

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

-----o0o-----



**TIỂU LUẬN MÔN:
DỰ ÁN PHÁT TRIỂN PHẦN MỀM
ĐỀ TÀI:
DOCKER FOR DEVOPS AND DEVELOPER**

Giảng viên hướng dẫn: Quách Xuân Trường

Nhóm thực hiện : Nhóm 9

Lớp: KTPM - K17B

Tên thành viên: Trần Minh Long

Dương Văn Định

Nguyễn Văn Nghiệp

Nguyễn Văn Hiếu

Trần Đức Long

Mục Lục

CHƯƠNG 1: GIỚI THIỆU CHUNG	5
1.1. Tổng quan về DevOps và Developer.....	5
1.1.2. Developer	5
1.2. Docker cho devops và developer.....	6
1.2.1. Docker cho devops	6
1.2.2. Docker cho Developer.....	6
1.3. Lý do lựa chọn Docker	7
1.4. Phạm vi của chủ đề.....	8
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT.....	9
2.1. Tổng quan về Docker	9
2.1.1. Định nghĩa	9
2.1.2. Lịch sử ra đời.....	10
2.1.3. Cài đặt Docker trên Window	10
2.2. Các thành phần cơ bản của Docker	14
2.2.1. Docker Engine	14
2.2.2. Docker Image	15
2.2.3. Docker Container.....	15
2.2.4. Dockerfile và Docker Compose	16
2.2.5. Docker Hub	16

2.3. Tính Chất của docker.....	17
2.3.1. Ưu điểm	17
2.3.2. Nhược điểm	18
2.4. Quy trình triển khai docker.....	19
2.4.1. Quy trình.....	19
2.4.2. Các câu lệnh thường dùng với Docker Container	20
2.4.3. Các câu lệnh thường dùng với Docker Image	20
2.5. So sánh Docker với Kubernetes	21
2.5.1. Tổng quan về Kubernetes	21
2.5.2. Tổng quan Docker Swarm.....	21
2.5.3. So sánh Kubernetes vs Docker Swarm.....	23
2.6. so sánh Docker với máy ảo.....	25
Sự khác nhau giữa máy ảo và Docker Container	28
CHƯƠNG 3: XÂY DỰNG DEMO	30
3.1 Giới thiệu chung về Demo.....	30
3.2 Ngôn ngữ lập trình và công cụ sử dụng	30
3.2.1 JavaScript	30
3.2.2 NodeJs	30
3.2.3 Express js.....	31
3.2.3 Hệ quản trị cơ sở dữ liệu MongoDB	31

3.2.4 Visual Studio code.....	32
3.3 Hình ảnh Demo về ứng dụng.....	33
3.3.1 Giao diện trang chủ	33
3.3.2 Giao diện quản lý bài viết.....	33
3.3.3 Giao diện thêm bài viết.....	34
3.3.4 Giao diện chi tiết bài viết.....	34
3.4 Triển khai ứng dụng với Docker	35
KẾT LUẬN	37
TÀI LIỆU THAM KHẢO	38

CHƯƠNG 1: GIỚI THIỆU CHUNG

1.1. Tổng quan về DevOps và Developer

1.1.1 DevOps

DevOps (kết hợp của cụm từ tiếng Anh "software DEvelopment" và "information technology OPerationS") là một thuật ngữ để chỉ một tập hợp các hành động trong đó nhấn mạnh sự hợp tác và trao đổi thông tin của các lập trình viên và chuyên viên tin học khi cùng làm việc để tự động hóa quá trình chuyển giao sản phẩm phần mềm và thay đổi kiến trúc hệ thống. Điều này nhằm thiết lập một nền văn hóa và môi trường nơi mà việc build, kiểm tra, và phát hành phần mềm có thể xảy ra nhanh chóng, thường xuyên, và đáng tin cậy hơn.

1.1.2. Developer

Developer (gọi tắt là Dev) là những người dùng ngôn ngữ lập trình để sáng tạo, xây dựng và bảo trì các chương trình, phần mềm, ứng dụng. Một số ngôn ngữ lập trình thường được sử dụng như Python, JavaScript, C, C++, C#,...

Developer thường đóng vai trò quan trọng trong quá trình tạo ra phần mềm. Bởi họ chính là người sử dụng thành thạo các ngôn ngữ lập trình để viết ra các đoạn mã lập trình. Hiện nay, rất nhiều bạn trẻ lựa chọn theo đuổi công việc này bởi những tiềm năng cũng như đãi ngộ mà nó mang lại.

Developer là một tên gọi khá chung cho các kỹ sư phần mềm. Trên thực tế, các developer thường có chuyên môn khác nhau. Và những công việc của họ sẽ xoay quanh chuyên môn đó. Cụ thể công việc của Developer là gì?

- Frontend Developer
- PHP Developer
- Backend Developer
- iOS Developer
- Android Developer

1.2. Docker cho devops và developer

1.2.1. Docker cho devops

Docker như một công cụ phù hợp hoàn toàn tốt trong hệ sinh thái DevOps. Nó được xây dựng cho các công ty phần mềm hiện đại đang bắt kịp với những thay đổi nhanh chóng trong công nghệ. Không thể bỏ qua Docker trong chuỗi công cụ DevOps; nó đã trở thành một công cụ thực tế và gần như không thể thay thế.

Những lợi ích của Docker với việc hỗ trợ DevOps là các use case và lợi thế mà nó mang lại cho quy trình phát triển phần mềm bằng cách container hóa các ứng dụng giúp phát triển dễ dàng và chu kỳ phát hành nhanh.

Docker có thể giải quyết hầu hết các vấn đề Dev và Ops, cho phép cả hai nhóm cộng tác hiệu quả và hoạt động hiệu quả.

Theo Báo cáo của RightScale 2019, Docker đã chiến thắng trong lĩnh vực container với sự tăng trưởng áp dụng YoY tuyệt vời.

Với Docker, doanh nghiệp có thể tạo ra các môi trường dev, thử nghiệm và sản xuất bất biến. Họ sẽ có mức độ kiểm soát cao đối với tất cả các thay đổi vì chúng được thực hiện bằng cách sử dụng Docker container và image bất biến. Họ luôn có thể quay lại phiên bản trước tại bất kỳ thời điểm nào nếu muốn.

Môi trường phát triển, thử nghiệm và sản xuất trở nên giống nhau hơn. Với Docker, đảm bảo rằng nếu một tính năng hoạt động trong môi trường phát triển, nó cũng sẽ hoạt động trong thử nghiệm và sản xuất.

1.2.2. Docker cho Developer

Docker giúp developer dễ dàng phát triển và triển khai các ứng dụng bên trong các môi trường ảo được container hóa. Điều này có nghĩa là các ứng dụng chạy giống nhau cho dù chúng ở đâu và chúng đang chạy trên máy nào.

Docker container có thể được triển khai cho bất kỳ máy nào mà không gặp vấn đề về sự tương thích, vì vậy hệ thống phần mềm sẽ giữ nguyên agnostic, giúp phần mềm sử dụng đơn giản hơn, ít phải phát triển hơn và dễ bảo trì và triển khai.

Một developer thường sẽ bắt đầu bằng cách truy cập Docker Hub, kho lưu trữ đám mây trực tuyến của các container Docker và kéo về một container chứa môi trường được cấu hình sẵn cho ngôn ngữ lập trình cụ thể của họ, như Ruby hoặc NodeJS, với tất cả các tệp và framework cần có để bắt đầu.

Docker là một trong những công cụ thực sự tuân theo tiêu chí "Xây dựng, Vận chuyển và Chạy".

Trên toàn thế giới và trên toàn ngành, rất nhiều công ty và viện nghiên cứu đang sử dụng Docker để tăng tốc các hoạt động phát triển của họ. PayPal hiện có hơn 700 ứng dụng và họ đã chuyển đổi tất cả chúng thành các ứng dụng dựa trên container. Họ chạy 150.000 container và điều này đã giúp họ tăng 50% năng suất dev.

MetLife, một ví dụ tuyệt vời khác, đã tiết kiệm rất lớn cho infrastructure của họ nhờ sử dụng ít hệ điều hành hơn nhưng quản lý được nhiều ứng dụng hơn. Điều này mang lại cho họ rất nhiều phần cứng, và tiết kiệm rất nhiều tiền cho infrastructure và giảm chi phí. Sau khi chuyển sang Docker, MetLife đã giảm 70% chi phí VM, ít hơn 67% CPU, giảm gấp 10 lần sử dụng CPU trung bình và giảm 66% chi phí. Đó là sức mạnh của Docker.

