Bài 50 - Docker cho Deep Learning

17 Nov 2020 - phamdinhkhanh

Menu

1. Docker

Trước khi bắt đầu bài viết này, chắc các bạn thắc mắc Docker thì có liên quan gì đến deep learning? Câu trả lời tất nhiên là có:

- Việc cài đặt các deep learning framework dễ xảy ra lỗi, docker giúp bạn có thể tận dụng lại các môi trường đã được build sẵn để không cần phải cài đặt phức tạp.
- Docker giúp triển khai môi trường huấn luyện các mô hình deep learning trên nhiều máy chủ khác nhau một cách dễ dàng.
- Các ứng dụng deep learning cũng dễ dàng triển khai và chuyển giao tới khách hàng thông qua docker.

Đó chính là mục tiêu để một nông dân gà mờ về docker như tôi cất công tìm hiểu về công nghệ này và cũng là để chia sẻ tới bạn đọc ứng dụng docker trong deep learning. Bài viết ở level rất sơ đẳng và xin ghi nhận các ý kiến đóng góp từ các master docker để hoàn thiện hơn.

1.1. Khái niệm

Trước khi có docker, việc phát triển và vận hành các application trên những máy tính khác nhau gặp nhiều khó khăn vì xung đột về môi trường, hệ điều hành, phiên bản các packages,…. Chính vì thế để tạo sự thuận tiện cho việc phát triển, vận hành và chuyển giao ứng dụng, docker đã ra đời, giúp xây dựng và đóng gói môi trường và tách biệt ứng dụng của bạn khỏi infrastructure. Chính vì vậy bạn có thể triển khai ứng dụng của mình dễ dàng hơn.

Xin trích dẫn:

Top

`Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.`

Source Docker (https://docs.docker.com/get-started/overview/)

1.2. Docker trong machine learning

Hầu hết các framework deep learning phổ biến như pytorch, tensorflow, mxnet, caffe, opencv,…. đều khá khó cài đặt vì chúng thường yêu cầu nhiều package dependencies và các driver deep learning như nvidia driver, cuda (một platform hỗ trợ các tính toán song song trên GPU), cuDNN (một `GPU-accelerated library` giúp tăng tốc tính toán GPU trên các kiến trúc deep neural networks),….

Docker có thể cung cấp cho bạn các images sẵn có cho các thư viện này, bạn chỉ cần pull về và sử dụng mà không phải lo về phần cài đặt.

1.3. Một số khái niệm về docker

Trước khi tìm hiểu về docker mình sẽ diễn giải các khái niệm liên quan đến docker như `container, image` theo cách dễ hiểu nhất.

- **Container**: Hiểu đơn giản nó là một môi trường độc lập cung cấp tất cả những thứ để chúng ta có thể chạy các ứng dụng của mình trong nó. Nếu trên python bạn khá quen thuộc với virtual enviroment thì container cũng chính là một hình thái như vậy.
- **Image**: Để khởi tạo được các Docker Container thì chúng ta cần tới Docker Image. Nếu chúng ta coi Docker Image là một class thì Docker Image chính là instance hoặc object của nó.
- **Docker Hub**: Gần như github, là nơi mà bạn có thể push, clone các Docker Image về máy. Nó tạo ra một open source để chia sẻ các docker image.
- **Docker Engine**: Là một công cụ gồm các chức năng giúp bạn triển khai docker.

1.4. Ưu điểm của docker so với virtual machine

Trước khi docker ra đời, để triển khai một ứng dụng chúng ta thường build một máy ảo (virtual machine), trên một server có thể có nhiều máy ảo với các OS khác nhau. Tuy nhiên khi không chạy ứng dụng thì chúng ta vẫn phải tiêu tốn một nguồn tài nguyên Ổ cứng, RAM, CPU cho các máy ảo này. Thêm vào đó là việc tắt mở các Virtual Machine khá lâu nên rất tốn kém chi phí thời gian. Vì vậy đây không phải là phương pháp tối ưu để triển khai ứng dụng.

