

Docker部署项目

Docker部署项目

1. Docker部署方式
2. 安装Docker社区版
3. 构建容器镜像
 - 3.1. Dockerfile编写
 - 3.2. 项目初始化启动脚本
 - 3.3. 编写.dockerignore文件
 - 3.4. 命令行方式构建容器镜像
 - 3.5. pycharm方式构建容器镜像
 - 3.6. Docker Hub方式构建容器镜像
4. 启动容器
5. 使用第三方容器化服务
 - 5.1. 添加MySQL容器
 - 5.2. 添加Elasticsearch容器
6. 总结

1. Docker部署方式

docker容器化技术是建立在轻量级虚拟化技术基础上的，允许应用程序及其依赖和配置完全隔离宿主主机地运行，而不需要使用虚拟机等完整的虚拟化解决方案。

让应用程序在Docker容器中运行的最大的好处之一是，一旦该容器通过了本地测试，就可以将它们运行到任何提供Docker支持的平台。

2. 安装Docker社区版

目前有两个版本的Docker，免费的社区版（CE）和付费的企业版（EE），本次使用为Docker CE。

要使用Docker CE，首先必须将其安装在系统上。在[Docker网站](#)上有适用于Windows，Mac OS X和多个Linux发行版的安装程序。如果你正在使用Microsoft Windows系统，请务必注意Docker CE依赖Hyper-V。如有必要，安装程序将为你启用此功能，但启用Hyper-V会限制诸如VirtualBox等其他虚拟化技术产品的运行。同时安装的Microsoft Windows系统也有版本要求，需要Win10专业版或者企业版的系统。

安装完成后，可通过在终端窗口或命令提示符处输入 `docker version` 命令来验证安装是否成功：

```
1  $ docker version
2  Client: Docker Engine - Community
3  Version:           18.09.2
4  API version:       1.39
5  Go version:        go1.10.8
6  Git commit:        6247962
7  Built:             Sun Feb 10 04:12:31 2019
8  OS/Arch:           windows/amd64
```

```
9   Experimental:      false
10
11  Server: Docker Engine - Community
12  Engine:
13    Version:          18.09.2
14    API version:       1.39 (minimum version 1.24)
15    Go version:        go1.10.6
16    Git commit:        6247962
17    Built:             Sun Feb 10 04:28:48 2019
18    OS/Arch:           windows/amd64
19    Experimental:      true
```

3. 构建容器镜像

为项目创建容器的第一步是为它构建一个**镜像**。容器镜像是用于创建容器的模板。它包含容器文件系统的完整表示，以及与网络，启动选项等相关的各种设置。

更好的方法是通过脚本生成容器镜像。创建脚本化容器镜像的命令是 `docker build`。该命令从一个名为 `Dockerfile` 的文件读取并执行构建指令（我需要创建这些指令）。`Dockerfile` 基本上可以认为是一个安装程序脚本，它执行安装步骤来部署应用程序，以及一些容器特定的设置。

3.1. Dockerfile编写

在项目目录下创建 `Dockerfile` 用于生成项目容器镜像。

```
1  FROM python:3.6-alpine
2
3  RUN apk add --no-cache --virtual=build-dependencies g++ zlib-dev jpeg-dev && \
4      adduser -D microblog
5
6  WORKDIR /home/microblog
7
8  COPY requirements.txt requirements.txt
9  RUN python -m venv venv && \
10     venv/bin/pip install --no-cache-dir -r requirements.txt -i
11     https://pypi.doubanio.com/simple/ --trusted-host pypi.doubanio.com && \
12     venv/bin/pip install --no-cache-dir gunicorn -i https://pypi.doubanio.com/simple/
13     --trusted-host pypi.doubanio.com
14
15  COPY app app
16  COPY migrations migrations
17  COPY microblog.py boot.sh ./
18  RUN chmod +x boot.sh
19
20  ENV FLASK_APP microblog.py
21
22  RUN chown -R microblog:microblog ./
23  USER microblog
24
25  EXPOSE 5000
26  ENTRYPOINT ["/boot.sh"]
```

Dockerfile 文件中命令说明如下：

- FROM python:3.6-alpine

FROM 命令指定将在其上构建新镜像的基础容器镜像。这样就可以从一个现有的镜像开始，添加或改变一些东西，并最终得到一个派生的镜像。镜像由名称和标签来标记，之间用冒号分隔。该标签用作版本控制机制，允许容器镜像提供多个版本。现在选择的镜像的名称是 python，它是Python的官方Docker镜像。该镜像的标签允许指定解释器版本和基础操作系统。3.6-alpine 标签是安装在Alpine Linux上的Python 3.6解释器。由于其体积小，Alpine Linux发行版比起更常见的发行版（例如Ubuntu）使用更广泛。可以在[Python镜像库](#)中查看可用的Python镜像标签。

- RUN apk add --no-cache --virtual=build-dependencies g++ zlib-dev jpeg-dev && adduser -D microblog

RUN 命令在容器的上下文中执行任意命令。与在shell提示符下输入命令相似：

1. apk add --no-cache --virtual=build-dependencies g++ zlib-dev jpeg-dev 命令是使用Alpine Linux的包管理器来安装 Pillow 相关依赖包，其中 --no-cache 表示无缓存，--virtual=build-dependencies 表示创建依赖包；

2. adduser -D microblog 命令创建一个名为 microblog 的新用户。大多数容器镜像都使用 root 作为默认用户，但以root身份运行应用程序并不是一个好习惯，所以此处创建了自己的用户。

- WORKDIR /home/microblog

WORKDIR 命令设置将要安装应用程序的默认目录。当创建 microblog 用户时，会自动创建了一个主目录，所以现在将该目录设置为默认目录。在 Dockerfile 中的任何剩余命令执行时，以及运行容器时，其当前目录均为这个默认目录。