1.3. Lý do lựa chọn Docker

Thời gian “start” và “stop” cực ngắn:

Lý do đầu tiên mà Docker đem lại đó chính là thời gian bắt đầu và dừng cực ngắn. Docker được so sánh với các máy ảo và cho thấy sự khác biệt của mình. Thời gian trong hai công đoạn này của Docker nhanh hơn, ấn tượng hơn hẳn.

Tự do trong chọn hệ thống:

Lý do thứ hai của Docker chính là lập trình viên có thể tự do chọn hệ thống. Người dùng có thể tiến hành khởi chạy container trong bất cứ hệ thống nào họ muốn. Đây chính là lợi ích cực kỳ độc đáo mà Docker đem lại.

Tốc độ làm việc nhanh:

Thời gian với dân lập trình chắc hẳn là điều rất quan trọng. Tốc độ làm việc của các nền tảng luôn được đặt lên hàng đầu. Thao tác build và loại bỏ được thực hiện bởi container nhanh hơn so với máy ảo. Chính vì thế, đến với Docker các bạn sẽ có tốc độ cũng như hiệu suất làm việc nhanh hơn. Đây chính là lời giải đáp cho câu hỏi tính năng nổi bật của docker là gì?

Đơn giản trong việc thiết lập môi trường: Các lập trình viên khi sử dụng Docker chỉ cần config 1 lần. Họ sẽ không cần cài đặt lại Dependencies sau đó nữa. Nếu có sự thay đổi mới về thành viên hay thiết bị, người dùng chỉ cần chia sẻ config mà thôi.

Hỗ trợ xóa: Lợi ích nổi bật cuối cùng của Docker chính là giúp cho word-space sạch sẽ hơn. Trong trường hợp lập trình viên cần xóa những môi trường có ảnh hưởng tới phần khác. Thì Docker hỗ trợ xóa nhanh chóng điều này.

1.4. Phạm vi của chủ đề

Giới thiệu về Docker và cách sử dụng, triển khai Docker cho 1 dự án cụ thể.

Tìm hiểu về tầm quan trọng của docker trong giai đoạn phát triển phần mềm, tầm quan trọng của Docker đối với devops và developer.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Tổng quan về Docker

2.1.1. Định nghĩa

Docker là một dự án mã nguồn mở giúp tự động triển khai các ứng dụng Linux và Windows vào trong các container ảo hóa. Docker cung cấp một lớp trừu tượng và tự động ảo hóa dựa trên Linux.



Docker là một công cụ giúp cho việc tạo ra và triển khai các container để phát triển, chạy ứng dụng được dễ dàng. Các container là môi trường, mà ở đó lập trình viên đưa vào các thành phần cần thiết để ứng dụng của họ chạy được, bằng cách đóng gói ứng dụng cùng với container như vậy, nó đảm bảo ứng dụng chạy được và giống nhau ở các máy khác nhau (Linux, Windows, MacOS, Windows Server ...).

Docker có vẻ rất giống máy ảo (nhiều người từng tạo máy ảo với công cụ ảo hóa như Virtualbox, VMWare), nhưng có điểm khác với VM: thay vì tạo ra toàn bộ hệ thống (dù ảo hóa), Docker lại cho phép ứng dụng sử dụng nhân của hệ điều hành đang chạy Docker để chạy ứng dụng bằng cách bổ sung thêm các thành phần còn thiếu cung cấp bởi container. Cách này làm tăng hiệu suất và giảm kích thước ứng dụng.

Hiểu đơn giản hơn thì Docker cung cấp cho người sử dụng những service và những công cụ cần thiết để giúp người sử dụng có thể chạy các chương trình và đóng gói chúng ở trong tất cả môi trường khác nhau một cách nhanh chóng và đơn giản nhất.

Tóm lại, Docker là một công cụ giúp deploy dự án phần mềm một cách nhanh chóng trong các container. Các môi trường liên quan cần thiết để phần mềm hoạt động đều được Docker cung cấp đầy đủ. Triển khai dự án phần mềm một cách nhanh chóng và dễ dàng. Có rất nhiều lợi ích đáng để sử dụng Docker.

2.1.2. Lịch sử ra đời

Solomon Hykes bắt đầu tạo ra Docker khi làm việc ở Pháp, trong một dự án nội bộ của dotCloud, một công ty nền tảng như dịch vụ ban đầu có thêm sự đóng góp của các kỹ sư dotCloud là Andrea Luzzardi và Francois-Xavier Bourlet.[Jeff Lindsay cũng đã tham gia như là một nhà cộng tác độc lập.

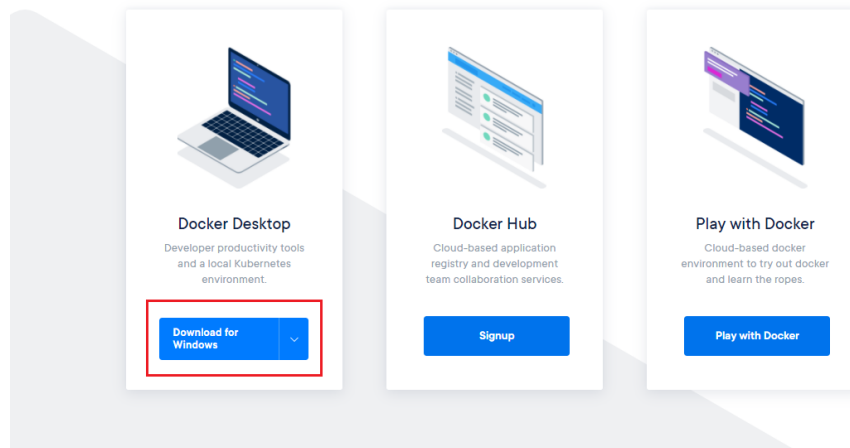
Docker được phát hành dạng mã nguồn mở trong tháng 3 năm 2013. Vào ngày 13 năm 2014, với phiên bản 0.9, Docker bỏ đi LXC và thay thế nó với thư viện của mình là libcontainer được viết bằng ngôn ngữ Go. Tới tháng 10 năm 2015, dự án Docker đã có hơn 25,600 sao trên GitHub (trở thành top 20 dự án có số sao cao nhất trên GitHub), có hơn 6,800 fork, và gần 1.100 lập trình viên tham gia đóng góp.

2.1.3. Cài đặt Docker trên Window

Tải và cài đặt Docker cho window tại <https://www.docker.com/get-started>

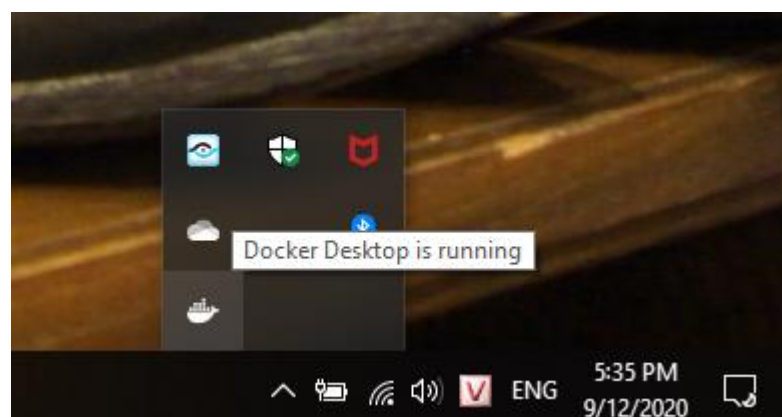
Get Started with Docker

We have a complete container solution for you - no matter who you are and where you are on your containerization journey.



Sau khi đã tải về, các bạn tiến hành cài đặt thông qua file *Docker Desktop Installer.exe* vừa tải về. Như mình đã giới thiệu, để chạy được docker cần bật tính năng Hyper-V của windows và trong quá trình cài đặt docker nếu windows chưa được bật Hyper-V, sẽ có một checkbox hỏi xem có muốn bật Hyper-V luôn không, thì mình nên lựa chọn checkbox để bật Hyper-V luôn.

