



Docker introduction 1

📅 Date	@August 13, 2022
🔖 Tags	In progress
👤 対象	docker 初心者



Context :

- 开门见山：浅谈一下笔者对Docker的看法并简单的介绍一些docker
- 浅尝辄止：简单的介绍一下如何学习docker，并通过跑一个实例，简单的介绍一下docker的特点
- 小试牛刀：通过简单的写一个Flaskapp，介绍一些开发中的小技巧

▼ 开门见山

为什么要学习，了解Docker

- dev(開発)およびops(运维) (共通)
- devops)云服务 (クラウドサービス)
 - [Microsoft Azure](#)
 - Amazon AWS
 - EC2
 - 通过构建docker环境，RUN docker image ，创建container
 - ECS
 - 通过在task中直接启动docker镜像文件 (image) ，构建服务
 - [Google GCP](#)
- dev(開発)
 - 本地环境 不会变成垃圾场
 - eg :
 - Java开发为例
 - PC安装JDK和相关package，maven。。

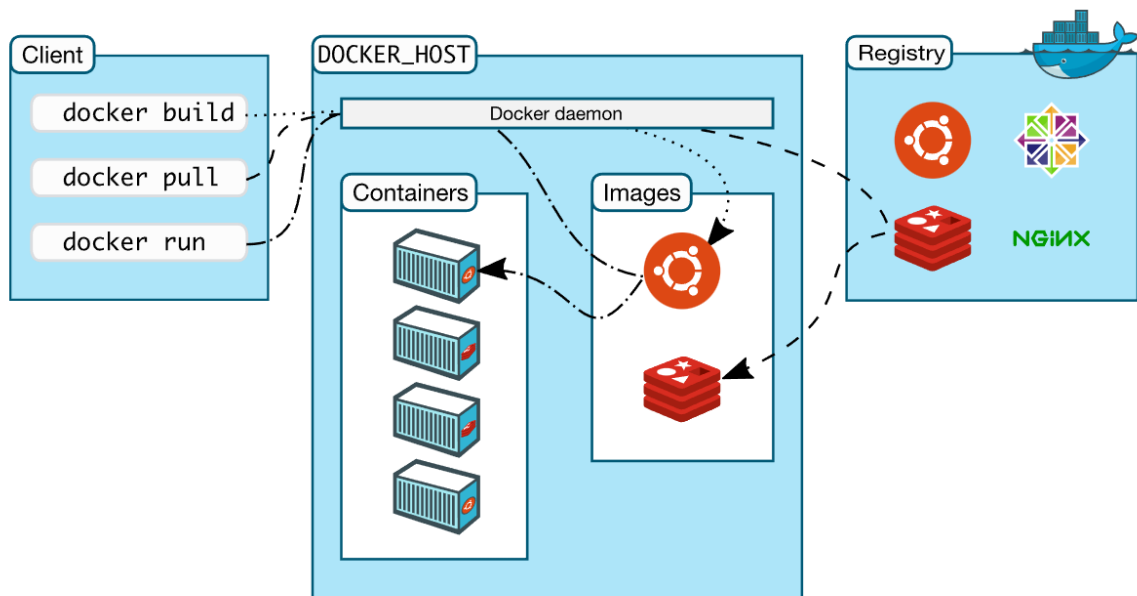
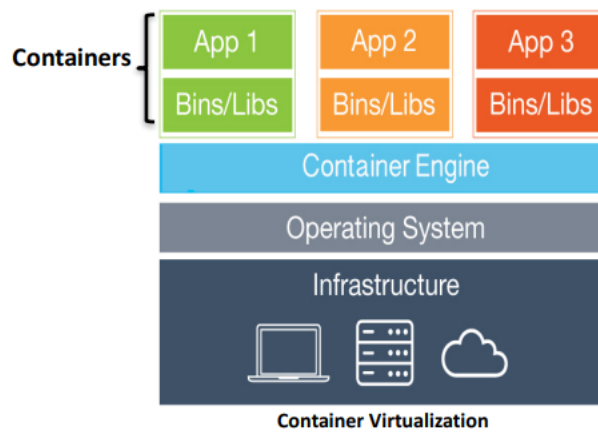
- 有数据库的 需要搞个数据库 mysql, orcale等等
- IDE配置, eclipse
- etc.....
- 突然电脑挂了。。。😓
- 开发环境可以随时重构,
- スクラム アジャイル開発 or アジャイル開発 or team開発方便资源共享, 提高开发速度
- ソフトウェアアップデート、脆弱性対応
- 本番環境と同じ環境を構築することが可能
- 可以在 (dockerhub) 开源资源里面找到, 你所需要的开发环境
 - Linux
 - python
 - tomcat, apache, java
 - etc....
- ops(運用)
 - 方便部署
 - 开发环境, 生产环境, 测试环境都**完全相同**
 - (SRE) 可以通过K8进行跨平台部署, 监视, 管理