Docker ra đời, giải quyết được vấn đề nhức nhối trên khi nó có thể khởi tạo được nhiều môi trường container sử dụng chung một OS kernel của host OS. Những môi trường này có thể chia sẻ tài nguyên lẫn nhau chứ không bị fix cứng như máy ảo. Và đặc biệt là khởi chạy một docker image chỉ vài giây. Công nghệ này được gọi là `containerlization` .

Top

Hình 1: Kiến trúc Docker trên một server. Mỗi container sẽ phụ trách một application, những container này có thể **chia sẻ** tài nguyên lẫn nhau, cùng chung một OS kernel và start runtime.

2. Cài đặt docker

Docker có nhiều phiên bản khác nhau bao gồm cả ubuntu, window và mac. Tuy nhiên phần này mình chỉ hướng dẫn trên ubuntu vì nếu bạn đang làm việc với deep learning thì mình mặc định là bạn sử dụng ubuntu.

Phần này đã có hướng dẫn chi tiết tại install docker engine (https://docs.docker.com/engine/install/ubuntu/). Mình xin tóm tắt lại, bước này sẽ gồm 2 phần chuẩn bị repository và cài đặt Docker Engine.

Trên terminal các bạn gõ các lệnh sau

Phần 1: Chuẩn bị repository

Step 1: Update apt package index, đây là một phần bắt buộc trước khi bạn cài các package trên ubuntu để tránh các lỗi phát sinh. Sau đó cài đặt các package làm việc với https.

```
$ sudo apt-get update

$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common
```

Step 2: Add GPG key cho docker. GPG key là một key giúp bạn giải mã và download docker.

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-k
```

Step 3: cài đặt bản stable của docker.

```
$ sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

Phần 2: Cài đặt Docker Engine

```
 $ sudo apt-get update
 $ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Sau khi cài đặt xong bạn run image `hello-world` để kiểm tra xem có thành công hay không :

```
$ sudo docker run hello-world
```

Nếu kết quả trả về là `Hello from docker` thì quá trình cài đặt đã thành công.

Top

3. Các lệnh cơ bản trong docker

Docker có rất nhiều lệnh giúp build, pull, run, compose images. Chúng ta sẽ không thể sử dụng hết các lệnh này nên mình sẽ giới thiệu một số lệnh chính thường dùng. Muốn tìm hiểu chi tiết hơn thì bạn có thể tham khảo tại Docker CLI (https://docs.docker.com/engine/reference/commandline/cli/).

- Run một docker image

```
$ docker run --name <container_name> -it --rm -p 8888:8888 -v $PWD:/tm
```

–name là tên của container sau khi được khởi tạo. Nếu không thiết lập tên thì docker sẽ sinh ra một tên mặc định cho nó.

-it là lựa chọn bắt buộc khi bạn run docker với interactive process (chẳng hạn shell) để thiết lập tty cho container process.

–rm sẽ xóa container sau khi exit docker.

-p {host_port}:{container_port} là tùy chọn mapping port giữa host và container.

-v {host_directory}:{container_directory} là lệnh mount volumn giữa host với container. Trong lệnh trên chúng ta đã mount `current directory` trên host với thư mục `/tmp` của container. Sau khi mount thì dữ liệu giữa hai thư mục sẽ như nhau.

-w thay đổi thư mục làm việc của container về `/tmp` .

- Pull docker image từ docker hub

```
$ docker pull <image_name:tag>
```

Trong đó `image_name` là tên của docker image trên docker hub và tag là nhãn đánh dấu phiên bản của docker image. Mặc định không điền nhãn thì sẽ download bản `lasted` .

Bạn có thể tìm được bản docker mà mình cần tải tại docker hub (https://hub.docker.com)

- Liệt kê danh sách các images