- COPY requirements.txt requirements.txt

COPY 命令将文件从宿主机复制到容器文件系统。该命令需要两个或更多参数，源文件/目录和目标文件/目录。源文件必须与 Dockerfile 所在的目录相关。目的地可以是绝对路径，也可以是相对于 WORKDIR 命令中设置目录的路径。在这第一个 COPY 命令中，将 requirements.txt 文件复制到容器文件系统的 microblog 用户的主目录中。

- RUN python -m venv venv && \ venv/bin/pip install --no-cache-dir -r requirements.txt -i https://pypi.doubanio.com/simple/ --trusted-host pypi.doubanio.com && \ venv/bin/pip install --no-cache-dir gunicorn -i https://pypi.doubanio.com/simple/ --trusted-host pypi.doubanio.com

使用 RUN 命令创建一个虚拟环境：

1. 根据 requirements.txt 文件在虚拟环境中安装所有依赖包，其中 --no-cache-dir 表示无缓存，-i https://pypi.doubanio.com/simple/ --trusted-host pypi.doubanio.com 为指定豆瓣下载源，加快安装速度；

2. 由于依赖文件仅包含通用依赖项，因此需要明确安装 gunicorn，以将其用作Web服务器。当然，也可以在 requirements.txt 文件中添加 gunicorn。

- COPY app app

COPY 命令复制 app 包到容器用户根目录中。

- COPY migrations migrations

COPY 命令复制含有数据库迁移的 migrations 目录到容器用户根目录中。

- COPY microblog.py boot.sh ./

`COPY` 命令复制 `microblog.py` 和 `boot.sh` 脚本到容器用户根目录中。其中 `boot.sh` 脚本用于容器启动后的项目初始化及服务启动命令配置。

- `RUN chmod +x boot.sh`

`RUN chmod` 命令确保将这个新的 `boot.sh` 文件正确设置为可执行文件。如果你使用的是基于 Unix 的文件系统，并且源文件已被标记为可执行文件，则复制的文件将会保留可执行权限。此处显式地对其进行授权，是因为在 Windows 上很难设置可执行权限。

- `ENV FLASK_APP microblog.py`

`ENV` 命令在容器中设置环境变量。

- `RUN chown -R microblog:microblog ./`

`RUN chown` 命令将存储在 `/home/microblog` 中的所有目录和文件的所有者设置为新的 `microblog` 用户，以便在容器启动时该用户可以正确运行这些文件。

- `USER microblog`

`USER` 命令使得这个新的 `microblog` 用户成为任何后续指令的默认用户，并且也是容器启动时的默认用户。

- `EXPOSE 5000`

`EXPOSE` 命令配置该容器将用于服务的端口，以便 Docker 可以适当地在容器中配置网络。

- `ENTRYPOINT ["/boot.sh"]`

`ENTRYPOINT` 命令定义了容器启动时应该执行的默认命令。这是启动应用程序 Web 服务器的命令。

3.2. 项目初始化启动脚本

在项目目录下新建 `boot.sh` 脚本，用于设置容器启动后的项目初始化处理。

需要注意 `gunicorn` 命令之前的 `exec`。在 shell 脚本中，`exec` 触发正在运行脚本的进程被给定的命令来替换掉，而不是将这个命令作为新进程启动。这很重要，因为 Docker 会将容器的生命与其上运行的第一个进程关联起来。在像这样的情况下，启动进程不是容器的主进程，就需要用主进程取代启动进程，以确保容器不会提前停止。

Docker 的一个有趣的方面是容器写入 `stdout` 或 `stderr` 的任何内容都将被捕获并存储为容器的日志。出于这个原因，`--access-logfile` 和 `--error-logfile` 都配置为 `-`，将会把日志发送到标准输出，以便由 Docker 作为日志存储。

注意：此处脚本中，行分割方式需要设置为 仅 LF (UNIX) 模式。

```
1 #!/bin/sh
2 source venv/bin/activate
3 flask db upgrade
4 flask translate compile
5 exec gunicorn -b :5000 --access-logfile - --error-logfile - microblog:app
```

3.3. 编写 .dockerignore 文件

新建 `.dockerignore` 文件，用于忽略无需打包的文件或目录，以减少构建容器镜像的大小。该文件中的内容与 `.gitignore` 一样即可。

3.4. 命令行方式构建容器镜像

通过 `docker build` 命令构建容器镜像，通过 `-t` 参数设置了新容器镜像的名称和标签。 `.` 表示容器构建的基础目录，这就是 `Dockerfile` 所在的目录。容器镜像名称一般定义为 `<登录Docker Hub的用户名>/<镜像名称>`，这样便于云端镜像仓库管理。

执行构建命令后，`docker` 会按照 `Dockerfile` 中配置的步骤，一步一步执行。

```
1  $ docker build -t superwong/microblog:latest .
2  Sending build context to Docker daemon 29.28MB
3  Step 1/14 : FROM python:3.6-alpine
4  ----> 35bb01a3d284
5  Step 2/14 : RUN apk add --no-cache --virtual=build-dependencies g++ zlib-dev jpeg-dev
   && adduser -D microblog
6  ----> Using cache
7  ----> 3661b34a4049
8  Step 3/14 : WORKDIR /home/microblog
9  ----> Using cache
10 ----> e812dc4f91a3
11 Step 4/14 : COPY requirements.txt requirements.txt
12 ----> Using cache
13 ----> 76c77f8ddb5c
14 Step 5/14 : RUN python -m venv venv && venv/bin/pip install --no-cache-dir -r
   requirements.txt -i https://pypi.doubanio.com/simple/ --trusted-host pypi.doubanio.com
   &&
15     venv/bin/pip install --no-cache-dir gunicorn -i https://pypi.doubanio.com/simple/ -
   -trusted-host pypi.doubanio.com
16 ----> Using cache
17 ----> 06d643204bed
18 Step 6/14 : COPY app app
19 ----> a8af1ce840bf
20 Step 7/14 : COPY migrations migrations
21 ----> 9847d1f29ec1
22 Step 8/14 : COPY microblog.py boot.sh ./
23 ----> 0247e7ebc616
24 Step 9/14 : RUN chmod +x boot.sh
25 ----> Running in 51d19632ebaa
26 Removing intermediate container 51d19632ebaa
27 ----> ec10b4d9d5ea
28 Step 10/14 : ENV FLASK_APP microblog.py
29 ----> Running in a1a407d7a06c
30 Removing intermediate container a1a407d7a06c
31 ----> c58ece72866b
32 Step 11/14 : RUN chown -R microblog:microblog ./
33 ----> Running in a460949ff96f
34 Removing intermediate container a460949ff96f
35 ----> 3f1f50eab975
36 Step 12/14 : USER microblog
37 ----> Running in a13ff1f4bf28
38 Removing intermediate container a13ff1f4bf28
39 ----> b0477817b084
40 Step 13/14 : EXPOSE 5000
41 ----> Running in 401e70eabb7b
42 Removing intermediate container 401e70eabb7b
43 ----> f4e70b2afaab
```

