

# 酒泉docker封装方法

2024/05/04 11:01

牛远卓

## 前情提要

本业务逻辑大致分为两部分：

- 1.docker外调用flask可执行文件（定时报废）以启动服务
- 2.浏览器url传文件路径以响应服务

flask的意义是为了用户能在本地浏览器傻瓜式调用服务器docker内的服务

## 前提条件

使用Vscode登录10.68.44.250：10242下的环境

```
# packages in environment at /mnt/sdd/niuyuanzhuo/anaconda3/envs/flask:
#
# Name                                Version                                Build Channel
_libgcc_mutex                         0.1                                    main
_openmp_mutex                         5.1                                   1_gnu
altgraph                             0.17.4                               pypi_0 pypi
ca-certificates                      2024.3.11                            h06a4308_0
certifi                              2020.12.5                             pypi_0 pypi
chardet                              4.0.0                               pypi_0 pypi
click                                7.1.2                               pypi_0 pypi
cyclor                               0.10.0                              pypi_0 pypi
flask                                1.1.2                               pypi_0 pypi
idna                                  2.10                                pypi_0 pypi
importlib-metadata                   7.1.0                               pypi_0 pypi
itsdangerous                         1.1.0                               pypi_0 pypi
jinja2                               2.11.3                              pypi_0 pypi
kiwisolver                           1.3.1                               pypi_0 pypi
ld_impl_linux-64                     2.38                                h1181459_1
libffi                               3.4.4                               h6a678d5_1
libgcc-ng                            11.2.0                              h1234567_1
libgomp                              11.2.0                              h1234567_1
libstdcxx-ng                         11.2.0                              h1234567_1
markupsafe                           1.1.1                               pypi_0 pypi
matplotlib                           3.4.1                               pypi_0 pypi
ncurses                              6.4                                 h6a678d5_0
```

numpy	1.20.2	pypi_0	pypi
openssl	3.0.13	h7f8727e_1	
packaging	24.0	pypi_0	pypi
pillow	8.2.0	pypi_0	pypi
pip	23.3.1	py39h06a4308_0	
pyinstaller	6.6.0	pypi_0	pypi
pyinstaller-hooks-contrib	2024.5	pypi_0	pypi
pyparsing	2.4.7	pypi_0	pypi
python	3.9.19	h955ad1f_0	
python-dateutil	2.8.1	pypi_0	pypi
readline	8.2	h5eee18b_0	
requests	2.25.1	pypi_0	pypi
setuptools	68.2.2	py39h06a4308_0	
six	1.15.0	pypi_0	pypi
sqlite	3.45.3	h5eee18b_0	
tk	8.6.12	h1ccaba5_0	
torch	1.8.1	pypi_0	pypi
torchaudio	0.8.1	pypi_0	pypi
torchvision	0.9.1	pypi_0	pypi
typing-extensions	3.7.4.3	pypi_0	pypi
tzdata	2024a	h04d1e81_0	
urllib3	1.26.4	pypi_0	pypi
werkzeug	1.0.1	pypi_0	pypi
wheel	0.41.2	py39h06a4308_0	
xz	5.4.6	h5eee18b_1	
zipp	3.18.1	pypi_0	pypi
zlib	1.2.13	h5eee18b_1	

将<https://github.com/vatsalgamit/Image-Classification-with-Pytorch-and-Flask>下载至本地电脑

## 调试接口

此时，main.py中MY\_FOLDER改为：

```
MY_FOLDER = os.path.join('app', 'static/uploads')
```

这一步先在“/mnt/sdb/nyz/nyz\_flask”地址下debug main.py，从而调整数据分析逻辑，注意不能使用gpu。主要修改下面这行

```
prediction = get_prediction(tensor)
```

终端响应如下即成功启动server服务

```
(base) niuyuanzhuo@ubuntu:/mnt/sdb/nyz/nyz_flask$ cd /mnt/sdb/nyz/nyz_flask ; /usr/bin/env /mnt/sdb/niuyuanzhuo/anaconda3/envs/flask/bin/python /home/niuyuanzhuo/.vscode-server/extensions/ms-python.python-2023.14.0/pythonFiles/lib/python/debugpy/adapter/../../debugpy/launcher 54843 -- /mnt/sdb/nyz/nyz_flask/app/main.py
* Serving Flask app "main" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8018/ (Press CTRL+C to quit)
127.0.0.1 - - [03/May/2024 22:31:59] "GET / HTTP/1.1" 200 -
```