```
$ docker images
```

Kết quả trả về:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
tensorflow/tensorflow	nightly-jupyter	c134af74bcb0	10 days ago	1.75GB
nvidia/cuda	latest	539690cdfcd6	10 days ago	4.77GB
pytorch/pytorch	latest	f8a1d10ae3d7	2 weeks ago	4.65GB
stereolabs/zed	3.3-runtime-cuda11.1-ubuntu18.04	935f383591e1	2 weeks ago	3.23GB
tensorflow/tensorflow	latest-jupyter	3c3d02b0ce58	6 weeks ago	1.62GB

Cột `IMAGE ID` chính là index của image và là duy nhất. Tiếp theo để xóa một image thì chúng ta sẽ dựa trên `image_id` .

- Liệt kê danh sách các container đang chạy

```
$ docker container ls
```

Trên máy mình hiện đang có một container đang chạy là `hikhanh` (`hikhanh` là container name được đặt từ –name ở lệnh docker run).

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
c91214e9b1ab	tensorflow/tensorflow	"/bin/bash"	2 minutes ago	Up 2 minutes	0.0.0.0:8888->8888/tcp	hikhanh

- Xóa một image

```
$ docker rmi <image_id>
# or
$ docker image rm <image_id>
```

- Xóa một container

Bình thường nếu chúng ta thêm argument `--rm` vào lệnh `docker run` thì container sẽ tự động remove sau khi exit. Nếu không thêm lựa chọn này thì container sẽ vẫn tồn tại và chiếm dụng tài nguyên của máy. Khi đó có thể xóa chúng bằng lệnh.

```
$ docker container rm <container_name>
```

Trong đó container_name chính là cột NAMES mà chúng ta thấy ở lệnh `docker container ls` .

Document cho các lệnh docker

Để tìm hiểu sâu hơn các lệnh của docker bạn có thể tham khảo các links sau:

- docker image commandline (https://docs.docker.com/engine/reference/commandline/image/)
- docker container commandline (https://docs.docker.com/engine/reference/commandline/container/)

4. Build docker

4.1. Build một image

Để build một docker image chúng ta sử dụng lệnh `docker build` . Lệnh này sẽ xây dựng image từ Dockerfile và context. context ở đây được hiểu là tập hợp các tệp nằm trong PATH hoặc URL được chỉ định.

Thế Dockerfile là gì? Dockerfile hiểu đơn giản là một file qui định các lệnh cần thiết để docker engine build một image như `FROM, COPY, RUN, EXPOSE` .

Có rất nhiều cách khác nhau để khởi tạo một docker image như sử dụng DockerFile; thông qua git repository; context đã được đóng gói sẵn trong các file `tar.gz, xz, bzip2 gzip` hoặc folder.

```
$ docker build -t <image_name:tag> .
```

Trong đó -t sẽ tạo ra một `<image_name:tag>` .

Dấu `.` là đại diện cho thư mục hiện hành nơi chứa `Dockerfiles` và các files context để build image.

Để hiểu rõ hơn về lệnh build chúng ta cùng thực hành một ví dụ đơn giản để build một image in ra dòng `hello world` từ file `hello.txt` .

```
echo "hello world" > hello.txt
```

Bạn tạo Dockerfiles với nội dung như sau:

Top

```
1     from busybox
2     COPY hello.txt /
3     RUN cat hello.txt
```

Tại cùng thư mục bạn gõ lệnh.

```
1     $ docker build -t hello:v1 .
```

Nội dung trả về

```
Sending build context to Docker daemon  6.144kB
Step 1/3 : FROM busybox
 ---> f0b02e9d092d
Step 2/3 : COPY hello.txt /
 ---> d5cdc87fbfbb
Step 3/3 : RUN cat /hello.txt
 ---> Running in e787f496b3ab
hello world
Removing intermediate container e787f496b3ab
 ---> 387dc4a3cf1a
Successfully built 387dc4a3cf1a
Successfully tagged hello:v1
```

Như vậy bạn thấy rằng một `<image:tag>` đã được tạo ra là `hello:v1` và in ra dòng `hello world` tại step thứ 3.

4.2. Build image từ các context

Giả sử các files context của chúng ta là phân tán. Chúng ta có thể build image thông qua khai báo từ lệnh `-f` .

Bạn move Dockerfile vào thư mục `dockerfiles` và file `hello.txt` vào thư mục `context`