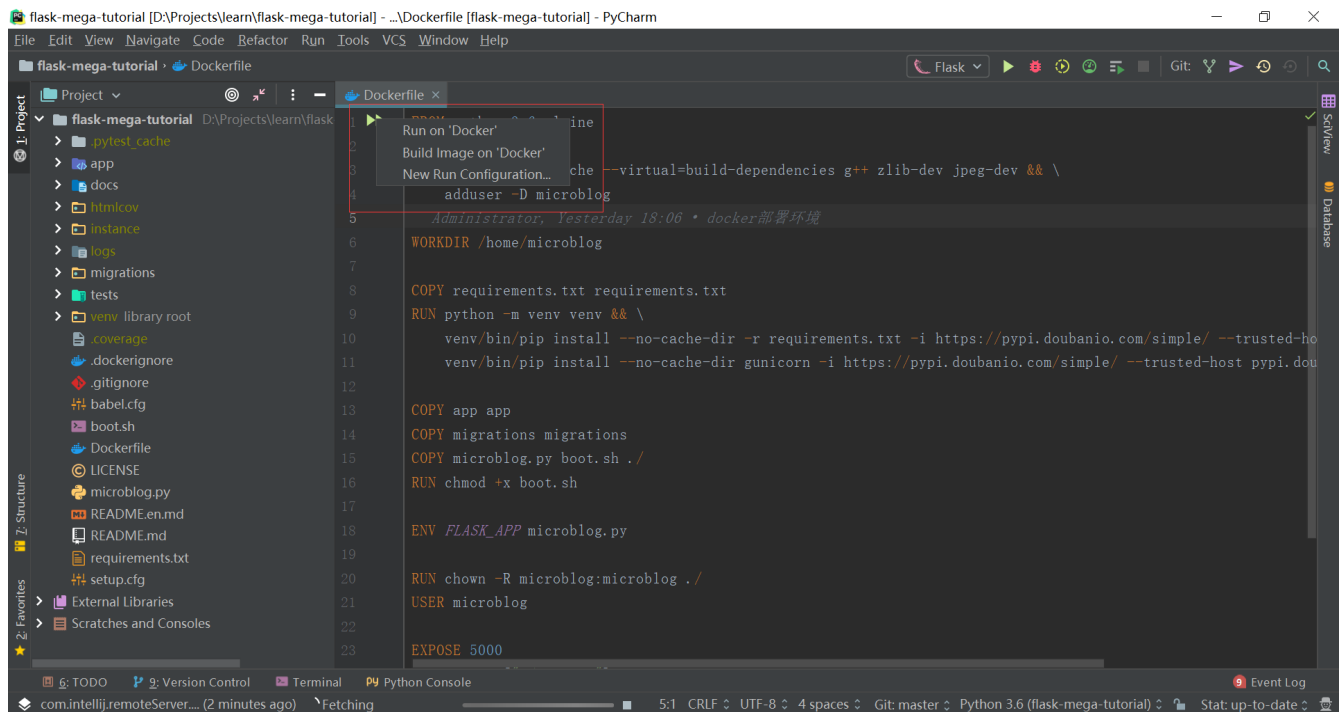
```
44 Step 14/14 : ENTRYPOINT ["/boot.sh"]
45 ----> Running in 8717892fa444
46 Removing intermediate container 8717892fa444
47 ----> f9ed6c8df228
48 Successfully built f9ed6c8df228
49 Successfully tagged superwong/microblog:latest
```

执行完成后查看本地生成的镜像

```
1 $ docker images
2 REPOSITORY          TAG                 IMAGE ID           CREATED            SIZE
3 superwong/microblog latest             f9ed6c8df228      3 minutes ago     435MB
4 python              3.6-alpine         35bb01a3d284      3 days ago        92MB
5 alpine              latest             055936d39205      3 days ago
6 hello-world         latest             b02214b8df4e      4 weeks ago       250MB
```

3.5. pycharm方式构建容器镜像

除了通过命令行的方式构建容器镜像意外，还可以通过pycharm图形化的方式构建。在pycharm中打开 `Dockerfile` 文件，点击代码区域左上角的双箭头标志，在弹出框中选择“New Run Configuration...”编写构建任务。



在弹出的配置窗口中配置tag参数，配置完成点击Run按钮执行创建任务。



Edit Run Configuration



Name:

☐ Share

☐ Allow parallel run

Server: 



Dockerfile:



Context folder:

.