Sau khi cài đặt hoàn tất, cần khởi động lại máy để có thể chạy docker, sau khi khởi động chúng ta có thể thấy biểu tượng Docker ở Tray Icon:



Để kiểm tra thông tin chi tiết: chúng ta thực hiện lệnh cmd : `>docker info`

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1457]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\>docker info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 19.03.12
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 4.19.76-linuxkit
 Operating System: Docker Desktop
 OSType: linux
 Architecture: x86_64
 CPUs: 2
 Total Memory: 1.922GiB
 Name: docker-desktop
 ID: A5FU:DKTX:QQDK:2EMD:CTVF:T4NH:KJLM:OF3Y:U6NC:EZLT:UI5A:2MVP
 Docker Root Dir: /var/lib/docker
 Debug Mode: true
  File Descriptors: 39
  Goroutines: 50
  System Time: 2020-09-12T10:33:09.509556572Z
  EventsListeners: 3
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
 Product License: Community Engine
```

Để kiểm tra phiên bản của Docker chúng ta thực hiện lệnh cmd: >docker --version

```
C:\> Administrator: Command Prompt

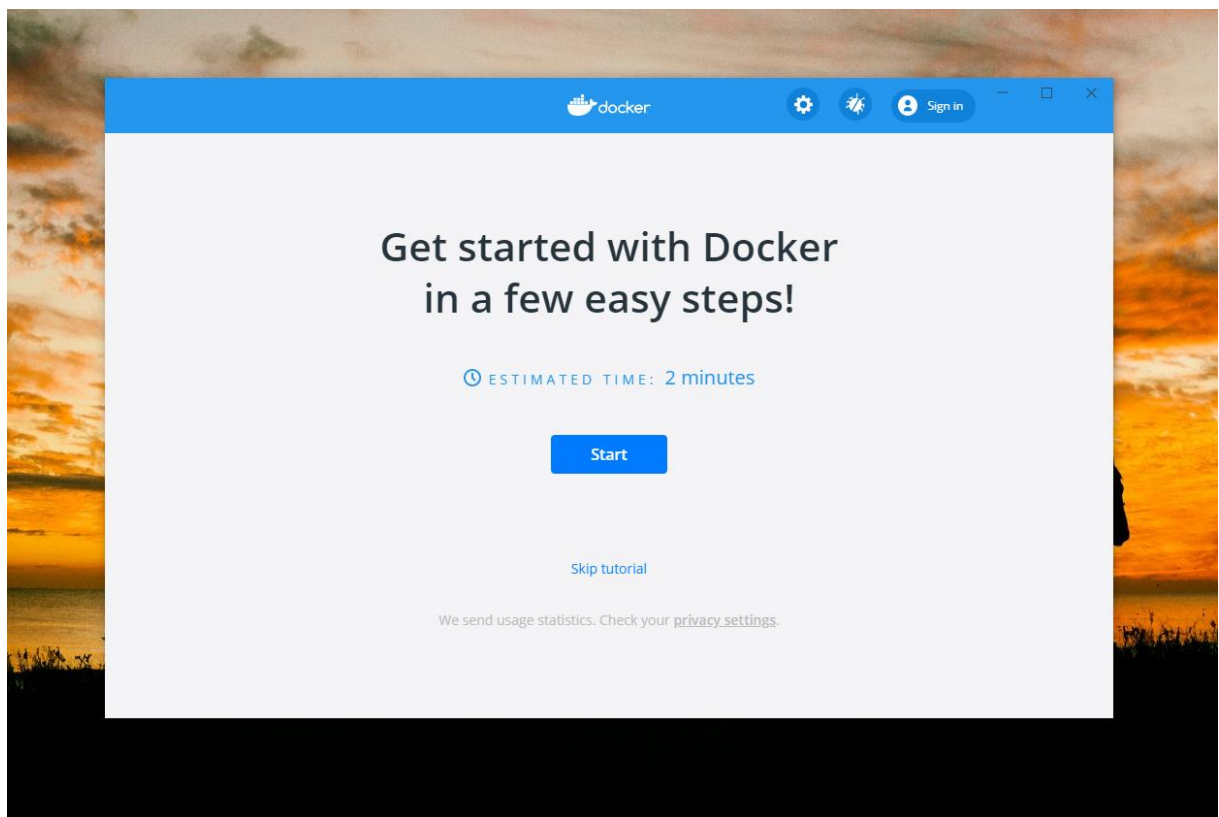
Microsoft Windows [Version 10.0.17763.1457]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\[redacted]>docker --version
Docker version 19.03.12, build 48a66213fe

