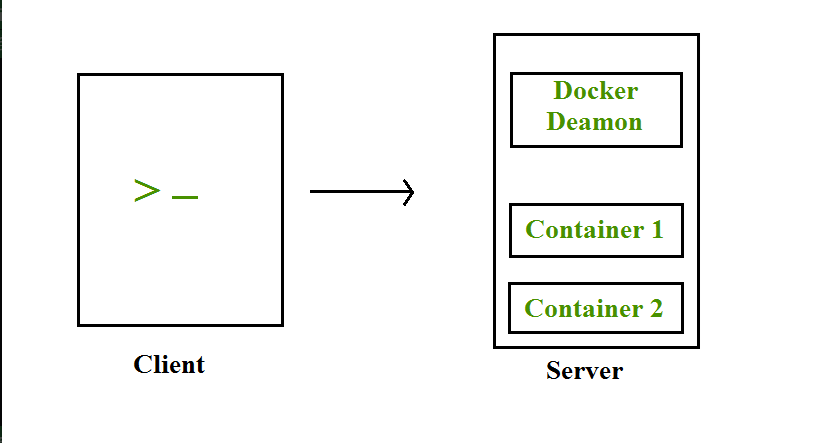
**Kiến trúc Docker**

Kiến trúc Docker bao gồm Docker client, Docker Daemon chạy trên Docker Host và kho lưu trữ Docker Hub. Docker có kiến ​​trúc client-server trong đó client giao tiếp với Docker Daemon chạy trên Docker Host bằng cách kết hợp REST API, Socket IO và TCP. Nếu chúng ta phải xây dựng Docker image, thì chúng ta sử dụng client để thực thi lệnh build cho Docker Daemon, sau đó Docker Daemon sẽ xây dựng một image dựa trên các đầu vào đã cho và lưu vào sổ đăng ký Docker. Nếu bạn không muốn tạo image, thì chỉ cần thực thi lệnh pull từ client, sau đó Docker Daemon sẽ kéo image từ Docker Hub, cuối cùng nếu chúng ta muốn chạy image, thì thực thi lệnh run từ client, lệnh này sẽ tạo container.



**Các thành phần của Docker**

Các thành phần chính của Docker bao gồm – Docker clients and servers, Docker images, Dockerfile, Docker Registries và Docker containers. Các thành phần này được giải thích chi tiết trong phần bên dưới:



**Docker Client và Server**

Docker có kiến ​​trúc client-server. Docker Daemon/Server bao gồm tất cả các container. Docker Daemon/Server nhận yêu cầu từ Docker client thông qua CLI hoặc REST API và do đó xử lý yêu cầu theo đó. Docker client và Daemon có thể có mặt trên cùng một host hoặc host khác nhau.

Docker Images – Docker images được sử dụng để xây dựng các container docker bằng cách sử dụng một mẫu chỉ đọc. Nền tảng của mỗi image là một base image ví dụ như base image như – ubuntu14.04 LTS và Fedora 20. Base image cũng có thể được tạo từ đầu và sau đó các ứng dụng cần thiết có thể được thêm vào base image bằng cách sửa đổi nó, do đó quá trình tạo một image mới này được gọi là “commiting the change”.

**Docker File**

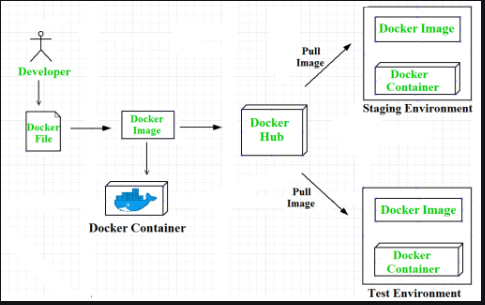
Dockerfile là một tệp văn bản chứa một loạt các hướng dẫn về cách xây dựng hình ảnh Docker của bạn. Hình ảnh này chứa tất cả mã dự án và các phụ thuộc của nó. Cùng một hình ảnh Docker có thể được sử dụng để quay 'n' số container, mỗi container có sửa đổi đối với hình ảnh cơ bản. Hình ảnh cuối cùng có thể được tải lên Docker Hub và chia sẻ giữa nhiều cộng tác viên khác nhau để thử nghiệm và triển khai. Bộ lệnh mà bạn cần sử dụng trong Docker File của mình là FROM, CMD, ENTRYPOINT, VOLUME, ENV và nhiều lệnh khác nữa.