Image tag:

superwong/microblog:latest

Build args:



Build options:



☒ Run built image

Container name:

Executable

Entrypoint:



Command:



Publish exposed ports to the host interfaces: ☐ All ☒ Specify

Bind ports:



Bind mounts:



Environment variables:



Run options:

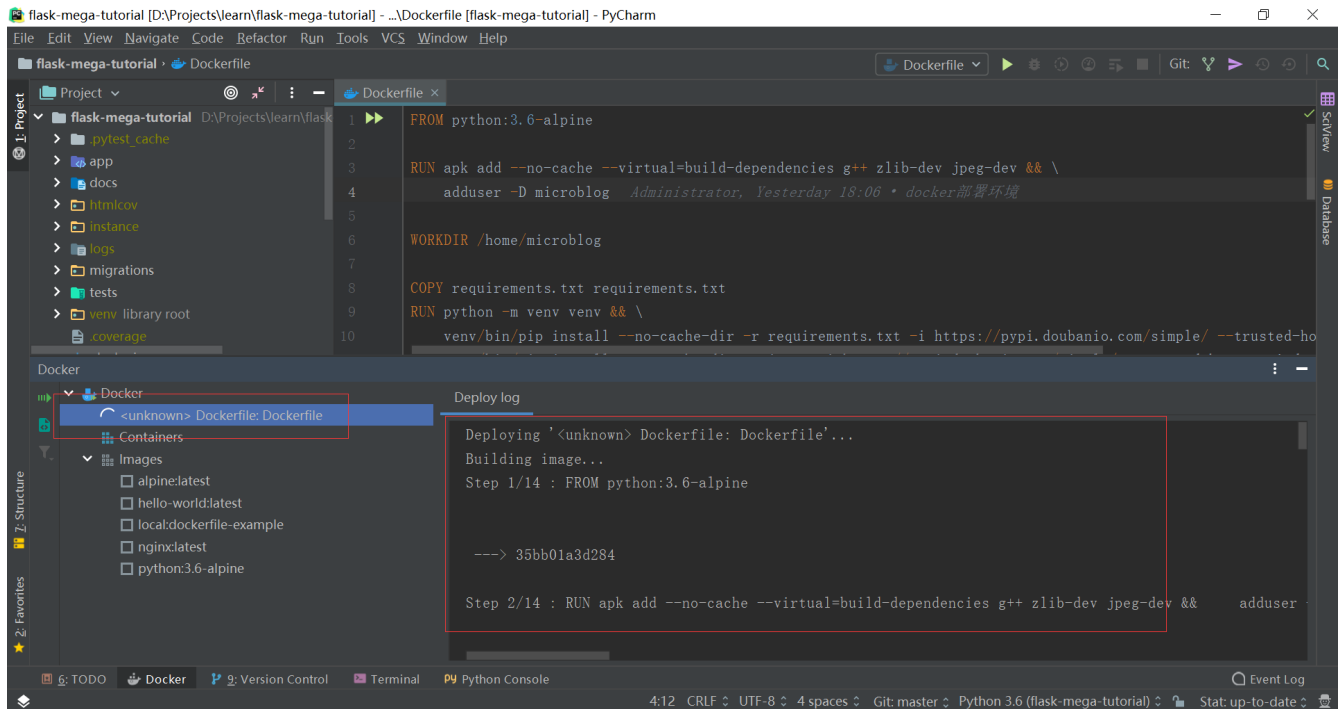


 Run

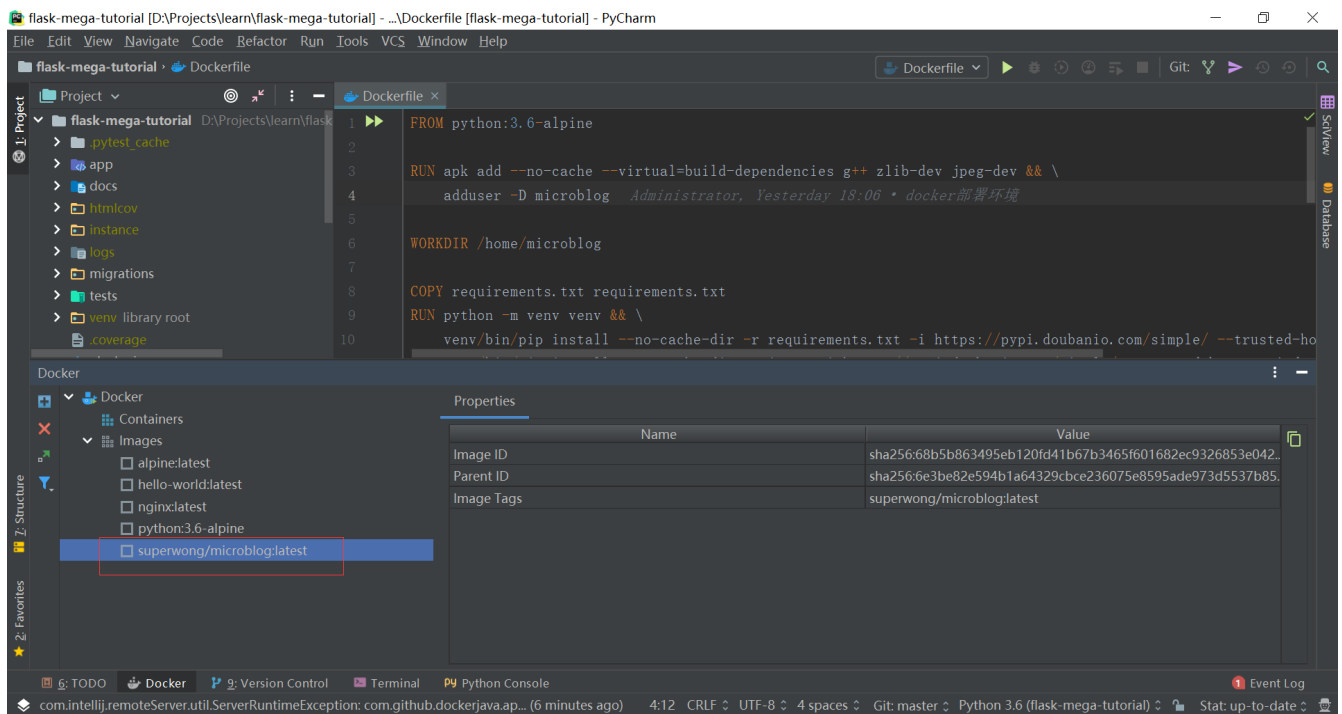
Cancel

Apply

弹出Docker操作界面，启动一个创建任务，开始构建镜像。



最终生成镜像文件



3.6. Docker Hub方式构建容器镜像


可以通过登录[docker hub](https://hub.docker.com/)，新建仓库

← → × <https://hub.docker.com/?ref=login> ☆ ▼ ⓘ ⋮

Download and Take a Tutorial


Get started by downloading Docker Desktop, and learn how you can build, tag and share a sample image on Hub.