C:\Users\[redacted]>
```

Khi quá trình khởi chạy hoàn tất, chúng ta có một màn hình giới thiệu khi mở Docker Desktop. Trong hướng dẫn đã có một số bài tập đơn giản để chúng ta xây dựng và làm quen với Docker.

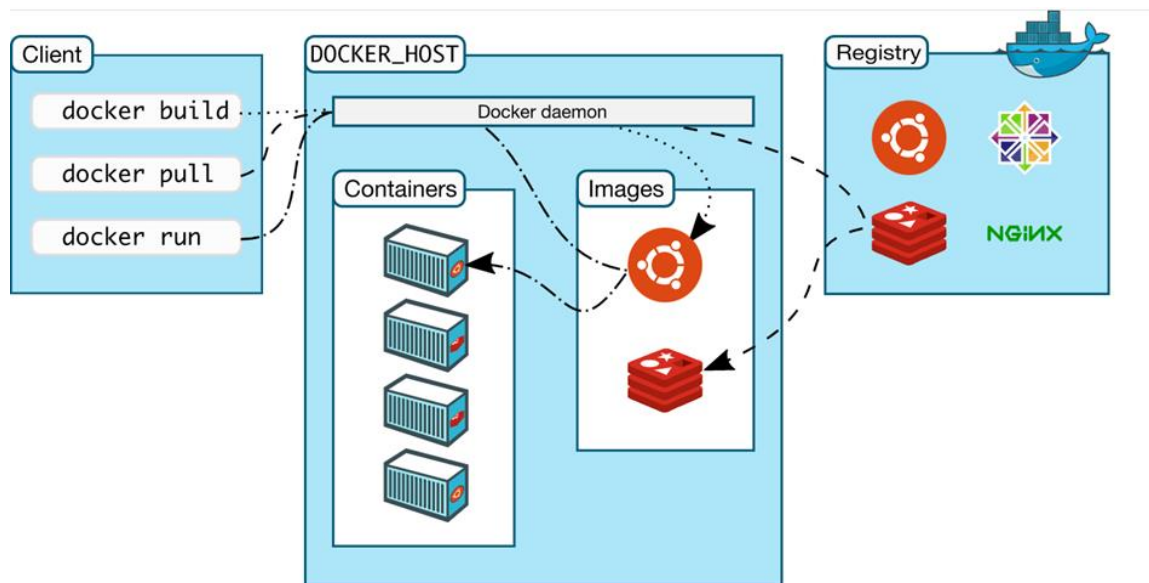
Khi quá trình khởi chạy hoàn tất, chúng ta có một màn hình giới thiệu khi mở Docker Desktop. Trong hướng dẫn đã có một số bài tập đơn giản để chúng ta xây dựng và làm quen với Docker.



2.2. Các thành phần cơ bản của Docker

2.2.1. Docker Engine

Docker Engine (Docker Daemon) là một thành phần quan trọng của Docker. Nó chính là thành phần chính giúp vận hành Docker. Có thể hiểu đây chính là cốt lõi của Docker.



Docker Engine được cài trên các máy chủ sử dụng Docker (Docker Host), nó chứa các đối tượng là thành phần con của docker (Container, Image) quản lý và vận hành chúng.

Docker Engine là công cụ Client - Server hỗ trợ công nghệ container để xử lý các nhiệm vụ và quy trình công việc liên quan đến việc xây dựng các ứng dụng dựa trên vùng chứa (container). Engine tạo ra một quy trình daemon phía máy chủ lưu trữ images, containers, networks và storage volumes. Daemon cũng cung cấp giao diện dòng lệnh phía máy khách (CLI) cho phép người dùng tương tác với daemon thông qua giao diện lập trình ứng dụng Docker.

2.2.2. Docker Image

Docker Image là một file có chứa tất cả các source code, libraries, dependencies, tools và các files khác cần thiết để ứng dụng có thể hoạt động.

Docker Image chỉ là các mẫu để từ đó build các Container tương ứng. Docker Image không cho phép deploy trực tiếp phần mềm lên chúng.

Image là thành phần để đóng gói ứng dụng và các thành phần mà ứng dụng phụ thuộc để chạy. Image có thể lưu local hoặc lưu Registry

Hiểu theo cách đơn giản nhất, Docker Image giống như hình ảnh mô phỏng của một môi trường thực thi với mục đích nhất định. Chúng chứa các source code, các thư viện để phục vụ cho việc chạy một phần mềm.

Ví dụ để hiểu về các Image: Image npm dùng chạy nodeJs, Image Java JDK dùng để chạy Java, Python . . .

2.2.3. Docker Container

Nếu như coi Docker Image là một class thì Docker Container là một instance của image. Container cung cấp môi trường thực cho ứng dụng hoạt động. Ở đây, người dùng có thể chạy các ứng dụng phần mềm.

Container nắm giữ toàn bộ các package cần thiết để khởi động và chạy ứng dụng. Các Container sẽ sử dụng chung tài nguyên của hệ thống nên rất nhẹ và giúp các thao tác kết nối, tương tác diễn ra nhanh chóng, tiện lợi hơn.

Container có công dụng giống hệt với tên của nó. Container như một “vùng chứa”. Source code phần mềm của bạn khi deploy sẽ chạy trực tiếp trong container này. Bạn có thể hình dung khi đó container như một server riêng, máy ảo riêng để chạy phần mềm.

Ưu điểm của Container là rất nhẹ và dễ dàng tạo, xóa. Đây chính là một trong những ưu điểm lớn nhất của docker container so với các công nghệ máy ảo khác.

2.2.4. Dockerfile và Docker Compose

a, Docker file

Dockerfile là thành phần của hệ sinh thái Docker, các Docker file khi build sẽ tạo thành Docker Image. Chính nhờ Dockerfile mà lập trình viên có thể tự xây dựng các Image, Container cần thiết cho project của bạn.

Dockerfile là một file dạng text không có đuôi, Dockerfile có cấu trúc theo quy chuẩn nhất định của Docker. Nội dung gồm tập hợp các câu lệnh cũng như phụ thuộc cần thiết để tạo ra Docker Image.

Dockerfile sinh ra giúp việc deploy thực nhanh hơn rất nhiều, khi triển khai phần mềm chỉ cần có dockerfile. Điều này giúp tạo Docker Image và Container nhanh hơn rất nhiều.

b, Docker Compose

Docker Compose là công cụ dùng để định nghĩa và run multi-container cho Docker application. Docker Compose, chúng ta sử dụng một file YAML để thiết lập các service cần thiết cho chương trình. Cuối cùng, với một câu lệnh, chúng ta sẽ tạo và khởi động tất cả các service từ các thiết lập đó.

Khi phát triển một chương trình, việc chạy một chương trình trong một môi trường cô lập và tương tác là rất cần thiết. Compose cho phép thiết lập và chạy tất cả các service cần thiết cho chương trình. Chỉ với một câu lệnh docker-compose up, các service đó sẽ được chạy với các container tương ứng.

Ví dụ project của bạn cần một Image có cài cả hai môi trường NodeJS và Python thì trong Dockerfile bạn có gọi chúng vào là được. Tất cả việc còn lại Docker sẽ giúp bạn

2.2.5. Docker Hub

Docker Hub là Registry, Container Image Library chính thức của Docker, là một dịch vụ do Docker cung cấp, cho phép tìm kiếm và chia sẻ các Container Image.

Các tính năng chính của Docker Hub là:

Repositories: Push và pull container images.

Teams & Organizations: Quản lý quyền truy cập vào private repositories của container images.

Official Images: Pull sử dụng container images chất lượng cao của Docker.

Publisher Images: Pull và sử dụng container images được cung cấp bởi vendors khác.

Builds: Tự động tạo container images từ GitHub và Bitbucket. Push chúng lên Docker Hub.

Webhooks: Kích hoạt các actions sau khi push thành công một repository lên Docker Hub với các dịch vụ khác.

2.3. Tính Chất của docker

2.3.1. Ưu điểm

Tính đồng nhất: Đây cũng chính là ưu điểm nổi bật nhất khi sử dụng Docker. Trong trường hợp nhiều người cùng phát triển một dự án trong môi trường, việc sử dụng Docker sẽ giúp hạn chế được sự sai khác nhất định giữa thành viên.

Tính nhất quán: Với Docker, bạn có thể test container được dùng để phát triển bằng CI. Dễ dàng deploy container đã được test bằng CI lên server. Ngoài ra, bạn cũng có thể thực hiện scale container đã được deploy.