```
1     mkdir -p dockerfiles context
2     mv Dockerfiles dockerfiles && mv hello.txt context
```

Tiếp theo chúng ta sẽ build một image là `hello:v2` từ các files context phân tán

```
1     $ docker build --no-cache -t hello:v2 -f dockerfiles/Dockerfile contex
```

```
Sending build context to Docker daemon  2.607kB
Step 1/3 : FROM busybox
 ---> f0b02e9d092d
Step 2/3 : COPY hello.txt /
 ---> 213eb718f9cd
Step 3/3 : RUN cat hello.txt
 ---> Running in 97ee63840864
hello world
Removing intermediate container 97ee63840864
 ---> 8ebc104bb299
Successfully built 8ebc104bb299
Successfully tagged hello:v2
```

Top

4.3. Các lệnh trong Dockerfile

Tiếp theo chúng ta sẽ tìm hiểu các lệnh trong Dockerfile

- FROM: Lệnh này thường được sử dụng đầu Dockerfile để khởi tạo một build stage từ base image. Base image được lấy từ Dockerhub - Repository (https://docs.docker.com/docker-hub/repos/) thường là những image có kích thước rất nhẹ và phù hợp với mục đích mà ta cần build.
- RUN: Sẽ thực thi các lệnh terminal trong quá trình build image. Có thể có nhiều lệnh RUN liên tiếp nhau.

```
1     RUN apt-get update
2     RUN apt-get install -y package-bar package-baz package-foo
3     RUN rm -rf /var/lib/apt/lists/*
```

Một lệnh RUN dài có thể xuống dòng để dễ đọc, dễ hiểu hơn bằng ký hiệu blash (`\`).

```
1     RUN apt-get update && apt-get install -y \
2         package-bar \
3         package-baz \
4         package-foo  \
5         && rm -rf /var/lib/apt/lists/*
```

Trong quá trình build một image thì mỗi lệnh `RUN` trong Dockerfile sẽ build thành một layer. Các layer sẽ giúp caching quá trình build và khi re-build sẽ nhanh hơn vì chỉ phải build lại bắt đầu từ dòng lệnh bị thay đổi và tận dụng các phần trước đó đã được caching.

Khi tách một lệnh `RUN` ghép thành nhiều lệnh `RUN` đơn, chúng ta sẽ có nhiều layer caching hơn và quá trình build sẽ nhanh hơn. Ví dụ trong 2 cách chạy lệnh `RUN` để thực hiện cùng một tác vụ như trên thì với cách chạy thứ 2 chúng ta sẽ phải chạy lại toàn bộ lệnh mỗi khi có một trong ba lệnh con thay đổi. Nhưng với cách chạy đầu tiên thì các lệnh sau thay đổi sẽ chỉ phải build lại từ dòng lệnh đó trở đi vì các dòng lệnh trước đã được lấy lại từ caching.

- LABEL: Cung cấp thông tin về metadata cho image như tác giả, email, công ty,…
- EXPOSE: Thiết lập port để access container sau khi nó khởi chạy.
- COPY: Cú pháp chung của lệnh là này là `COPY <src> <dest>` . Lệnh này nhằm copy thư mục từ host (là máy mà chúng ta cài docker image) vào container.Ví dụ trên máy chúng ta có thư mục `host_dir` . Chúng ta muốn copy vào container tại địa chỉ tuyệt đối `/app/` .

Note: Nếu chúng ta lấy đường dẫn của `<dest>` là `/folder/` thì đây là đường dẫn tuyệt đối xuất phát từ root. Còn nếu chúng ta lấy đường dẫn của `<dest>` là `folder/` thì nó được xem như đường dẫn tương đối bắt đầu từ `<WORK_DIR>/folder/` . Đây là một kiến thức cơ bản nhưng lại là một trong những nguyên nhân gây lỗi khi build.