[Get started with Docker Desktop](#)



Create a Repository

Push container images to Docker Hub





Create an Organization

Manage Docker Hub repositories with your team


Access the world's largest library of container images

填入需要创建的仓库的信息，选择是否公开。同时需要绑定自己的github账号，设置对应的项目即可实现github更新后，镜像自动重新构建。

 **docker hub** Explore Repositories Organizations Get Help ▼ superwong ▼ 

Repositories > Create Using 0 of 1 private repositories. [Get more](#)


Create Repository


 superwong microblog

flask-mega-tutorial

Visibility

Using 0 of 1 private repositories. [Get more](#)

☐ **Public**  Public repositories appear in Docker Hub search results

☒ **Private**  Only you can view private repositories

Build Settings (optional)

Autobuild triggers a new build with every `git` push to your source code repository. [Learn More](#)

Pro tip

You may push a new image to this repository using the CLI:

```
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
```

Make sure to change *tagname* with your desired image repository tag.

← → ↻ <https://cloud.docker.com/repository/create> ☆ ▼ 👤 ⋮

tag.

Visibility


Using 0 of 1 private repositories. [Get more](#)


☐ **Public** Public repositories appear in Docker Hub search results

☒ **Private** Only you can view private repositories

Build Settings *(optional)*

Autobuild triggers a new build with every `git push` to your source code repository. [Learn More](#).


Connected


Disconnected

Cancel

Create

Create & Build

EXPLORE


ACCOUNT

PUBLISH

RESOURCES

SUPPORT

创建完成后，进行github项目绑定。

 Explore Repositories Organizations Get Help ▼ superwong ▼ 👤

Repositories > superwong / microblog > Using 0 of 1 private repositories. [Get more](#)

General

Tags

BUILDS


Timeline


Collaborators


Webhooks

Settings

We made some changes to our autobuilds. [Learn more](#). X

 **superwong / microblog**

flask-mega-tutorial 

 Last pushed: never


Docker commands


To push a new tag to this repository,

`docker push superwong/microblog:tagname`

Tags

This repository is empty. When it's not empty, you'll see a list of the most recent tags here.



[Explore](#) [Repositories](#) [Organizations](#) [Get Help](#) [superwong](#) 

[Repositories](#) [superwong / microblog](#) [Builds](#) Using 0 of 1 private repositories. [Get more](#)

[General](#) [Tags](#) [Builds](#) [Timeline](#) [Collaborators](#) [Webhooks](#) [Settings](#)


We made some changes to our autobuilds. [Learn more.](#)


[Configure Automated Builds](#)


Automated Builds


Autobuild triggers a new build with every **git push** to your source code repository. [Learn More.](#)

ENABLE AUTOBUILD BY CONNECTING TO AN EXTERNAL REPOSITORY SOURCE


Link to GitHub
Connected


Link to Bitbucket
Disconnected



[Explore](#) [Repositories](#) [Organizations](#) [Get Help](#) [superwong](#) 


[Repositories](#) [superwong / microblog](#) [Builds](#) [Edit](#) Using 0 of 1 private repositories. [Get more](#)


[General](#) [Tags](#) [Builds](#) [Timeline](#) [Collaborators](#) [Webhooks](#) [Settings](#)

We made some changes to our autobuilds. [Learn more.](#)

Build configurations

SOURCE REPOSITORY

 superwong

 flask-mega-tutorial

NOTE: Changing source repository may affect existing build rules.

BUILD LOCATION

Build on Docker Hub's infrastructure

AUTOTEST


☒ Off


☐ Internal Pull Requests

☐ Internal and External Pull Requests



REPOSITORY LINKS

☐ Off


☒ Enable for Base Image 

BUILD RULES 

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Context 	Autobuild	Build Caching	
<div>Branch</div>	master	latest	Dockerfile	/	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

[View example build rules](#)

BUILD ENVIRONMENT VARIABLES 

Cancel


Save

Save and Build

点击保存后，跳转至自动创建镜像页面，点击“Trigger”按钮，可以发起创建任务。

Automated Builds


Autobuild triggers a new build with every **git push** to your source code repository. [Learn More.](#)

 [superwongo/flask-mega-tutorial](#)

 | Use Docker Hub's infrastructure | Autotests: Off

Docker Tag	Source	Build Status	Autobuild	Build caching	
latest	master	EMPTY	✓	✓	<div>Trigger ▶</div>


Recent Builds

 [Github Ping](#)


🕒 a few seconds ago

Automated Builds


Autobuild triggers a new build with every **git push** to your source code repository. [Learn More.](#)

 [superwongo/flask-mega-tutorial](#)


 | Use Docker Hub's infrastructure | Autotests: Off

Docker Tag	Source	Build Status	Autobuild	Build caching	
latest	master	BUILDING	✓	✓	


Recent Builds


 [Build in 'master' \(e4bcf15f\)](#)