Tính đóng gói: Với Docker, bạn có thể ần môi trường bao gồm cả App vào Container, có thể test được Container đồng thời cũng có thể dễ dàng bỏ hay tạo Container.

Dễ dàng sử dụng: Docker rất dễ cho mọi người sử dụng từ developers, systems admins, architects...v...v.. nó tận dụng lợi thế của container để build, test nhanh chóng.

Có thể đóng gói ứng dụng trên laptop của họ và chạy trên public cloud, private cloud..v.v... “Build once, run anywhere”.

Tốc độ: Docker container rất nhẹ và nhanh, bạn có thể tạo và chạy docker container trong vài giây so sánh với VMs thì mỗi lần chạy VMs cần rất nhiều thời gian khởi động.

Khả năng di động: môi trường develop được dựng lên bằng docker có thể chuyển từ người này sang người khác mà không làm thay đổi cấu hình ở trong.

Chia sẻ: DockerHub là một “app store for docker images”. Trên DockerHub có hàng ngàn public images được tạo bởi cộng đồng. Dễ dàng tìm thấy những image mà bạn cần và chỉ cần pull về và sử dụng với một số sửa đổi nhỏ.

Môi trường chạy và khả năng mở rộng: Bạn có thể chia nhỏ những chức năng của ứng dụng thành các container riêng lẻ. Ví dụ Database chạy trên một container và Redis cache có thể chạy trên một container khác trong khi ứng dụng Node.js lại chạy trên một cái khác nữa. Với Docker, rất dễ để liên kết các container với nhau để tạo thành một ứng dụng, làm cho nó dễ dàng scale, update các thành phần độc lập với nhau.

2.3.2. Nhược điểm

- Nó không cung cấp một tùy chọn lưu trữ.
- Theo dõi tài tệ.
- Không có lập trình lại tự động các nút không hoạt động.
- Các hành động phải được thực hiện trong CLI.
- Quản lý thủ công nhiều trường hợp.
- Bạn cần hỗ trợ các công cụ khác.
- Triển khai cụm thủ công phức tạp.
- Không hỗ trợ kiểm tra sức khỏe.
- Docker là một công ty hoạt động vì lợi nhuận và một số thành phần quan trọng của nó, chẳng hạn như Docker Engine và Docker Desktop, không phải là mã nguồn mở.

2.4. Quy trình triển khai docker

2.4.1. Quy trình

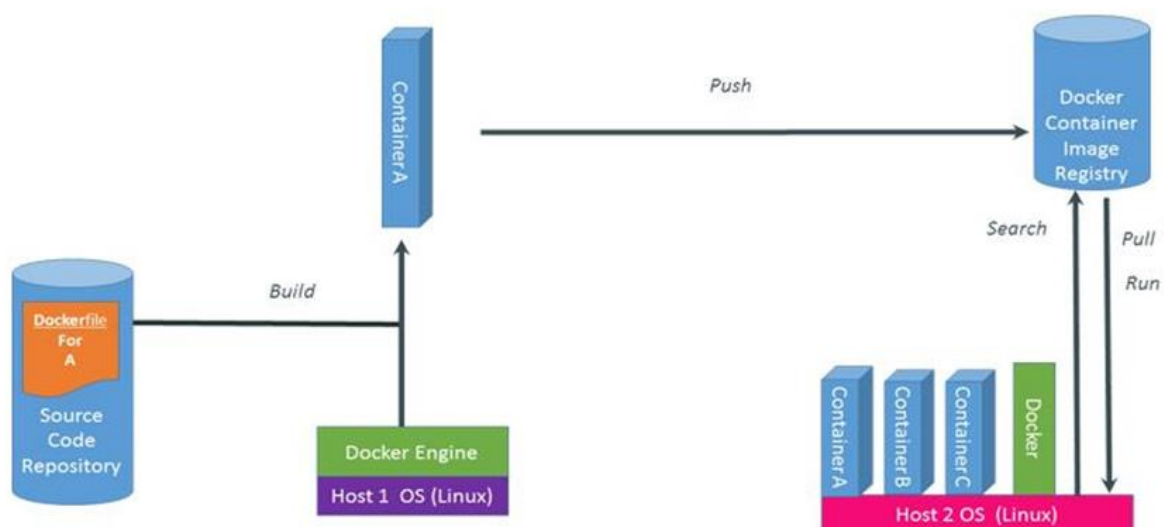
Docker được xây dựng dựa trên 3 thành phần chính:

- Docker daemon: daemon (server) nhận lệnh từ docker client thông qua CLI hoặc RestAPI
- Docker client: Có thể ở trên cùng một host hoặc khác host với docker daemen
- Docker registry (hub): là một dịch vụ máy chủ cho phép lưu trữ các docker image của cá nhân, công ty, team,... Dịch vụ Docker Registry có thể được cung cấp bởi tổ chức thứ 3 hoặc là dịch vụ nội bộ được xây dựng riêng nếu bạn muốn. Một số dịch vụ Docker Registry phổ biến như : Azure Container Registry, Docker Hub...

Một hệ thống Docker thường sẽ được thực thi dựa theo 3 bước chính như sau:

Bước 1: Build

Đầu tiên, bạn cần tạo một dockerfile, trong dockerfile này thì nó chính là code của chúng ta. Nó sẽ được Build tại một máy tính đã được cài đặt sẵn Docker Engine. Sau khi đã build, thì ta sẽ có được Container, trong Container đã có chứa ứng dụng đi kèm bộ thư viện của chúng ta.



Bước 2: Push

Sau khi đã có được Container thì chúng ta cần phải thực hiện Push Container này lên cloud rồi lưu nó tại đó.

Bước 3: Pull và Run

Nếu như một máy tính khác đang muốn sử dụng Container thì chúng ta cần buộc máy thực hiện việc Pull container này về máy. Tất nhiên, máy này cũng cần phải được cài Docker Engine, sau đó sẽ thực hiện Run Container.

2.4.2. Các câu lệnh thường dùng với Docker Container

- Docker container commit: Tạo image mới từ những thay đổi của container.
- Docker container create: Tạo container mới.
- Docker container exec: Chạy các command khi container đang hoạt động.
- Docker container kill: Chấm dứt hoạt động của một hoặc nhiều container.
- Docker container pause: Tạm dừng tất cả tiến trình bên trong một hoặc nhiều container.
- Docker container run: Chạy các command trong một container mới.
- Docker container start: Chạy một container hoặc nhiều container đã dừng.
- Docker container rename: Đổi tên container.
- Docker container restart: Khởi động lại một hoặc nhiều container.

2.4.3. Các câu lệnh thường dùng với Docker Image

- Docker images: Liệt kê các image
- Docker image build: Build image từ file Dockerfile.
- Docker image history: Hiện thị lịch sử của image.
- Docker image import: Import nội dung từ tarball để tạo ra filesystem của image.
- Docker image inspect: Hiện thị thông tin chi tiết của một hoặc nhiều image.
- Docker image load: Nạp image từ file *.tar hoặc STDIN.

- Docker image prune: Xóa các image không sử dụng.
- Docker image pull: Pull một image hoặc repository từ Docker HUB đăng ký.
- Docker image push: Đẩy image, repository lên Docker HUB.
- Docker image save: Lưu một hoặc nhiều image vào file *.tar.
- Docker image tag: Gắn tag cho TARGET IMAGE tương ứng với SOURCE IMAGE

2.5. So sánh Docker với Kubernetes

Khi hầu hết mọi người nói về “Kubernetes và Docker”, ý của họ thực sự là “Kubernetes và Docker Swarm”. Về sau, Docker đã tự xây dựng giải pháp cluster cho các container Docker, và lợi thế là được tích hợp chặt chẽ vào hệ sinh thái của Docker và sử dụng API của riêng nó. Giống như hầu hết các bộ lập lịch, Docker Swarm cung cấp cách quản lý một số lượng lớn các container trải rộng trên các cụm máy chủ. Hệ thống lọc và lập lịch của nó cho phép lựa chọn các node tối ưu trong một cụm để deploy các container.

2.5.1. Tổng quan về Kubernetes

Kubernetes là nền tảng được xây dựng dựa trên nhiều năm kinh nghiệm của Google về việc chạy workload ở quy mô lớn trong quy trình production. Theo định nghĩa tại website của Kubernetes, “Kubernetes là một hệ thống nguồn mở (open-source system) để tự động hóa việc triển khai, thay đổi kích thước và quản lý các ứng dụng được container hoá”.

