docker容器化部署

2020年11月29日 21:12

序

在之前我完成了nginx+uwsgi+supervior对flask项目的部署。在这之后，了解到了利用docker对项目进行容器化部署。

真正开始利用docker部署项目之前，我希望能先搞清楚docker给目前的项目带来了什么。之前利用nginx+uwsgi作为web服务器，flask作为web应用，最后利用supervior来保护整个服务进程以实现进程的可持续。这是一个比较清晰而且逻辑上已经比较完整的流程。那么为什么还需要docker来进行进一步的外部封装呢?

场景

目前在我看来，docker的出现主要解放了后端运维人员的一些繁杂工作。原本开发人员提供给运维人员的待上线内容往往只是一个应用，以这次项目为例，开发人员向运维人员提交的就仅仅是一个flask框架web应用。运维人员的工作就是将开发人员提供的web应用部署到服务器上，也就是项目的正式上线（上次做的工作就是比较朴素的上线手段）。这二者之间存在一个障碍，就是环境的问题。

产生的问题

开发人员在开发过程中应用运行在本地的环境，称为开发环境，其中往往涵盖了错综复杂的依赖关系。而运维人员在上线应用时需要将开发人员在本地搭建的环境完整地再搭建一遍，而且不用的机器不同的环境基础设施往往会让这个问题变得更加麻烦。这个过程无疑是耗时耗力的。而docker主要解决的也就是上述的环境问题。

解决问题的方案

docker的思想就体现在它的图标上。docker就像将邮轮上的各类货物封装成一个集装箱一样地去处理我们服务器上的各类应用，而封装货物的载体也就是‘集装箱’在我目前的理解下就是虚拟机镜像。将应用连同环境打包进镜像，而在上线时是将集装箱（镜像）连同里面装的货物（应用）一同装载进邮轮（服务器）。由此免去了应用与服务器的环境配置，以实现快速部署。

当然这其中还有一层关系，镜像（image）与容器（container）之间的关系。简单来说镜像是一个模板（没有任何动态数据），而容器是镜像的一个实例。真正运行的是容器而不是镜像。

详见：https://blog.csdn.net/taifei/article/details/82697312

docker图标

下面开始用docker对之前部署的方案（nginx+uwsgi+supervisor）进行进一步的封装。

值得注意的是，docker拥有自己的类似于git的运作方式，我们可以直接把镜像存至云端然后在服务端对其直接进行拉取即可，我也操作过这种方式，其中有很多坑类似于更换各类源的方式，系统版本适配，Hyper-V应用兼容等问题。但是这次部署最终我采用的是通过配置docker-compose 以及 Dockerfile 文件来在服务端自动生成镜像容器。

概述

①安装docker -> ②配置docker-compose -> ③配置Dockerfile -> ④修订uwsgi.ini -> ⑤修订nginx.conf -> ⑥构造镜像 -> ⑦生成容器并启动