本地浏览器输入url。以后验证各种程序是否跑通，都在浏览器下输入以下url

`http://localhost:app端口号/网页相对路径?入参1=入参内容1&入参2=入参内容2`

app端口号为main.py中的端口号

`app.run(host='0.0.0.0', debug=False, threaded=False, port=8018)`

网页相对路径为main.py中的网页相对路径

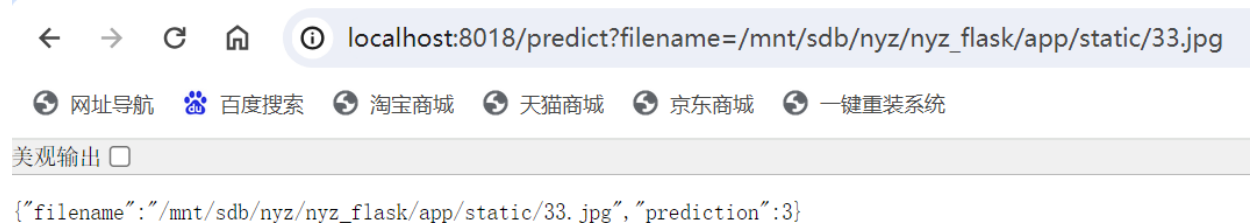
`@app.route('/predict', methods = ['GET', 'POST'])`

入参与入参内容需要自己设计。若需要两组或多组入参与入参内容，需要使用&隔开。

其中入参内容基本上是共享图片地址（server与client都能见）。由于我暂时没有共享文件系统，因此我给的文件地址是server文件系统下的。此时我的server是在10242服务器中，于是文件路径也要改成10242服务器的路径。由于我只需要传一个入参，因此不需要&。形如：

`http://localhost:8018/predict?filename=/mnt/sdb/nyz/nyz_flask/app/static/33.jpg`

跑完示例，可以看到返回的是json格式的输出



← → ↻ 🏠 ⓘ localhost:8018/predict?filename=/mnt/sdb/nyz/nyz\_flask/app/static/33.jpg

🌐 网址导航 🐶 百度搜索 🛒 淘宝商城 🐾 天猫商城 🛒 京东商城 🔄 一键重装系统

美观输出 ☐

```
{ "filename": "/mnt/sdb/nyz/nyz_flask/app/static/33.jpg", "prediction": 3 }
```

## 撰写接口文档

大家可以根据业务需求更改，但是出参需要是json格式。以下为一个分类任务的示例。

入参：

filename=文件地址，形如filename=/mnt/sdb/nyz/nyz\_flask/app/static/33.jpg

出参：

1.数据分析成功时：prediction是分类结果，filename是输入文件路径，errcode是是否报错

形如 {'prediction': prediction.item(), 'filename': filename, 'errcode': 1}

2.数据分析失败时: error是报错原因, errcode是是否报错

形如 {'error':'error during prediction', 'errcode': 0}

## 终端运行

假设调试没问题, 则尝试下在终端运行

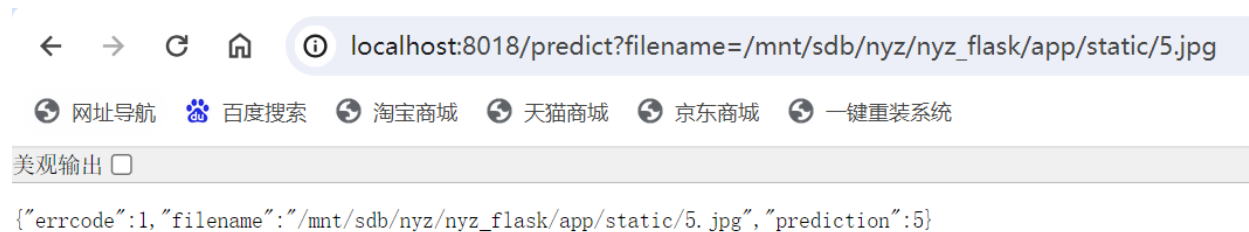
此时, MY\_FOLDER为

```
MY_FOLDER = os.path.join('static/uploads')
```