2.5.2. Tổng quan Docker Swarm

Docker Swarm là công cụ điều phối container của Docker, sử dụng Docker API tiêu chuẩn và networking nhằm giúp người dùng dễ dàng thích nghi với môi trường làm việc của các Docker container. Docker Swarm được thiết kế để hoạt động theo 04 nguyên tắc chủ đạo:

- Tính năng mạnh mẽ cùng thiết kế đơn giản, thân thiện với người dùng.
- Kiến trúc resilient zero single-point-of-failure

- Bảo mật mặc định với các certificate được tạo tự động
- Khả năng tương thích với các component (thành phần) hiện có

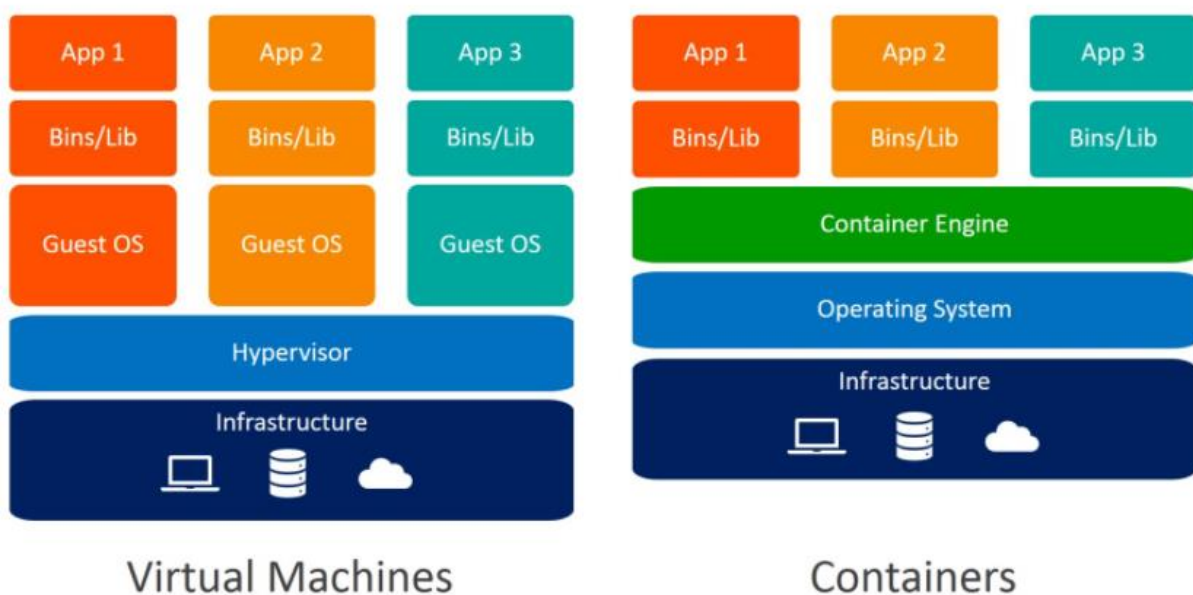
2.5.3. So sánh Kubernetes vs Docker Swarm

	Kubernetes	Docker
Định nghĩa	một ứng dụng có thể được deploy bằng cách sử dụng kết hợp các pod, deployments và services (hoặc micro-services).	các ứng dụng có thể được deploy như một service (hoặc micro-service) trong một Swarm cluster. File YAML có thể được dùng để cụ thể hoá multi-container. Hơn nữa, Docker Compose có thể deploy ứng dụng.
Cài đặt và thiết lập	bước install được thực hiện thủ công và cần có kế hoạch cụ thể để Kubernetes hoạt động trơn tru. Hướng dẫn cài đặt thường được không thống nhất giữa các nhà cung cấp. Ngoài ra, cần nắm được cấu hình cluster như địa chỉ IP của một node hoặc nhiệm vụ của mỗi node.	Với Docker, chỉ cần một bộ công cụ tùy chọn để build theo môi trường và cấu hình. Docker Swarm cũng cung cấp tính linh hoạt bằng cách cho phép tất cả các node mới tham gia vào một cluster hiện có với tư cách là manager hoặc worker.
Yêu cầu	Kubernetes yêu cầu kiến thức về CLI (Command Line Interface) để chạy trên Docker. Cần hiểu về Docker CLI để điều hướng bên trong một cấu trúc, sau đó bổ sung infrastructure ngôn ngữ chung Kubernetes để chạy các program đó.	sử dụng ngôn ngữ chung để điều hướng trong một cấu trúc. Điều này cung cấp tính biến thiên và tốc độ cho công cụ này.

Giám sát	Kubernetes hỗ trợ nhiều phiên bản logging (ghi nhật ký) và giám sát khi các service được triển khai trong cluster	Docker Swarm được hỗ trợ để chỉ giám sát với các ứng dụng của bên thứ ba. Lời khuyên là nên sử dụng Docker với Reimann để giám sát, tuy nhiên vì Docker Swarm có API mở, nên việc kết nối với nhiều ứng dụng dễ dàng hơn.
Khả năng thay đổi quy mô	Kubernetes là một all-in-one framework cho các hệ thống phân tán. Đây là một hệ thống phức tạp vì nó cung cấp một bộ API thống nhất và đảm bảo mạnh mẽ về trạng thái cluster, làm chậm deployment và thay đổi quy mô container.	So với Kubernetes, Docker Swarm có thể deploy các container nhanh hơn; điều này cho phép thời gian phản ứng nhanh để thay đổi quy mô theo yêu cầu.
Tính sẵn sàng	Trong Kubernetes, tất cả các pod được phân phối giữa các node và điều này cung cấp tính sẵn sàng cao bằng cách chấp nhận lỗi ứng dụng. Hơn nữa, các load-balancing của Kubernetes phát hiện các pod không lành mạnh và loại bỏ chúng, điều này hỗ trợ tính sẵn sàng cao.	Docker Swarm cũng cung cấp tính sẵn sàng cao vì các service có thể được nhân bản trong các node Swarm. Trong Docker Swarm, các node quản lý Swarm chịu trách nhiệm cho toàn bộ cluster và quản lý tài nguyên của các node worker.

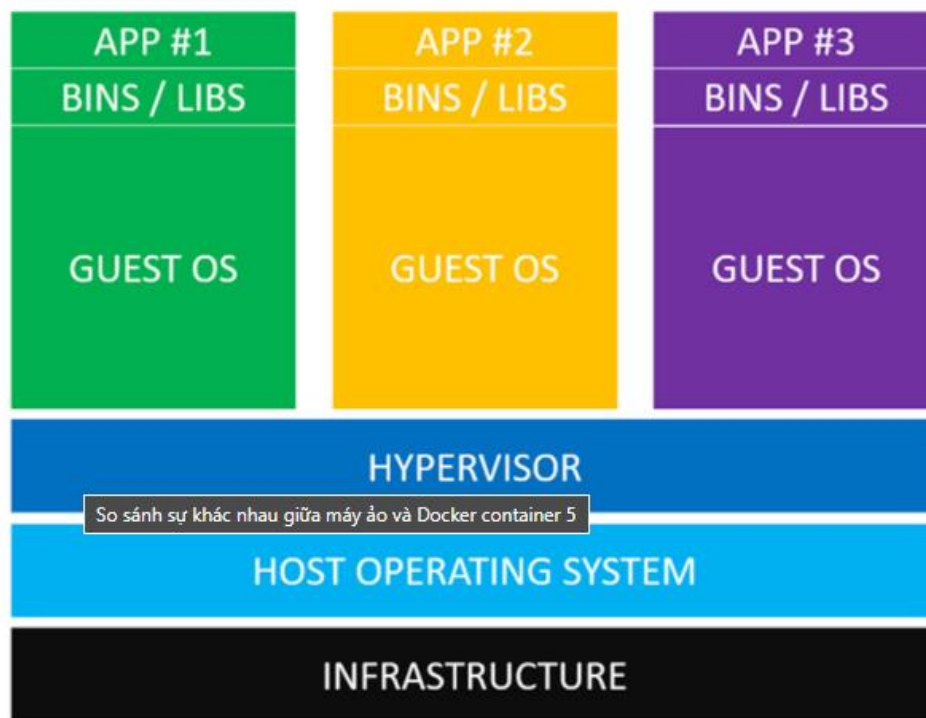
Networking	Kubernetes network có tính chất phẳng vì nó cho phép tất cả các pod giao tiếp với nhau. Trong Kubernetes, model yêu cầu 02 CIDR: 01 CIDR để yêu cầu các pod lấy địa chỉ IP, 01 CIDR là cho các service.	Trong Docker Swarm, một node tham gia một cluster tạo ra một overlay network các service bao trùm tất cả các host trong Swarm và một network cầu nối Docker duy nhất cho các container. Người dùng Docker Swarm có thể tùy chọn mã hóa data traffic (lưu lượng dữ liệu) container khi tự tạo overlay network.
------------	---	---

2.6. so sánh Docker với máy ảo



Kiến trúc của service hoạt động trên máy ảo

Máy ảo (Virtual Machine – VM) là một mô phỏng của hệ thống máy tính. Nói một cách đơn giản, công nghệ này giúp người dùng có thể tạo ra nhiều “máy tính logic” trên một “máy tính vật lý”, dễ dàng quản lý vấn đề bảo mật. Kiến trúc của service khi hoạt động trên nền tảng máy ảo thông qua ảnh minh họa bên dưới:



Kiến trúc của service trên máy ảo.

Kiến trúc của service trên máy ảo.

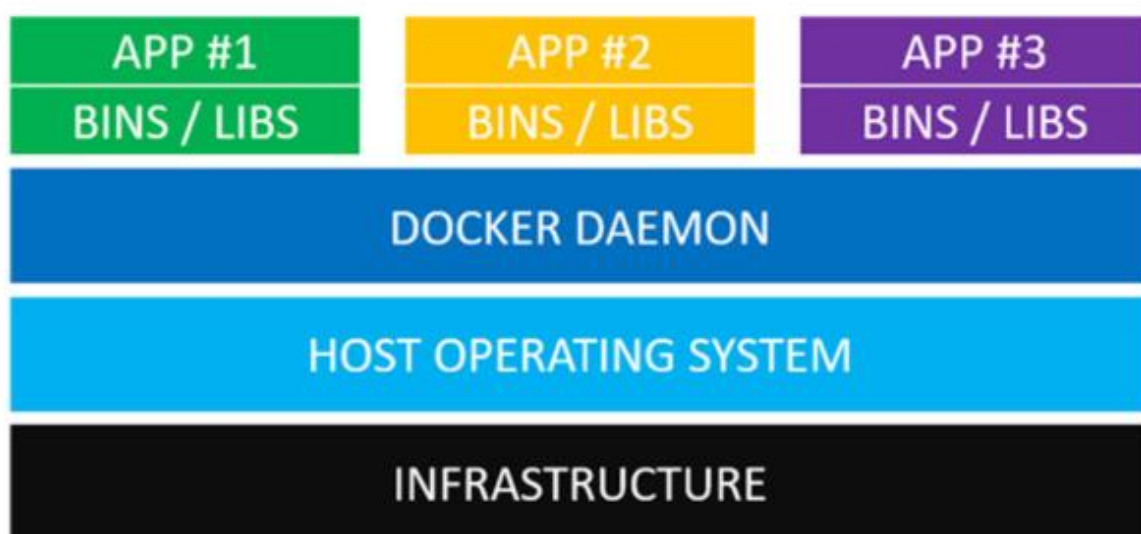
- **Infrastructure (hạ tầng):** có thể là laptop, một server (máy chủ) chuyên dụng (dedicated server) hoạt động trong một trung tâm dữ liệu (Data Center) hoặc một server riêng ảo (Virtual Private Server – VPS) trên cloud như DigitalOcean, Amazon, GCP,...
- **Host Operating System (hệ điều hành của server):** hệ điều hành đang sử dụng cho server đang sử dụng, có thể là macOS, Windows hoặc Linux.
- **Hypervisor** (phần mềm giám sát máy ảo): có thể coi máy ảo như một máy tính độc lập được đóng gói vào một tập tin duy nhất, nhưng cần phải có một phần mềm để có thể chạy tập tin đó. Các hypervisor được sử dụng để tạo, startup, dừng và reset lại các máy ảo, cho phép mỗi máy ảo hoặc “guest” truy cập vào lớp tài nguyên phần cứng vật lý bên dưới, chẳng hạn như CPU, RAM và ổ cứng. Phần mềm này cũng có thể giới hạn số lượng tài nguyên hệ thống mà mỗi máy ảo có thể sử dụng để đảm bảo cho nhiều máy ảo cùng hoạt động đồng thời trên một hệ thống. Có thể kể đến các hypervisor phổ biến trên thị trường hiện nay như HyperKit cho macOS,

Hyper-V cho Windows và KVM cho Linux. Ngoài ra, 2 hypervisor phổ biến khác là VirtualBox và VMWare.