Docker-compose

连接Docker容器的配置文件，定义了体系内的docker容器参数，以及他们连接的关系。

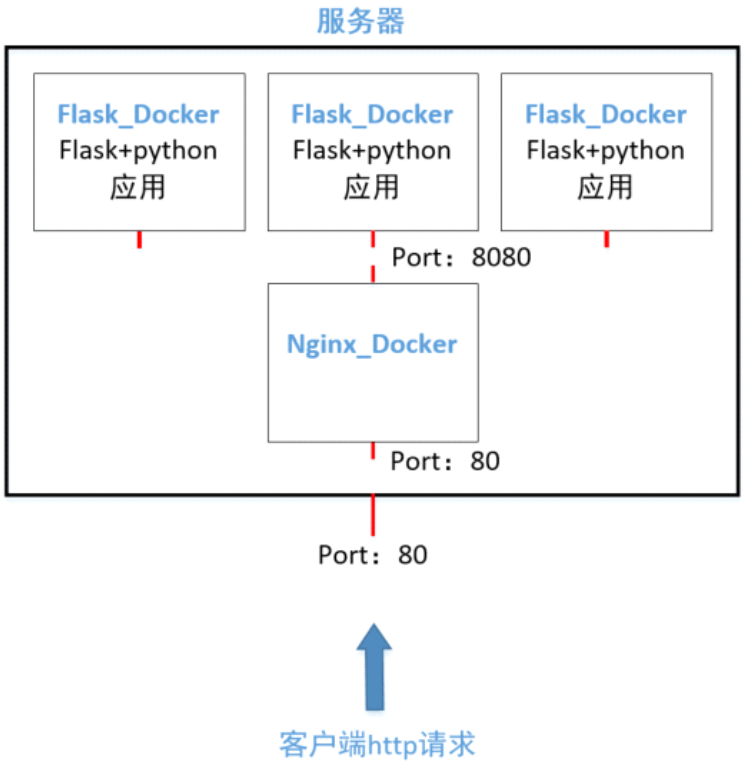
Dockerfile

配置Docker镜像信息，定义了镜像如何生成，由此文件生成Docker镜像。

Uwsgi.ini & nginx.conf

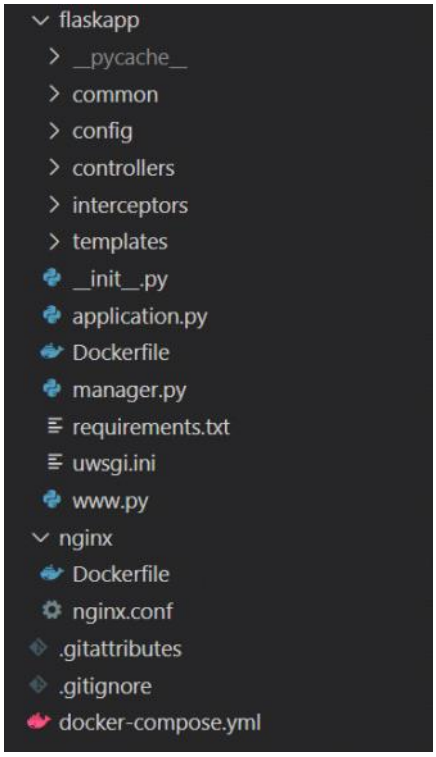
uwsgi与nginx web server的配置文件，二者相关联，具体关系详见【nginx+uwsgi+supervisor在ubuntu部署flask项目】博客。

系统结构图



图源:https://www.jianshu.com/p/d6c10802b70b

文件结构图



Github:https://github.com/vanot313/flask.demo

flaskapp

应用根目录，其中包含完整flask应用，Dockerfile配置文件以及uwsgi配置文件。

Nginx

Nginx根目录，其中包含Dockerfile配置文件以及uwsgi配置文件。

Docker-compose.yml

Docker-compose配置文件。

建议结合上面结构图食用。

安装Docker

链接一：https://www.runoob.com/docker/ubuntu-docker-install.html

链接二：https://cloud.tencent.com/developer/article/1167995

结合这些链接的教程做大致没有问题，但是还是有些问题hh。我们采用手动安装。

```
curl -fsSL https://get.docker.com | bash -s docker --mirror Aliyun
```

采用阿里云镜像来拉取docker包。

```
sudo apt-get update
```

更新apt索引。

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

安装apt依赖包，用于通过HTTPS来获取仓库。

至于HTTPS于HTTP的区别，HTTPS经由HTTP进行通信，但增加了一定安全措施。有些资源的标头是HTTPS协议，因此需要更新它需要的依赖来确保资源的获取。

```
curl -fsSL https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
```

添加Docker的GPG密钥。这里有些地方的教程采用的是官方源，会导致下载速度慢的情况。这里用的是ustc的镜像。

```
sudo add-apt-repository \
   "deb [arch=amd64] https://mirrors.ustc.edu.cn/docker-ce/linux/ubuntu/ \
```

```
  $(lsb_release -cs) \
  stable"
```