在"/mnt/sdb/nyz/nyz\_flask/app"下终端运行

```
python main.py
```

换张图片, 同样没问题



成功后在运行文件前加入定时操作

```
import datetime
expiration_date = datetime.date(2024, 4, 25) + datetime.timedelta(days=180)
current_date = datetime.date.today()

assert current_date < expiration_date
if current_date > expiration_date:
    exit()
```

## 封装可执行文件

假设源码运行没问题, 下一步是封装可执行文件, 并在通过可执行文件启动服务

此时, MY\_FOLDER改为

```
MY_FOLDER = os.path.join(sys._MEIPASS, 'static/uploads')
```

原因是可执行文件运行时，按照自己的绝对路径寻找文件（该绝对路径与寻常的终端程序不见面）。可执行文件运行时，会将自己的绝对路径存储在sys.\_MEIPASS中。而单纯在main.py中，改成自己终端的程序路径会出错。详情请见<https://pyinstaller.org/en/stable/runtime-information.html#using-file>

打包成可执行文件

```
pip install pyinstaller
pyinstaller -w -F --add-data "templates:templates" --add-data "static:static" --add-data "app:app" main.py
or
pyinstaller -F --add-data "module:module" --add-data "static:static" --add-data "ultralytics:ultralytics" main.py
```

同样的，自己main.py需要依赖的程序文件夹（推荐将需要的文件放入文件夹中）需要复制到运行时的绝对路径，这就需要--add-data "source:dest"。引号内，冒号前是本地文件夹相对路径，后者是运行时文件加相对路径。两者都写一样就行。详情请见<https://elc.github.io/posts/executable-flask-pyinstaller/>

打开生成的main.spec文件，可以看到文件夹复制成功

```
a = Analysis(
    ['main.py'],
    pathex=[],
    binaries=[],
    datas=[('templates', 'templates'), ('static', 'static'), ('app', 'app')],
    hiddenimports=[],
    hookspath=[],
    hooksconfig={},
    runtime_hooks=[],
    excludes=[],
    noarchive=False,
    optimize=0,
)
```

最后，根据输出可执行文件地址运行之

```
182874 INFO: Copying bootloader EXE to /mnt/sdb/nyz/nyz_flask/app/dist/main
182874 INFO: Appending PKG archive to custom ELF section in EXE
195440 INFO: Building EXE from EXE-00.toc completed successfully.
```

```
/mnt/sdb/nyz/nyz_flask/app/dist/main
```

没问题

## docker部署

生成好可执行文件后，先在服务器上从app文件夹移除main.py以及其他关键程序或数据

写Dockerfile，并放入app文件夹的同一层。主要注意python版本与项目环境一致即可

```
# syntax=docker/dockerfile:1

FROM python:3.8-slim-buster

COPY ./app /app
COPY ./app/requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt -i
https://pypi.tuna.tsinghua.edu.cn/simple

EXPOSE 8008
CMD [ "/app/dist/main"]
```

若不知道每行意思，可以参考<https://www.freecodecamp.org/news/how-to-dockerize-a-flask-app/>  
在Dockerfile路径下，形成镜像

```
docker build -t 自己的镜像名:版本号 .
```