```
1     COPY /host_dir /app/
```

- ADD: ADD cũng làm nhiệm vụ tương tự như COPY nhưng nó hỗ trợ thêm 2 tính năng nữa là copy từ một link URL trực tiếp vào container và thứ hai là bạn có thể extract một tar file trực tiếp vào container.
- CMD: Là câu lệnh được thực thi mặc định trong docker image. CMD sẽ không thực thi trong quá trình build image. Một file sẽ chỉ cần một lệnh CMD duy nhất. Cấu trúc của CMD là `CMD ["executable", "param1", "param2"…]` hoặc `CMD ["param1", "param2"…]` .

top

Chẳng hạn chúng ta muốn khi run docker thì sẽ in ra địa chỉ `$HOME` . Chúng ta có thể thêm dòng lệnh:

```
1     CMD ["echo", "$HOME"]
```

Ở đây `echo` chính là thành phần executable và `$HOME` là param được truyền vào.

- ENTRYPOINT: Cung cấp các lệnh mặc định cùng tham số khi thực thi container. Lệnh CMD và ENTRYPOINT sẽ có chức năng giống nhau, ngoại trừ ENTRYPOINT có thể lặp lại nhiều lần trong một Dockerfile trong khi CMD là duy nhất.
- ENV: Thiết lập các biến environment.
- VOLUME: Mount folder từ máy host tới container.

Ví dụ bạn muốn mount một folder `/main` từ host lên `/container` thì bạn sẽ thực hiện mount như sau:

```
1     VOLUME /main
```

- WORKDIR: thay đổi thư mục làm việc hiện hành cho các lệnh thực thi như `RUN, CMD, ENTRYPOINT, COPY, ADD` ở phía sau nó. Lệnh này cũng giống như cd trong linux.
- ONBUILD: Tạo một trigger như là một điểm chờ cho việc build image. Các lệnh phía sau lệnh ONBUILD sẽ không được thực thi cho đến khi image đó được sử dụng làm base image cho việc build một image khác thì các lệnh sau ONBUILD sẽ được được thực thi theo tuần tự.

Lệnh ONBUILD này sẽ cho phép chúng ta sử dụng lại template image chung cho nhiều images khác nhau. Bạn chỉ cần wrap lại template đó dưới dạng một builder image và các image khác sẽ chờ cho đến khi template image được build thì mới build tiếp.

Mình xin lấy ví dụ:

Đây là image template của mình, sẽ được sử dụng lại ở rất nhiều image khác có Dockerfile như sau:

```
1     # image template
2     FROM ubuntu
3     ONBUILD RUN mkdir /myvol
4     ONBUILD RUN echo "hello world" > /myvol/greeting
5     ONBUILD VOLUME /myvol
6
```

build template image với tên gọi là `test`

```
1     $ docker build -t test .
```

Top

```
Sending build context to Docker daemon  4.608kB
Step 1/4 : FROM ubuntu
 ---> d70eaf7277ea
Step 2/4 : ONBUILD RUN mkdir /myvol
 ---> Using cache
 ---> 37e29adbb1f0
Step 3/4 : ONBUILD RUN echo "hello world" > /myvol/greeting
 ---> Using cache
 ---> 2f4bfaaa67ce
Step 4/4 : ONBUILD VOLUME /myvol
 ---> Using cache
 ---> 67b8990814be
Successfully built 67b8990814be
Successfully tagged test:latest
```

Ta thấy 3 lệnh được thực hiện với ONBUILD là RUN và VOLUME sẽ không được thực thi.

Tiếp theo một image khác sẽ được build tiếp từ image đầu tiên.

```
1     # Runtime image
2     FROM test
3     RUN echo "FROM test run after!"
```

```
Step 1/2 : FROM test
# Executing 3 build triggers
 ---> Running in 32182d3b47ee
Removing intermediate container 32182d3b47ee
 ---> Running in a4e97e5609ba
Removing intermediate container a4e97e5609ba
 ---> Running in f4b397db1f78
Removing intermediate container f4b397db1f78
 ---> 06f6ef8d4fb5
Step 2/2 : RUN echo "FROM test run after!"
 ---> Running in bf3a8ae03c3a
FROM test run after!
Removing intermediate container bf3a8ae03c3a
 ---> 202a00076d30
Successfully built 202a00076d30
Successfully tagged runtime:latest
```

Kết quả cho thấy 3 lệnh ONBUILD trước đó đã được thực thi, sau đó mới đến lệnh cuối cùng là `RUN echo "FROM test run after!"`.

- ENV: Xác định một biến môi trường cho docker image. Giá trị này sẽ tồn tại trong toàn bộ các build stage.
- ARG: Định nghĩa một argument được sử dụng trong quá trình `docker build`. Nó tương tự như argument parser trong python.