配置apt仓库镜像。同上有些教程采用的是官方源，会导致apt拉取速度慢。

```
sudo apt-get update
```

更新apt索引。

```
sudo apt install docker-ce
```

安装docker。

```
sudo systemctl status docker
```

测试docker是否安装成功。

```
ubuntu@VM-0-2-ubuntu:~$ systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2020-12-07 00:00:06 CST; 2 days ago
     Docs: https://docs.docker.com
 Main PID: 3574 (dockerd)
    Tasks: 31
   CGroup: /system.slice/docker.service
           ├─3574 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
           ├─5076 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 80 -container-ip 172.19.0.2 -container-port 80
           ├─5088 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8085 -container-ip 172.19.0.3 -container-port 8085
           └─5100 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 5000 -container-ip 172.19.0.3 -container-port 5000
```

★ 出现类似信息证明你已经抵达第一个里程碑。

并可以做进一步测试。

```
docker run hello-world
```

检查是否可以从Docker Hub访问和下载镜像。

```
ubuntu@VM-0-2-ubuntu:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

因为我之前下载过了，所以这里直接给的是成功的反馈，如果是第一次下载，可能还会有下载的过程。

如此以来，docker在ubuntu的安装就完成了。

配置Docker-compose&Dockerfile

从文件结构图中可以看到我们要配置一个Docker-compose文件于两个Dockerfile文件。对于它们的关系还有疑问的可以返回前面去结合系统结构和概述看看。那么开始配置。

Docker-compose.yml

```yaml
version: "2.3"
services:
  flask: #flask容器
    build: ./flaskapp#项目目录
    container_name: flask #容器名
    ports: #端口
      - "5000:5000"
      - "8085:8085"
    restart: always
    environment: #项目内application实例名
      - APP_NAME=app
    expose: #开放端口
      - 5000
  nginx:
    build: ./nginx #nginx目录
    container_name: nginx #容器名
```

```
        restart: always
        ports: #端口
          - "80:80"
```

nginx/Dockerfile

```
FROM nginx #采用nginx基础镜像
RUN rm /etc/nginx/conf.d/default.conf #删除系统默认配置
COPY nginx.conf /etc/nginx/conf.d/ #采用自己编写的nginx.conf
```

nginx/nginx.conf

```
server {
        listen 80; #监听80端口
        #server_name 127.0.0.1;
        location / {
            include      uwsgi_params;
            uwsgi_pass   flask:5000; #flask容器名:uwsgi端口
        }
}
```

flaskapp/Dockerfile

```
FROM python:3.4 #采用python3.4 基础镜像
MAINTAINER 2834552723@qq.com #作者邮箱
RUN mkdir /app #创建项目目录
WORKDIR /app #设置项目目录
ADD . /app #将当前目录文件复制进项目目录
#通过豆瓣镜像安装项目依赖包
RUN pip install -i https://pypi.doubanio.com/simple/  --trusted-host pypi.doubanio.com -r requirements.txt && pip install uwsgi -i https://pypi.doubanio.com/simple/  --trusted-host pypi.doubanio.com
#启动uwsgi服务
CMD ["uwsgi","--ini","uwsgi.ini"]
```

flaskapp/uwsgi.ini

```
[uwsgi]
socket=:5000 #端口
wsgi-file=manager.py #项目入口文件
callable=app #application实例
py-autoreload=1
processes=4
threads=2
master=True
buffer-size=32768
```

以上是具体配置文件以及各项内容的含义，具体是否成功还需要利用docker进行镜像容器生成才能知晓。

Docker构造镜像生成容器

先通过cd指令到达项目根目录，并进入管理员身份。

并进行docker-compose的镜像构造指令。

```
docker-compose build
```

这里就不再现build过程，一般来说上面的docker配置无误的话，这一步不会出现问题。

build完成后可以查看docker的镜像列表以确认是否完成构造。

```
docker-compose images
```

```
root@VM-0-2-ubuntu:/home/docker/demo/flask.demo# docker-compose images
Container     Repository       Tag       Image Id      Size
------------------------------------------------------------
flask       flaskdemo_flask   latest   03bb7c7d85df   905 MB
nginx       flaskdemo_nginx   latest   2d0c4e97dee2   127 MB
```

接着就可以尝试启动容器了，执行