为什么要用Docker

- 柔軟性
 - 它可以在混合环境中搭载
 - 开发环境部署 (windows, ios, linux, ubuntu)
 - テスト環境
 - 本番環境
- 軽量化
 - 可以自由选择配置文件 (Linux, Debain) , 只下载必要的packages。
 - 可以通过极大的精简了像文件 (image) 的大小, 来节省服务器空间。
- 微服务(マイクロサービス)优化 (K8+docker)
 - 控制层分流
 - 可以对从web端口, 移动端, 小程序等等请求进行分流, 引入对应的container
 - 服务层分流
 - 可以根据用户服务, 商品服务, 订单服务等等不同的服务类型, 引入相应container

Docker 能干什么, 他是怎么工作的

- 通过下载docker软件及启动相应(image)镜像文件来实现快速的搭建虚拟服务器环境
- 对(image)镜像文件进行版本管理



▼ 浅尝辄止

- 如何学习docker
 - 英文官方 <https://docs.docker.com/get-started/overview/>
 - 日文版チュートリアル <https://docs.docker.jp/get-started/index.html>
- RUN docker to say "Hello world"

```
docker run busybox:1.24 echo "hello world"

docker run repository:tag command [argument]
```

▼ 実行結果 1

```
Unable to find image 'busybox:1.24' locally
1.24: Pulling from library/busybox
Image docker.io/library/busybox:1.24 uses outdated schema1 manifest format. Please upgrade to a schema2 image for better future
```

```
385e281390cc: Pull complete
a3ed95cae02: Pull complete
Digest: sha256:8ea3273d79b47a8b6d018be398c17590a4b5ec604515f416c5b797db9dde3ad8
Status: Downloaded newer image for busybox:1.24
hello world
PS C:\Users\user\Desktop\python_program>
```

▼ 実行結果 2

当docker在本地文件中没有找到，所需要启动的image文件的时候，就回去dockerhub里面下载image

```
PS C:\Users\user\Desktop\python_program> docker run busybox:1.24 echo "hello world"
hello world
PS C:\Users\user\Desktop\python_program>
```

◦ 练习

- `docker run busybox:1.24 ls /`

- 到【xxx】里面看看

```
docker run -it busybox:1.24
```

▼ log

```
PS C:\Users\user\Desktop\python_program> docker run -it busybox:1.24
/ # ls
bin  dev  etc  home  proc  root  sys  tmp  usr  var
/ # touch a.txt
/ # ls
a.txt bin  dev  etc  home  proc  root  sys  tmp  usr  var
/ # exit
PS C:\Users\user\Desktop\python_program> docker run -it busybox:1.24
/ # ls
bin  dev  etc  home  proc  root  sys  tmp  usr  var
/ #
```

▼ 小试牛刀 (Hands On)

github source :

- Dockerfile

```
FROM python:3.8-slim-buster

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip3 install -r requirements.txt

COPY . .

CMD [ "python3", "-m", "flask", "run", "--host=0.0.0.0"]
```

- app.py

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, Docker!'
```

- command

```
# 生成image
docker build --tag flask-image .
# version管理
docker tag python-docker:latest python-docker:v1.0.0
# 删除指定 image
docker rmi python-docker:v1.0.0
# 查看image
docker images

# 启动镜像文件
docker run flask-image
~~~~
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://172.17.0.2:5000
~~~~

# 浏览器查看 or curl查看
http://localhost:5000/
curl localhost:5000

# 停止运行 container
docker stop xxxx

# 添加端口 再次启动
docker run -p 8000:5000 flask-image

# 浏览器查看 or curl查看
http://localhost:5000/
curl localhost:5000
```

- docker-compose.yml

```
version: '3.8' <-- version不同文件的写法稍有不同 一般用 V3

services:
  python-flask:
    build:
      context: .
      dockerfile: Dockerfile
    image: flask-image
    container_name: flask-app
    ports:
      - 8000:5000
    volumes:
      - ./:/app
```

- command

```
# 如果有image则导入已有image并启动container, 若image不存在则构建image并启动container
docker-compose up -d

# 重新构造image并启动container
docker-compose up --build -d
```

- 小记：

- docker为运维方式提供了快速构建，重构系统的新的途径。并且在开发中已经广为应用。我们应当学会，并掌握它。