```
1     from alpine
2     ARG USER
3     RUN echo $USER
```

Khi chạy docker build

```
1     $ docker build --build-arg USER=phamdinhkhanh .
```

Top

ARG sẽ không tồn tại trong nhiều build stage, do đó ở các stage tiếp theo muốn sử dụng chúng ta phải định nghĩa lại chúng.

```
from busybox
ARG SETTINGS
RUN ./run/setup $SETTINGS

# Re-define in next stage
FROM busybox
ARG SETTINGS
RUN ./run/other $SETTINGS
```

5. Build Docker cho deep learning

5.1. My first deep learning docker

Chúng ta có thể cài đặt được một môi trường chứa sẵn các framework deep learning phổ biến như `tensorflow, pytorch, mxnet` để huấn luyện và triển khai các mô hình deep learning. Để tạo một môi trường như vậy sẽ khá khó vì các packages này rất dễ bị xung đột về depencencies. Sau một quá trình mày mò thì mình đã tìm ra được giải pháp để build thư viện này nhờ vào việc sử dụng một base image chuẩn là `anaconda3` . Bạn có thể tham khảo Dockerfile bên dưới:

Requirement:

Máy tính của bạn phải có sẵn :

- Ubuntu 18.04
- CUDA v10.1
- cuDNN v7

Dockerfile

Top

```
from okwrtdsh/anaconda3:10.0-cudnn7

RUN apt-get update -qq \
 && apt-get install --no-install-recommends -y \
        graphviz \
 && apt-get clean \
 && rm -rf /var/lib/apt/lists/*

# opencv install
RUN conda install -c anaconda opencv

# pytorch install
# RUN conda install -c pytorch magma-cuda101
RUN pip install torch==1.7.0+cu101 torchvision==0.8.1+cu101 torchaudi

# tensorflow install
# RUN conda install -c anaconda tensorflow
RUN pip install wrapt --upgrade --ignore-installed
RUN pip install tensorflow

# mxnet install
RUN pip --no-cache-dir install \
        mxnet-cu101 \
        mxnet-cu101mkl \
        graphviz \
        pydot_ng \
        msgpack

RUN echo "completed!"
CMD ["python3"]
```

Ở trên là bản build cho CUDA version 10.1. Bạn đọc cũng có thể thay thế bản build chuẩn cho môi trường của mình bằng cách thay thế `10.1` trong Dockerfile bằng version tương ứng.

Tiếp theo dựng image với tên `khanh-dl` .

```
$ docker build -t khanh-dl .
```

Khởi tạo container:

```
$ docker run --name dl_container --rm -it khanh-dl
```

5.2. Kết nối jupyter notebook

Bạn cũng có thể kết nối với jupyter notebook

```
$ docker run --name dl_container --rm -it -p 8888:8888 khanh-dl bash
```

lệnh `bash` ở cuối sẽ đưa về terminal của container. Tiếp theo chúng ta sẽ khởi chạy jupyter notebook.

Top

```
$ jupyter notebook
```

Có thể bạn sẽ không mount được port 8888 do bị chiếm dụng. Lúc đó có thể sử dụng lệnh mount all port từ host sang container.

```
$docker run --name dl_container --rm -it --net="host" khanh-dl bash
```

Note: Thanks anh Nghĩa Cao đã contribute ý tưởng này.

5.3. Push docker của bạn lên docker hub

Docker hub là một nơi tuyệt vời để chia sẻ docker image. Để push docker bạn cần thực hiện các bước sau:

- Tạo repository trên docker hub (https://hub.docker.com/repositories).
- Login docker hub.