e.g.

```
docker build -t nyz:v16 .
```

外部运行容器调用容器内可执行文件，**注意不能使用gpu**

```
docker run -it -p 8018:8018 nyz:v16
```

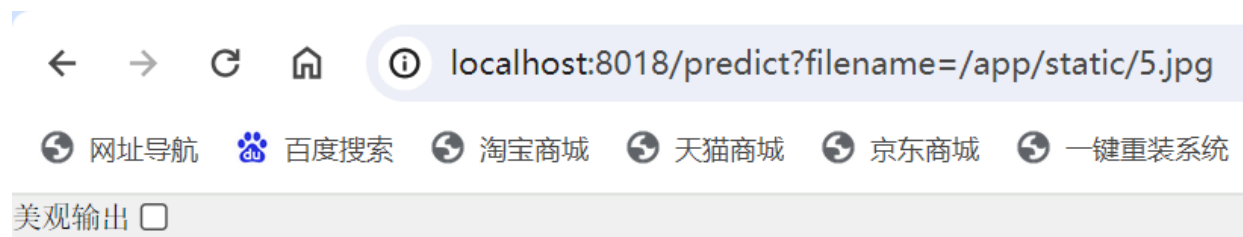
注意-p后的两个数字最好和main.py中端口号都一致，若端口占用，最好一起更改

```
app.run(host='0.0.0.0', debug=False, threaded=False, port=8018)
```

在浏览器响应服务，没问题

```
http://localhost:8018/predict?filename=/app/static/5.jpg
```

同样的，由于我暂时没有共享文件系统，因此我给的文件地址是server文件系统下的。此时我的server是在docker中，于是文件路径也要改成docker的路径。docker的文件路径需要参考Dockerfile。



```
{"errcode":1,"filename":"/app/static/5.jpg","prediction":5}
```

## 打包image.tar

之前都没有后台生成运行的容器，现在需要运行的容器以知道它的容器ID

```
docker run -itd -p 8018:8018 nyz:v16
docker ps
```

这时候会看到镜像名为nyz:v16的容器ID

commit为新的镜像：

```
docker commit -c "CMD /app/dist/main" (可以是Dockerfile中的CMD，也可能不用" 容器ID
nyz:v16(改成自己打包的)
```

保存image.tar

```
docker save -o /算法文件夹/image.tar(改成绝对路径) nyz:v16(改成自己打包的)
```

若生成的image.tar大于4.38GB（项目光盘容量），则需要分解image.tar为不同满足大小要求的images.tars。具体方法如下：

```
split -b 4000M image.tar "image_part_"
```

当然，也可以先更改image.tar的名字再分解

## 附录

个人记录使用。网页版main.py

```
from flask import Flask, request, jsonify, render_template
from torch_utils import transform_image, get_prediction
from werkzeug.utils import secure_filename
import os
import sys
import datetime
expiration_date = datetime.date(2024, 4, 25) + datetime.timedelta(days=180)
current_date = datetime.date.today()

assert current_date < expiration_date
if current_date > expiration_date:
    exit()

app = Flask(__name__)#, static_folder= os.getcwd() +
'/static', template_folder=os.getcwd() + '/templates'
# if getattr(sys, 'frozen', False):
#     template_folder = os.path.join(sys._MEIPASS, 'templates')
#     app = Flask(__name__, template_folder=template_folder)
# else:
```

```

# app = Flask(__name__)

# MY_FOLDER = os.path.join('app', 'static/uploads') # debug时
# MY_FOLDER = os.path.join('static/uploads') # 在终端运行时
MY_FOLDER = os.path.join(sys._MEIPASS, 'static/uploads') # 本地运行可执行文件时
UPLOAD_FOLDER = MY_FOLDER

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.',1)[1].lower() in
ALLOWED_EXTENSIONS

@app.route('/', methods = ['GET', 'POST'])
def index():
    return render_template('index.html')

@app.route('/predict', methods = ['GET', 'POST'])
def predict():
    if request.method == 'POST':
        file = request.files['file']
        filename = file.filename
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)

        if file is None or file.filename == "":
            return jsonify({'error': 'no-file'})
        if not allowed_file(file.filename):
            return jsonify({'error': "format not supported"})

        try:
            file.seek(0)
            image_bytes = file.read()
            tensor = transform_image(image_bytes)
            prediction = get_prediction(tensor)
            data = {'prediction': prediction.item()}
            # print(data)
            return render_template("index.html", prediction = data['prediction'],
file_path = file_path)
        except:
            return jsonify({'error': 'error during prediction'})

if __name__ == '__main__':
    app.debug = True
    app.run(host='0.0.0.0', debug=False, threaded=False, port=8018)

```



[https://github.com/danlim-wz/flask\\_image\\_classifier](https://github.com/danlim-wz/flask_image_classifier)