**Docker Registries**

Docker Registry là một thành phần lưu trữ cho Docker images. Chúng ta có thể lưu trữ các images trong các kho lưu trữ công khai/riêng tư để nhiều người dùng có thể cộng tác xây dựng ứng dụng. Docker Hub là kho lưu trữ đám mây của Docker. Docker Hub được gọi là sổ đăng ký công khai, nơi mọi người có thể kéo các images có sẵn và đẩy các images của họ mà không cần tạo một image từ đầu.

**Docker Container**

Docker Container là các phiên bản chạy thời gian thực của Docker image. Container chứa toàn bộ bộ công cụ cần thiết cho một ứng dụng, do đó ứng dụng có thể chạy theo cách biệt lập. Ví dụ: Giả sử có một image của Ubuntu OS với NGINX SERVER khi image này được chạy bằng lệnh docker run, thì một container sẽ được tạo và NGINX SERVER sẽ chạy trên Ubuntu OS.



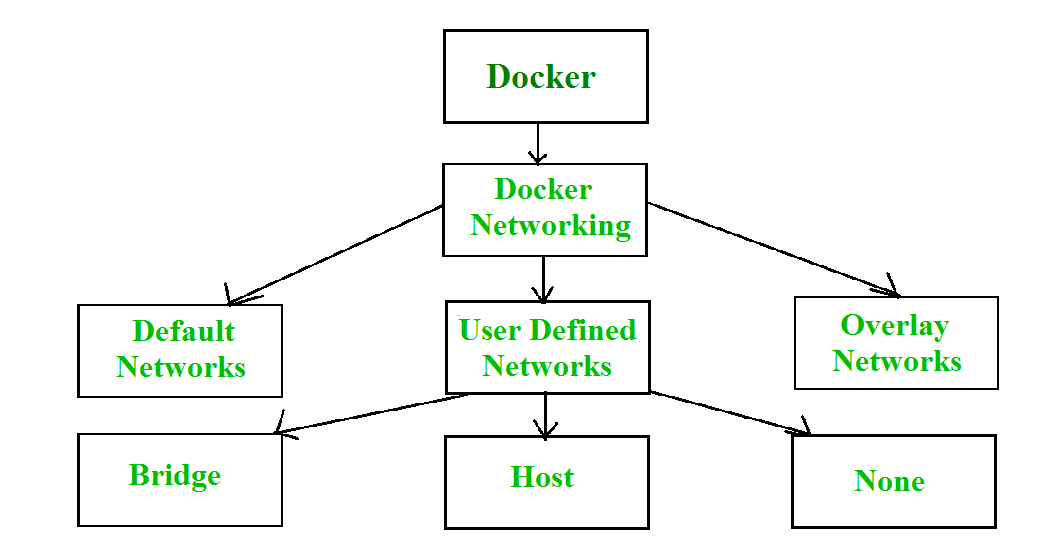
**Docker Compose**

Docker Compose là một công cụ mà chúng ta có thể sử dụng để tạo ứng dụng đa container. Công cụ này giúp cấu hình và

chạy các ứng dụng được tạo thành từ nhiều container dễ dàng hơn.

**Mạng Docker**

Khi chúng ta tạo và chạy một container, Docker tự gán một địa chỉ IP cho nó theo mặc định. Hầu hết thời gian, cần phải tạo và triển khai mạng Docker theo nhu cầu của chúng ta. Vì vậy, Docker cho phép chúng ta thiết kế mạng theo yêu cầu của mình. Có ba loại mạng Docker - mạng mặc định, mạng do người dùng xác định và mạng phủ. 



Để có danh sách tất cả các mạng mặc định mà Docker tạo ra, chúng ta chạy lệnh

**docker network ls**

**Có ba loại mạng trong Docker**

1. **Bridged network**: Khi một container Docker mới được tạo mà không có đối số –network, Docker theo mặc định sẽ kết nối container với mạng cầu nối. Trong các mạng cầu nối, tất cả các container trong một máy chủ duy nhất có thể kết nối thông qua địa chỉ IP của chúng. Mạng cầu nối được tạo khi phạm vi của các máy chủ Docker là một, tức là khi tất cả các container chạy trên một máy chủ duy nhất. Chúng ta cần một mạng phủ để tạo một mạng có phạm vi của nhiều hơn một máy chủ Docker.
2. **Host network**: Khi một container Docker mới được tạo với đối số –network=host, nó sẽ đẩy container vào ngăn xếp mạng máy chủ nơi daemon Docker đang chạy. Tất cả các giao diện của máy chủ đều có thể truy cập được từ container được gán cho mạng máy chủ.
3. **None network**: Khi một container Docker mới được tạo với đối số –network=none, nó sẽ đặt container Docker vào ngăn xếp mạng của nó. Vì vậy, trong mạng không có này, không có địa chỉ IP nào được gán cho container, do đó chúng không thể giao tiếp với nhau.