```
$ docker login
```

- Docker tag: Tag docker image trước khi push lên

```
$ docker tag khanh-dl:latest <username>/<repository>:<tag>
```

- Push docker image vừa tag

```
$ docker push <username>/<repository>:<tag>
```

- Bạn có thể pull phiên bản docker của mình ở trên tại:

```
$ docker pull khanhpd2/khanh-dl:0.1
```

Note: Thanks Quang BĐ đã giúp anh push first docker đầu tiên.

Hoặc nếu bạn theo trường phái mì ăn liền thì có thể tham khảo một bản docker cho all deep learning framework khá nhẹ là: ufoym-deepo (https://github.com/ufoym/deepo)

6. Docker compose

Giả sử bạn có rất nhiều các container cần được vận hành cùng nhau. Cần có một cơ chế giúp cho việc quản lý chúng tốt hơn. Docker compose (mình sẽ gọi là Compose) là một tool mà bạn có thể định nghĩa và chạy multi-container Docker. Việc cần làm là bạn định nghĩa tonaf bộ các services cần thiết thông qua một YAML file.

Compose sẽ hoạt động trên toàn bộ các môi trường khác nhau như: production, staging, development, testing, ….

Khởi tạo một Compose sẽ trải qua ba bước cơ bản:

- Định nghĩa môi trường với Dockerfile để có thể dựng nó ở mọi nơi.
- Xác định các services được tạo thành trong docker-compose.yml để bạn có thể chạy chúng trong những môi trường tách biệt.

Top

- Khởi chạy Compose để vận hành toàn bộ các services của bạn thông qua `docker-compose up`.

Một `docker-compose.yml` sẽ có dạng như sau:

```
version: "3.8"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Trong đó định nghĩa các services. Đối với mỗi service thì có cách build như thế nào ? địa chỉ port, volumes, links,…. Để hiểu hơn các thành phần này thì bạn có thể tham khảo tại hướng dẫn compose-file (https://docs.docker.com/compose/compose-file/).

Compose has commands for managing the whole lifecycle of your application:

Start, stop, and rebuild services View the status of running services Stream the log output of running services Run a one-off command on a service

Compose sẽ có những command giúp quản lý toàn bộ lifecycle application của bạn:

- Start, stop, rebuild các services.
- View status của quá trình running services.
- Stream log output của các running services.

Ngoài ra có những tính năng giúp docker compose hiệu quả đó là:

- Nhiều môi trường tách biệt trên một single host: Đặc điểm này có gì ưu việt ? Bạn có thể tạo ra nhiều môi trường tách biệt từ single image gốc mà ổn định hơn cho từng feature branch chẳng hạn.
- Bảo toàn data khi các containers được khởi tạo: Khi khởi chạy `docker-compose up` thì nó sẽ copy lại nội dung từ các containers đã chạy trước đó nên bảo toàn được dữ liệu.
- Chỉ khởi tạo lại các containers khi chúng thay đổi: Tương tự như cơ chế caching, nếu container thay đổi thì mới khởi tạo lại. Do đó tốc độ compose nhanh hơn.

7. Nguồn tham khảo

https://github.com/floydhub/dl-docker

https://github.com/okwrtdsh/anaconda3

https://github.com/yhavinga/jupyter-tensorflow-pytorch-gpu

https://towardsdatascience.com/build-a-docker-container-with-your-machine-learning-model-3cf906f5e07e

Top

https://kipalog.com/posts/Docker-image-in-production—cau-chuyen-1GB-hay-100MB

https://viblo.asia/p/docker-tong-hop-mot-vai-kien-thuc-co-ban-yMnKM9XDK7P

https://docs.docker.com/engine/reference/run/

https://viblo.asia/p/docker-chua-biet-gi-den-biet-dung-phan-2-dockerfile-RQqKLzeOl7z

https://kipalog.com/posts/Dev-hien-dai-phan-1–Setup-moi-truong-dev-voi-docker

https://github.com/ufoym/deepo

Top