- **Guest OS (hệ điều hành của máy ảo):** với mỗi máy ảo được tạo ra, người quản trị cần phải cài đặt một hệ điều hành đi kèm cho máy ảo đó để cài đặt và triển khai các service cần thiết. Ví dụ bạn cần triển khai 3 service trên 3 máy ảo khác nhau và mỗi “Guest OS” sẽ tiêu tốn ít nhất 700 MB dung lượng ổ cứng, vậy sẽ tiêu tốn khoảng 2.1 GB dung lượng ổ cứng của server để tạo 3 máy ảo để triển khai 3 service khác nhau. Chưa kể để các “Guest OS” này sẽ sử dụng một lượng tài nguyên khác của hệ thống như CPU, RAM.
- **Bins / Libs (các tập tin / thư viện cần thiết):** các service / application sẽ cần phải có các gói tập tin, thư viện đi kèm để có thể hoạt động.
- **App (application – service):** mã nguồn của các ứng dụng, phần mềm.

Kiến trúc của service hoạt động trên Docker container

Nếu so sánh với kiến trúc của service khi hoạt động trên nền tảng máy ảo, service hoạt động trên Docker container sẽ loại bỏ lớp “Guest OS”, với tốc độ khởi tạo service nhanh hơn hẳn so với việc sử dụng máy ảo. Docker Container sẽ giảm thiểu và đơn giản hóa các bản cập nhật bảo mật. Kiến trúc của service khi hoạt động trên nền tảng Docker container thông qua ảnh minh họa bên dưới:



Kiến trúc của service trên Docker container

Kiến trúc của service trên Docker container

- **Infrastructure (hạ tầng):** có thể là laptop, một server chuyên dụng (dedicated server) hoạt động trong một trung tâm dữ liệu (Data Center) hoặc một server riêng ảo (Virtual Private Server – VPS) trên cloud như DigitalOcean, Amazon, GCP,...
- **Host Operating System (hệ điều hành của server):** hệ điều hành đang sử dụng cho server đang sử dụng, có thể là macOS, Windows hoặc Linux.
- **Docker daemon (còn gọi là Docker Engine):** đây là service hoạt động trên server, được dùng để quản lý các thành phần cần thiết để khởi tạo và tương tác với Docker container.
- **Bins / Libs (các tập tin / thư viện cần thiết):** các gói tập tin, thư viện đi kèm của service được thêm vào Docker image.
- **App (application – service):** mã nguồn của các ứng dụng, phần mềm được thêm vào Docker container.

Sự khác nhau giữa máy ảo và Docker Container

Docker daemon có thể giao tiếp trực tiếp với hệ điều hành của server và phân bổ tài nguyên cho các Docker container đang chạy, đảm bảo mỗi container hoạt động độc lập với các container khác và hệ điều hành của server. Thay vì phải đợi một phút để máy ảo khởi động, người dùng có thể khởi động Docker container chỉ trong vài mili giây và tiết kiệm được rất nhiều dung lượng ổ đĩa và các tài nguyên hệ thống khác do không cần phải sử dụng “guest OS” chồng kênh cho mỗi ứng dụng. Người dùng sẽ không cần ảo hóa vì Docker chạy trực tiếp trên hệ điều hành của server.

Máy ảo	Docker container
Kích thước (dung lượng) lớn.	Kích thước (dung lượng) nhỏ.
Hiệu suất hạn chế.	Hiệu suất gốc (native).
Mỗi máy ảo sẽ có một hệ điều hành riêng.	Container sẽ sử dụng hệ điều hành của host.
Ảo hóa về mặt phần cứng	Ảo hóa về mặt hệ điều hành
Thời gian khởi động tính theo phút	Thời gian khởi động tính theo mili giây
Phân bổ bộ nhớ theo nhu cầu cần thiết	Yêu cầu ít dung lượng bộ nhớ hơn
Hoàn toàn bị cô lập và an toàn hơn	Cô lập ở mức tiến trình, có thể kém an toàn hơn

Việc sử dụng máy ảo hay Docker container sẽ phụ thuộc vào nhu cầu sử dụng của người dùng. Máy ảo rất phù hợp trong việc cách ly tài nguyên hệ thống và toàn bộ môi trường làm việc. Đây sẽ là lựa chọn tốt hơn để chạy các ứng dụng yêu cầu tất cả tài nguyên và chức năng của hệ điều hành khi bạn cần chạy nhiều ứng dụng trên server hoặc có nhiều hệ điều hành khác nhau để quản lý. Ví dụ: công ty của bạn cung cấp dịch vụ web hosting, bạn có thể sẽ sử dụng máy ảo để phân phối tài nguyên của server công ty cho từng khách hàng.

Mặt khác, triết lý của Docker là cô lập các ứng dụng riêng lẻ, không phải toàn bộ hệ thống. Một ví dụ hoàn hảo về điều này sẽ là chia nhỏ một loạt các dịch vụ ứng dụng web thành các Docker image của riêng chúng và triển khai chúng bằng Docker Container. Docker Container là lựa chọn tốt hơn khi ưu tiên lớn nhất của bạn là tối đa hóa số lượng ứng dụng đang chạy trên một số lượng server tối thiểu.

CHƯƠNG 3: XÂY DỰNG DEMO

3.1 Giới thiệu chung về Demo

Nhóm thực hiện xây dựng một ứng dụng website dạng blog với chức năng quản lý bài viết cơ bản như thêm, sửa, xóa bài viết.

Sau đó thực hiện triển khai ứng dụng bằng Docker, thực hiện build, pull và run Container để hiểu rõ cơ bản về quy trình quản lý, triển khai một ứng dụng Docker

3.2 Ngôn ngữ lập trình và công cụ sử dụng

3.2.1 JavaScript

JavaScript, theo phiên bản hiện hành, là một ngôn ngữ lập trình thông dịch được phát triển từ các ý niệm nguyên mẫu. Ngôn ngữ này được dùng rộng rãi cho các trang web (phía người dùng) cũng như phía máy chủ (với Nodejs). Nó vốn được phát triển bởi Brendan Eich tại hãng truyền thông Netscape với cái tên đầu tiên *Mocha*, rồi sau đó đổi tên thành *LiveScript*, và cuối cùng thành JavaScript. Giống Java JavaScript có cú pháp tương tự C, nhưng nó gần với Self hơn Java. **.js** là phần mở rộng thường được dùng cho tập tin mã nguồn JavaScript.

Phiên bản mới nhất của JavaScript là ECMAScript 7. ECMAScript là phiên bản chuẩn hóa của JavaScript. Trình duyệt Mozilla phiên bản 1.8 beta 1 có hỗ trợ không đầy đủ cho EX4- phần mở rộng cho JavaScript hỗ trợ làm việc với XML. được chuẩn hóa trong ECMA-357.

3.2.2 NodeJs

Nodejs là một nền tảng được xây dựng trên “V8 Javascript engine” được viết bằng c++ và Javascript. Nền tảng này được phát triển bởi Ryan Lienhart Dahl vào năm 2009. NodeJS là một nền tảng hỗ trợ xây dựng các ứng dụng web. NodeJS là một mã nguồn mở được sử dụng rộng bởi hàng ngàn lập trình viên trên toàn thế giới. NodeJS có thể chạy trên nhiều nền tảng hệ điều hành khác nhau từ Window cho tới Linux, OS X