```
docker-compose up
```

```
root@VM-0-2-ubuntu:/home/docker/demo/flask.demo# docker-compose up
Starting nginx ...
Starting flask ...
Starting nginx
Starting flask ... done
Attaching to nginx, flask
nginx    | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
nginx    | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
nginx    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
nginx    | 10-listen-on-ipv6-by-default.sh: error: /etc/nginx/conf.d/default.conf is not a file or does not exist
nginx    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
nginx    | /docker-entrypoint.sh: Configuration complete; ready for start up
flask    | [uWSGI] getting INI configuration from uwsgi.ini
flask    | *** Starting uWSGI 2.0.19.1 (64bit) on [Wed Dec  9 05:24:11 2020] ***
flask    | compiled with version: 6.3.0 20170516 on 08 December 2020 02:36:47
flask    | os: Linux-4.15.0-118-generic #119-Ubuntu SMP Tue Sep 8 12:30:01 UTC 2020
flask    | nodename: f644a1691c77
flask    | machine: x86_64
flask    | clock source: unix
flask    | pcre jit disabled
flask    | detected number of CPU cores: 1
flask    | current working directory: /app
flask    | detected binary path: /usr/local/bin/uwsgi
flask    | uWSGI running as root, you can use --uid/--gid/--chroot options
flask    | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
flask    | your memory page size is 4096 bytes
flask    | detected max file descriptor number: 1048576
flask    | lock engine: pthread robust mutexes
flask    | thunder lock: disabled (you can enable it with --thunder-lock)
flask    | uwsgi socket 0 bound to TCP address :5000 fd 3
flask    | uWSGI running as root, you can use --uid/--gid/--chroot options
flask    | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
flask    | Python version: 3.4.10 (default, Mar 20 2019, 00:50:15)  [GCC 6.3.0 20170516]
flask    | Python main interpreter initialized at 0x564a6a985a70
flask    | uWSGI running as root, you can use --uid/--gid/--chroot options
flask    | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
flask    | python threads support enabled
flask    | your server socket listen backlog is limited to 100 connections
flask    | your mercy for graceful operations on workers is 60 seconds
flask    | mapped 703600 bytes (687 KB) for 8 cores
flask    | *** Operational MODE: preforking+threaded ***
flask    | WSGI app 0 (mountpoint='') ready in 1 seconds on interpreter 0x564a6a985a70 pid: 1 (default app)
flask    | uWSGI running as root, you can use --uid/--gid/--chroot options
flask    | *** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
flask    | *** uWSGI is running in multiple interpreter mode ***
flask    | spawned uWSGI master process (pid: 1)
flask    | spawned uWSGI worker 1 (pid: 9, cores: 2)
flask    | spawned uWSGI worker 2 (pid: 10, cores: 2)
flask    | spawned uWSGI worker 3 (pid: 11, cores: 2)
flask    | spawned uWSGI worker 4 (pid: 12, cores: 2)
flask    | Python auto-reloader enabled
```

★ 可以看到flask容器与nginx容器反馈给我的运行log，如此一来我们就成功启动了，到达第二个里程碑。

并可以通过检查容器状态的指令来查看容器运行状态。

```
docker-compose ps
```

```
root@VM-0-2-ubuntu:/home/docker/demo/flask.demo# docker-compose ps
Name              Command               State                    Ports
-------------------------------------------------------------------------------------------
flask   uwsgi --ini uwsgi.ini           Up      0.0.0.0:5000->5000/tcp, 0.0.0.0:8085->8085/tcp
nginx   /docker-entrypoint.sh ngin ...   Up      0.0.0.0:80->80/tcp
```

接着我们应当进行浏览器访问测试，在其他终端浏览器中访问你的服务器ip。你可以看到nginx与flask的实时反馈。