[e4bcf15](#)

 latest

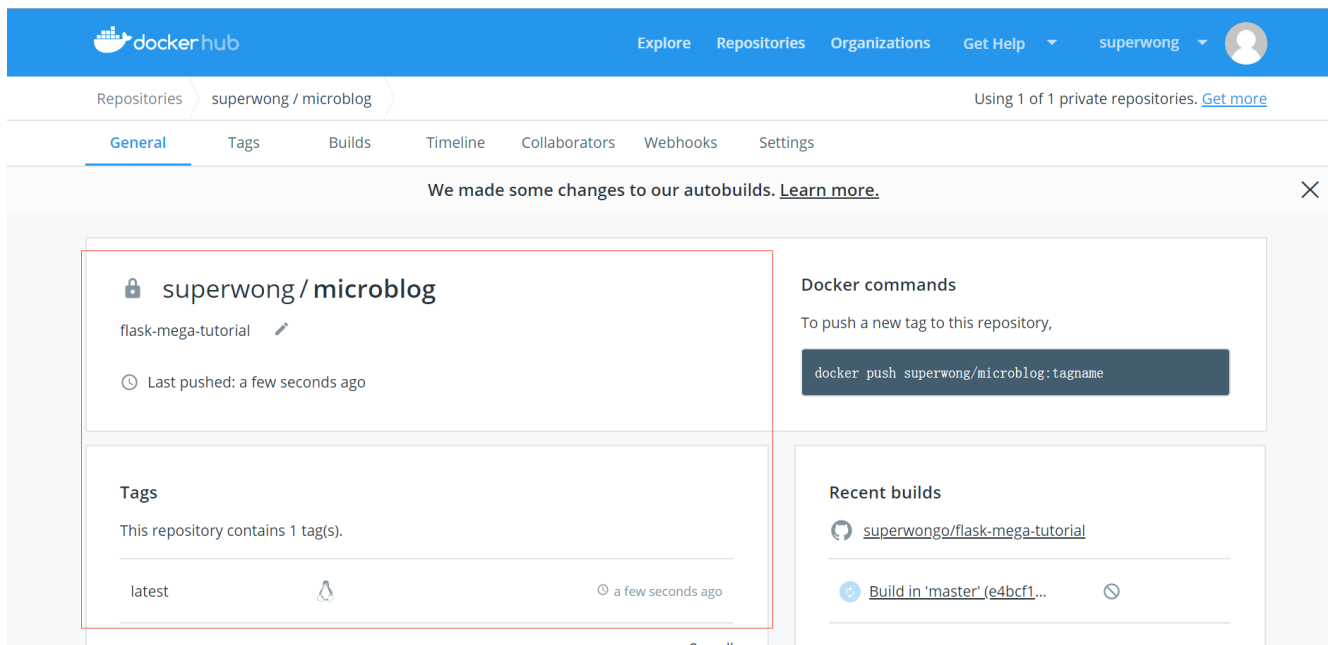
🕒 a few seconds ago



 [Github Ping](#)

🕒 a few seconds ago

构建完成后，既可以在仓库首页查看到。

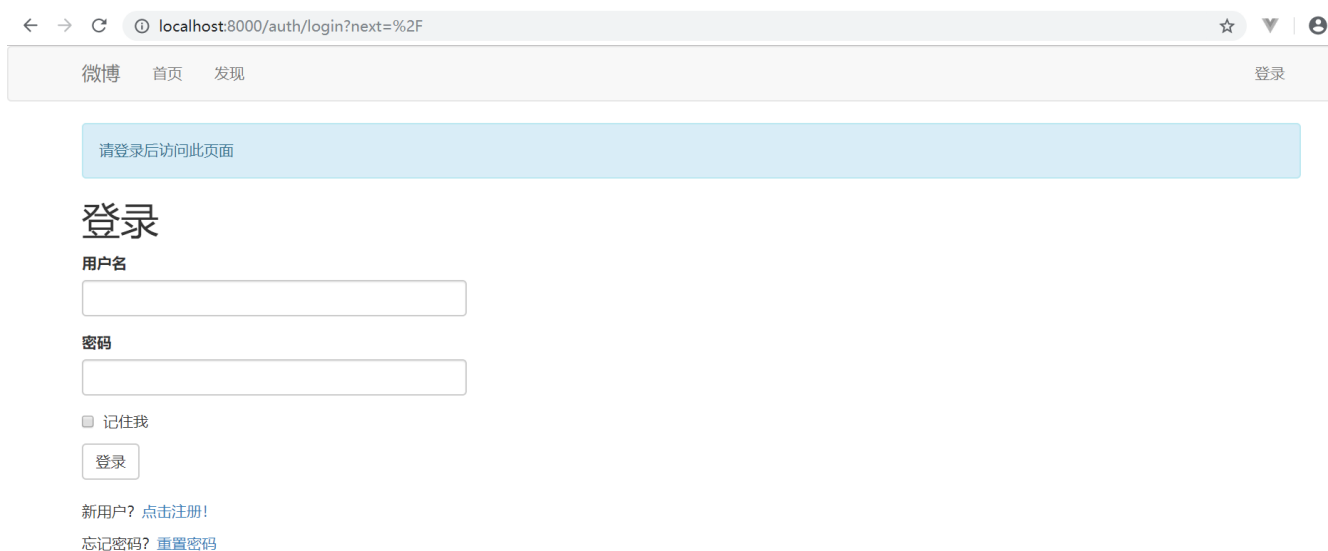


4. 启动容器

使用已创建的镜像，现在可以启动容器版本的应用程序。通过 `docker run` 命令，通常再搭配大量的参数，就可以完成容器的启动。首先要展示一个基本的例子：

```
1 $ docker run --name microblog -d -p 8000:5000 --rm superwong/microblog:latest
2 6c79aa99238ea86882ba0d9a19db157bbe4c4a58c1dcc39d082305462f60fd91
```