nên đó cũng là một lợi thế. NodeJS cung cấp các thư viện phong phú ở dạng Javascript Module khác nhau giúp đơn giản hóa việc lập trình và giảm thời gian ở mức thấp nhất.

Khi nói đến NodeJS thì phải nghĩ tới vấn đề Realtime và xử lý bất đồng bộ. Realtime ở đây chính là xử lý giao tiếp từ client tới máy chủ theo thời gian thực. Máy chủ Nodejs có thể xử lý nhiều request cùng một thời điểm mặc dù không chạy đa luồng.

3.2.3 Express js

Expressjs là một web framework được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. Expressjs hỗ trợ các method HTTP và middleware tạo ra API vô cùng mạnh mẽ và dễ sử dụng.

Express js là một Framework nhỏ, nhưng linh hoạt được xây dựng trên nền tảng của Nodejs. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. Expressjs có vô số các package hỗ trợ lập trình viên nhanh hơn. Express cung cấp thêm về các tính năng (feature) để dev lập trình tốt hơn nhưng không làm giảm tốc độ của NodeJS. Các Framework nổi tiếng của NodeJS hiện nay đều sử dụng ExpressJS như một core function

3.2.3 Hệ quản trị cơ sở dữ liệu MongoDB

MongoDB là một database hướng tài liệu thuộc loại NoSQL database được viết bằng C++. MongoDB tránh cấu trúc table-based của relational database để thích ứng với các tài liệu như JSON có một schema rất linh hoạt gọi là BSON. MongoDB sử dụng lưu trữ dữ liệu dưới dạng Document JSON nên mỗi một collection sẽ các các kích cỡ và các document khác nhau. Các dữ liệu được lưu trữ trong document kiểu JSON nên truy vấn sẽ rất nhanh.

Ngoài ra, MongoDB là một cơ sở dữ liệu đa nền tảng, hoạt động trên các khái niệm Collection và Document, nó cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng

3.2.4 Visual Studio code

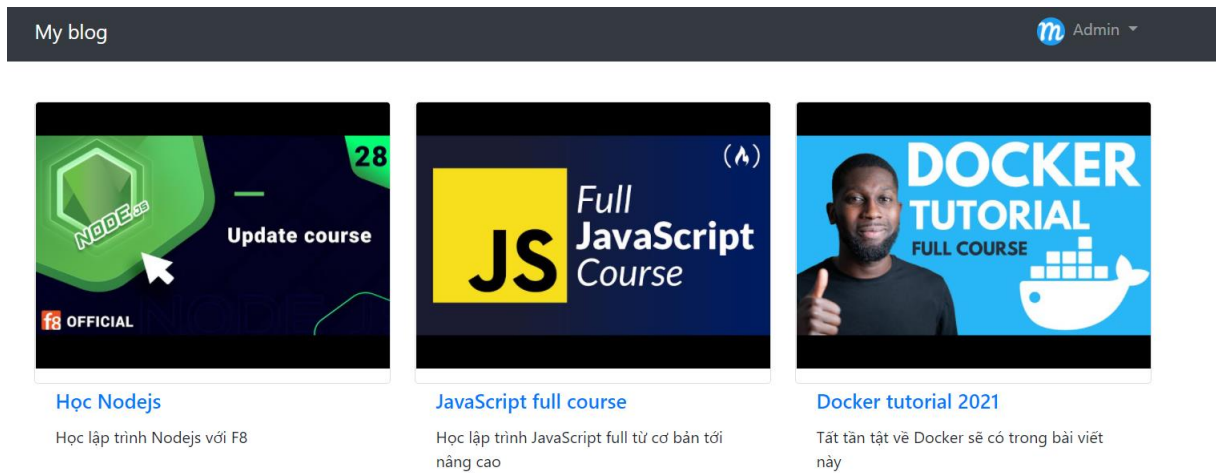
Visual Studio Code là sản phẩm của Microsoft, ra mắt vào tháng 4 năm 2015 ở hội nghị Build. Đây là một Editor có đặc điểm nổi bật là đơn giản, gọn nhẹ, dễ dàng cài đặt. Visual Studio Code có thể cài đặt được trên cả Windows, Linux và Mac OS và hỗ trợ nhiều ngôn ngữ lập trình.

Những lý do để chúng ta nên sử dụng VScode cho dự án lần này:

1. Có những setting riêng cho Workspace
2. Startup time nhanh tương đối
3. Cross-platform support
4. Tính linh hoạt cao thông qua extensions và setting
5. Giao diện thân thiện và dễ sử dụng
6. Có tích hợp Terminal cho lập trình nodejs


3.3 Hình ảnh Demo về ứng dụng

3.3.1 Giao diện trang chủ



3.3.2 Giao diện quản lý bài viết

My blog



Admin

Bài viết của tôi

STT	Tên bài viết	Mô tả ngắn	Ngày tạo			
1	Học Nodejs	Học lập trình Nodejs với F8	Tue Mar 08 2022 09:28:25 GMT+0700 (Indochina Time)	Xem	Sửa	Xóa
2	JavaScript full course	Học lập trình JavaScript full từ cơ bản tới nâng cao	Thu Mar 17 2022 08:09:52 GMT+0700 (Indochina Time)	Xem	Sửa	Xóa
3	Docker tutorial 2021	Tất tần tật về Docker sẽ có trong bài viết này	Thu Mar 17 2022 08:16:14 GMT+0700 (Indochina Time)	Xem	Sửa	Xóa

3.3.3 Giao diện thêm bài viết

My blog

 Admin ▾

TẠO BÀI VIẾT MỚI

Title

Description


Content

Video Id

Tạo bài viết

3.3.4 Giao diện chi tiết bài viết

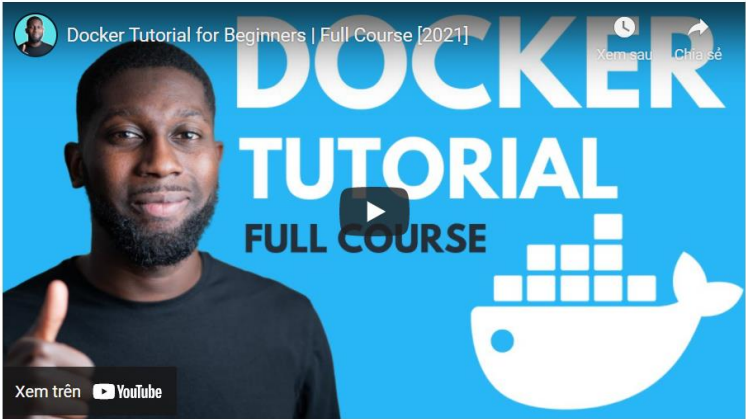
My blog


 Admin ▾

Docker tutorial 2021

Tất tần tật về Docker sẽ có trong bài viết này

Video chi tiết



Xem trên  YouTube

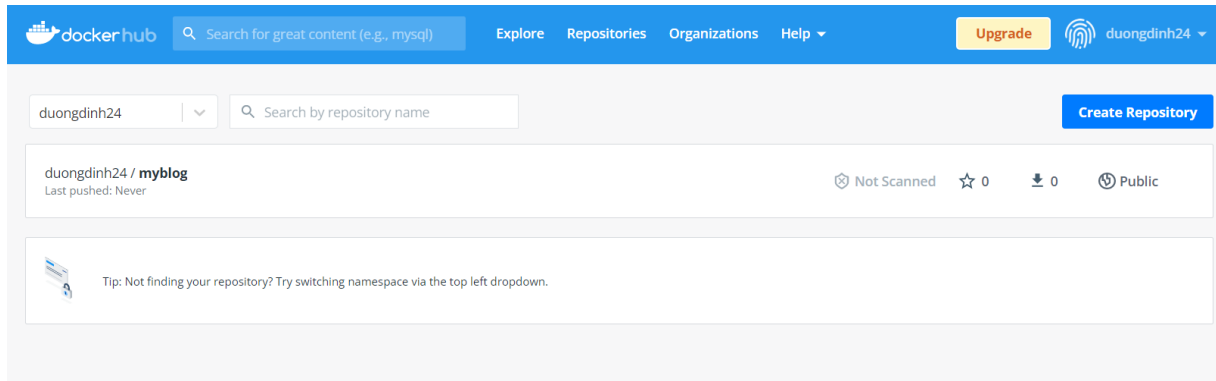
Nội dung

Học nội dung cơ bản về Docker chỉ với một video duy nhất. Mời bạn xem video

3.4 Triển khai ứng dụng với Docker

Bước 1: Tạo Docker repository trên Docker Hub

Hình ảnh repository đã tạo với tên myblog



Bước 2: Tạo Dockerfile cho ứng dụng

Nội dung bên trong Dockerfile:

- 1 FROM node:latest
- 2 WORKDIR /app
- 3 COPY package*.json ./
- 4 COPY . .
- 5 RUN npm install
- 6 EXPOSE 3000
- 7 CMD npm start

Giải thích các câu lệnh:

- 1: Nền tảng mà ứng dụng cần để sử dụng, ở đây là nodejs phiên bản mới nhất

2. Tạo thư mục làm việc cho Container xây dựng từ image này
3. Copy toàn bộ nội dung file package*.json vào thư mục làm việc
4. Copy toàn bộ mã nguồn của ứng dụng vào thư mục làm việc
5. Tự động chạy câu lệnh npm install để cài các dependencies
6. Map port của container khi chạy với port 3000 của máy tính
7. Câu lệnh command sẽ tự chạy khi khởi chạy Docker Container

Bước 3: Thực hiện build image từ Docker file trên.

Chạy lệnh: `docker build -t [user/repository]:version [path]`

Bước 4: Push Image lên docker

Chạy lệnh: `docker push [image name]:tag`

Bước 5: Pull image từ Docker hub

Chạy lệnh: `docker pull [image name]: tag`

Bước 6: Run container

Chạy lệnh: `docker run --name myblog -d -p 3000:3000 duongdinh24/myblog:latest`

Sau khi chạy xong docker

KẾT LUẬN

Sau một thời gian nhóm em bắt tay vào nghiên cứu cùng với sự giúp đỡ tận tình của thầy giáo Quách Xuân Trường, nhóm chúng em đã hoàn thành đề tài “docker for devops and developer”. Qua đây bản thân em cũng như các thành viên trong nhóm đã học hỏi được rất nhiều điều về làm việc nhóm, cách thức tiến hành, phân chia công việc và thực hiện đề tài. Đặc biệt là đã giúp cho chúng em có khả năng làm việc theo nhóm tốt hơn.

Tuy nhiên trong quá trình tìm hiểu và làm việc do thời gian có hạn cũng như kinh nghiệm của bản thân còn hạn chế nên chắc chắn trong báo cáo này không tránh khỏi thiếu sót và những chỗ xử lý vấn đề chưa được tối ưu. Chúng em rất mong nhận được những nhận xét, đánh giá từ phía các thầy cô, đặc biệt của thầy giáo hướng dẫn và giảng dạy bộ môn Vận hành bảo trì phần mềm.

TÀI LIỆU THAM KHẢO

[1]: Giáo trình vận hành và bảo trì phần mềm Đại Học Công Nghệ Thông Tin và Truyền Thông Thái Nguyên

[2]: <https://vibo.asia>

[3]: <https://docs.docker.com/>

[4]: <https://www.toptal.com/devops/getting-started-with-docker-simplifying-devops>