```
flask    | 2020-12-09 05:27:25,443 INFO sqlalchemy.engine.base.Engine SHOW VARIABLES LIKE 'sql_mode'
flask    | 2020-12-09 05:27:25,443 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,446 INFO sqlalchemy.engine.base.Engine SHOW VARIABLES LIKE 'lower_case_table_names'
flask    | 2020-12-09 05:27:25,447 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,450 INFO sqlalchemy.engine.base.Engine SELECT DATABASE()
flask    | 2020-12-09 05:27:25,450 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,452 INFO sqlalchemy.engine.base.Engine show collation where `Charset` = 'utf8mb4' and `Collation` = 'utf8mb4_bin'
flask    | 2020-12-09 05:27:25,452 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,453 INFO sqlalchemy.engine.base.Engine SELECT CAST('test plain returns' AS CHAR(60)) AS anon_1
flask    | 2020-12-09 05:27:25,454 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,455 INFO sqlalchemy.engine.base.Engine SELECT CAST('test unicode returns' AS CHAR(60)) AS anon_1
flask    | 2020-12-09 05:27:25,455 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,456 INFO sqlalchemy.engine.base.Engine SELECT CAST('test collated returns' AS CHAR CHARACTER SET utf8mb4) COLLATE utf8mb4_bin AS anon_1
flask    | 2020-12-09 05:27:25,456 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,458 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
flask    | 2020-12-09 05:27:25,462 INFO sqlalchemy.engine.base.Engine SELECT account.id AS account_id, account.name AS account_name, account.password AS account_password
flask    | FROM account
flask    | 2020-12-09 05:27:25,462 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:25,471 INFO sqlalchemy.engine.base.Engine ROLLBACK
nginx    | 115.200.21.240 - - [09/Dec/2020:05:27:25 +0000] "GET / HTTP/1.1" 200 182 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.67 Safari/537.36 E
flask    | [pid: 12|app: 0|req: 1/1] 115.200.21.240 () {40 vars in 749 bytes} [Wed Dec  9 05:27:25 2020] GET / => generated 182 bytes in 66 msecs (HTTP/1.1 200) 2 headers in 80 bytes (1 switches on core 0)
nginx    | 115.200.21.240 - - [09/Dec/2020:05:27:25 +0000] "GET /favicon.ico HTTP/1.1" 200 13 "http://49.235.73.129/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
0.664.55" "-"
flask    | [pid: 12|app: 0|req: 2/2] 115.200.21.240 () {40 vars in 688 bytes} [Wed Dec  9 05:27:25 2020] GET /favicon.ico => generated 13 bytes in 1 msecs (HTTP/1.1 200) 2 headers in 79 bytes (1 switches o
flask    | 2020-12-09 05:27:27,714 INFO sqlalchemy.engine.base.Engine BEGIN (implicit)
flask    | 2020-12-09 05:27:27,715 INFO sqlalchemy.engine.base.Engine SELECT account.id AS account_id, account.name AS account_name, account.password AS account_password
flask    | FROM account
flask    | 2020-12-09 05:27:27,716 INFO sqlalchemy.engine.base.Engine {}
flask    | 2020-12-09 05:27:27,717 INFO sqlalchemy.engine.base.Engine ROLLBACK
nginx    | 83.97.20.35 - - [09/Dec/2020:05:27:27 +0000] "GET / HTTP/1.0" 200 182 "-" "-" "-"
flask    | [pid: 12|app: 0|req: 3/3] 83.97.20.35 () {30 vars in 313 bytes} [Wed Dec  9 05:27:27 2020] GET / => generated 182 bytes in 5 msecs (HTTP/1.0 200) 2 headers in 80 bytes (1 switches on core 1)
```

★ 并访问成功！里程碑三！

为什么里程碑二与里程碑三似乎之间没什么操作却要分别设置两个呢?
因为你在浏览器访问的测试阶段可能nginx与flask给你返回的是大量bug而不是成功交互的信息。。。。。。这里我吃了好些苦头，这里就不铺开细说，因为这一块每个人可能遇到的问题会有所不同。

另外，我们一般会将docker-compose放在后台运行，我们只需要更改docker-compose up的执行参数。

```
docker-compose up -d
root@VM-0-2-ubuntu:/home/docker/demo/flask.demo# docker-compose up -d
Starting nginx ...
Starting flask ...
Starting nginx
Starting flask ... done
root@VM-0-2-ubuntu:/home/docker/demo/flask.demo#
```

末

这篇博文是对之前的补充（另外还有一些没有用到的docker功能例如docker的类git功能还没有实际用过，之后如果有必要再做补充）。这样一来后端运维的基础设施算是完全搭建完毕（希望如此）。接下来已经公布赛题了，可以就具体问题去做出需求解决，希望中间层的推进已经到了可以利用一些专业知识来分析问题的程度吧。

以上。