启动 `--name` 选项为新容器提供了一个名称。 `-d` 选项告诉 Docker 在后台运行容器。如果没有 `-d`，容器将作为前台应用程序运行，从而阻塞命令提示符。 `-p` 选项将容器端口映射到主机端口。第一个端口是主机上的端口，右边的端口是容器内的端口。上面的例子暴露了主机端口 8000，其对应容器中的端口 5000，因此即使内部容器使用 5000，也将在宿主主机上访问端口 8000 来访问应用程序。一旦容器停止，`--rm` 选项将使其自动被删除。虽然这不是必需的，但是完成或者中断的容器通常不再需要，因此可以自动删除。最后一个参数是容器使用的容器镜像名称和标签。运行上述命令后，可以在 <http://localhost:8000> 上访问该应用程序。



`docker run` 的输出是分配给新容器的ID。这是一个很长的十六进制字符串，在随后的命令中可以使用它来引用容器。实际上，只有前几个字符是必需的，足以保证ID的唯一性。

通过 `docker ps` 命令查看正在运行的容器，通过 `docker ps -a` 可以查看所有的容器。

```
1 $ docker ps
2 CONTAINER ID        IMAGE               COMMAND             CREATED
   STATUS            PORTS              NAMES
3 6c79aa99238e       superwong/microblog:latest  "./boot.sh"        3 minutes ago
   Up 3 minutes      0.0.0.0:8000->5000/tcp  microblog
```

通过 `docker stop` 命令停止容器，其中停止容器的标识就是启动时反馈的容器的ID

```
1 $ docker stop 6c79aa99238e
2 6c79aa99238e
3 $ docker ps -a
4 CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
   PORTS              NAMES
5
```

回顾一下，应用程序配置中有许多来自环境变量的选项。例如，Flask密钥，数据库URL和电子邮件服务器选项都是从环境变量中导入的。所以在通过 `docker run` 命令启动容器时，可以使用 `-e` 选项来设置。以下示例设置了密钥和gmail帐户：

```
1 $ docker run --name microblog -d -p 8000:5000 --rm -e SECRET_KEY=my-secret-key \
2   -e MAIL_SERVER=smtp.googlemail.com -e MAIL_PORT=587 -e MAIL_USE_TLS=true \
3   -e MAIL_USERNAME=<your-gmail-username> -e MAIL_PASSWORD=<your-gmail-password> \
4   superwong/microblog:latest
```

5. 使用第三方容器化服务

现在要做的是创建两个额外的容器，一个用于MySQL数据库，另一个用于Elasticsearch服务，然后加长启动Microblog容器的命令，以使其能够访问这两个新的容器。

5.1. 添加MySQL容器

像许多其他产品和服务一样，MySQL在Docker镜像仓库中提供了公共容器镜像。MariaDB数据库是MySQL的一个分支，主要由开源社区在维护，采用GPL授权许可 MariaDB的目的是完全兼容MySQL，包括API和命令行，使之能轻松成为MySQL的代替品。目前mysql的官方镜像存在bug，所以此处使用MariaDB。

```
D:\Projects\learn\flask-mega-tutorial (master -> github)
```

\$ docker search mysql				
NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
mysql	MySQL is a widely used, open-source relation...	8142	[OK]	
mariadb	MariaDB is a community-developed fork of Mys...	2771	[OK]	
mysql/mysql-server	Optimized MySQL Server Docker images. Create...	607		[OK]
percona	Percona Server is a fork of the MySQL relati...	432	[OK]	
zabbix/zabbix-server-mysql	Zabbix Server with MySQL database support	192		[OK]
hypriot/rpi-mysql	RPi-compatible Docker Image with Mysql	113		
zabbix/zabbix-web-nginx-mysql	Zabbix frontend based on Nginx web-server wi...	101		[OK]
centurylink/mysql	Image containing mysql. Optimized to be link...	60		[OK]
centos/mysql-57-centos7	MySQL 5.7 SQL database server	52		
landinternet/ubuntu-16-nginx-php-phpmyadmin-mysql-5	ubuntu-16-nginx-php-phpmyadmin-mysql-5	50		[OK]
mysql/mysql-cluster	Experimental MySQL Cluster Docker images. Cr...	44		
tutum/mysql	Base docker image to run a MySQL database se...	32		
zabbix/zabbix-web-apache-mysql	Zabbix frontend based on Apache web-server w...	29		[OK]
schickling/mysql-backup-s3	Backup MySQL to S3 (supports periodic backup...	27		[OK]
bitnami/mysql	Bitnami MySQL Docker Image	26		[OK]
zabbix/zabbix-proxy-mysql	Zabbix proxy with MySQL database support	22		[OK]
linuxserver/mysql	A Mysql container, brought to you by LinuxSe...	20		
centos/mysql-56-centos7	MySQL 5.6 SQL database server	13		
mysql/mysql-router	MySQL Router provides transparent routing be...	11		
circleci/mysql	MySQL is a widely used, open-source relation...	11		
openshift/mysql-55-centos7	DEPRECATED: A Centos7 based MySQL v5.5 image...	6		
dsteinkopf/backup-all-mysql	backup all DBs in a mysql server	6		[OK]
ansibleplaybookbundle/mysql-apb	An APB which deploys RHSCl MySQL	0		[OK]
cloudposse/mysql	Improved `mysql` service with support for `m...	0		[OK]
widdpim/mysql-client	Dockerized MySQL Client (5.7) including CurL...	0		[OK]

通过 `docker pull` 命令可以拉取远程服务器上的镜像

```
1 $ docker pull mariadb
2 Using default tag: latest
3 latest: Pulling from library/mariadb
4 f476d66f5408: Pull complete
5 8882c27f669e: Pull complete
6 d9af21273955: Pull complete
7 f5029279ec12: Pull complete
8 173c32de09a3: Pull complete
9 a680461080e8: Pull complete
10 0221175dfea0: Pull complete
11 91853d409e6e: Pull complete
12 00e9f3c4d1b0: Pull complete
13 a4f7fbf65de1: Pull complete
14 0729fd7d4e44: Pull complete
15 51b6350ed40a: Pull complete
16 8a836bb68b9f: Pull complete
17 e52b04f9b91f: Pull complete
18 Digest: sha256:47e675de3bf9c05a0ad4d5cdb0174d0da293f4027cdad35b68b6e03571ce5c84
19 Status: Downloaded newer image for mariadb:latest
```

通过 `docker run` 命令启动MySQL服务，同时需要设置数据库名称、数据库用户名、数据库用户密码等环境变量。其中，`MYSQL_RANDOM_ROOT_PASSWORD` 参数设置为 `yes`，代表数据库会为root用户设置随机密码。

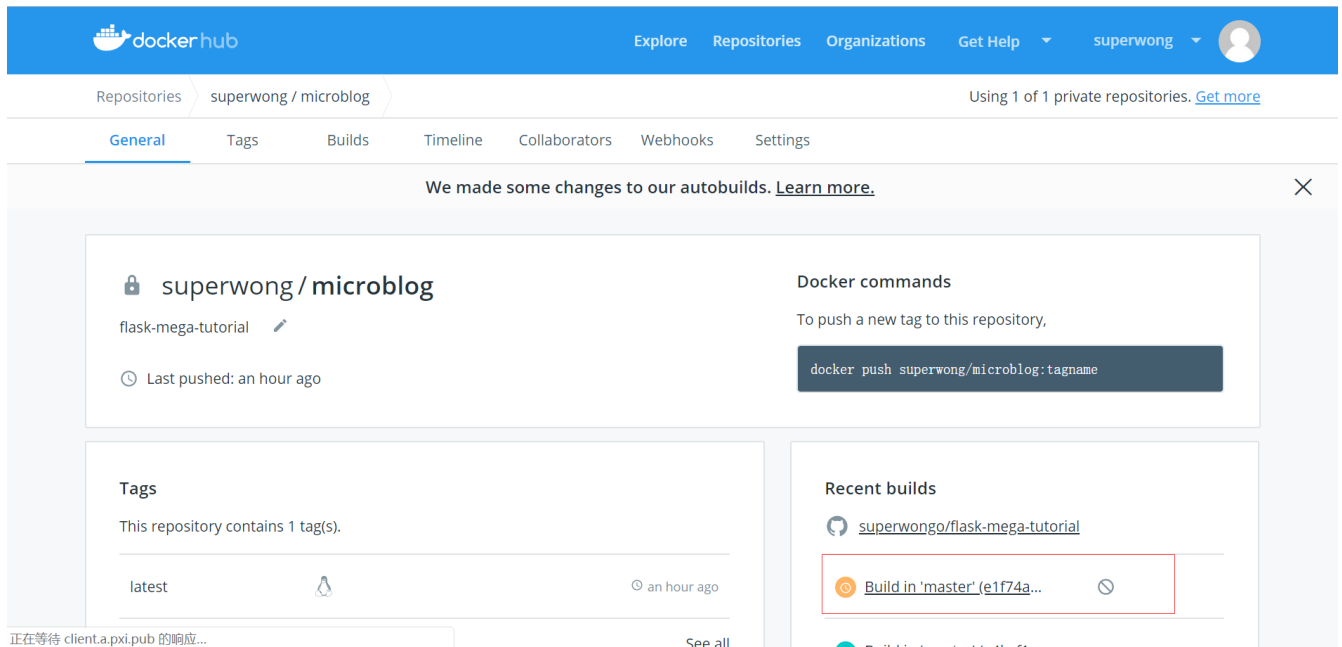
若是需要数据库的持久化，可以增加 `-v $PWD/instance/mysql:/var/lib/mysql` 命令，用于将宿主机中本地项目目录挂载到容器的对应目录中，这样即使容器被删除，本地目录下的数据库文件仍会保留。但是对于Windows系统，容易造成挂载的目录所属用户为root，导致该目录写文件失败（目前没找到解决方法）。

```
1 $ docker run --name mysql -d -p 3306:3306 -e MYSQL_RANDOM_ROOT_PASSWORD=yes \
2     -e MYSQL_DATABASE=microblog \
3     -e MYSQL_USER=microblog \
4     -e MYSQL_PASSWORD=microblog \
5     mariadb:latest \
6     --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
```

修改 `Dockerfile` 文件，添加 `pymysql` 到 `Dockerfile` 中。

```
1 RUN python -m venv venv && \
2   venv/bin/pip install --no-cache-dir -r requirements.txt -i
   https://pypi.doubanio.com/simple/ --trusted-host pypi.doubanio.com && \
3   venv/bin/pip install --no-cache-dir gunicorn pymysql -i
   https://pypi.doubanio.com/simple/ --trusted-host pypi.doubanio.com
```

提交github版本库，docker hub会根据github的变化，重新构建镜像文件。



可以从docker hub重新拉取最新的镜像，或者在本地重新构建镜像。

```
1 $ docker build -t microblog:latest .
```

再次启动Microblog容器，但是这次连接到数据库容器，以便两者都可以通过网络进行通信。

`--link` 选项告诉Docker让正要运行的容器可以访问参数中指定的容器。该参数包含由冒号分隔的两个名称。第一部分是要链接的容器的名称或ID，在本例中是在上面创建的一个名为 `mysql` 的容器。第二部分定义了一个可以在这个容器中被用来引用链接的主机名。这里使用 `dbserver` 作为代表数据库服务器的通用名称。

通过建立两个容器之间的链接，可以设置 `DATABASE_URL` 环境变量，以便SQLAlchemy被引导使用其他容器中的MySQL数据库。数据库URL将使用 `dbserver` 作为数据库主机名，`microblog` 作为数据库名称和用户，以及在启动MySQL时设置的密码。

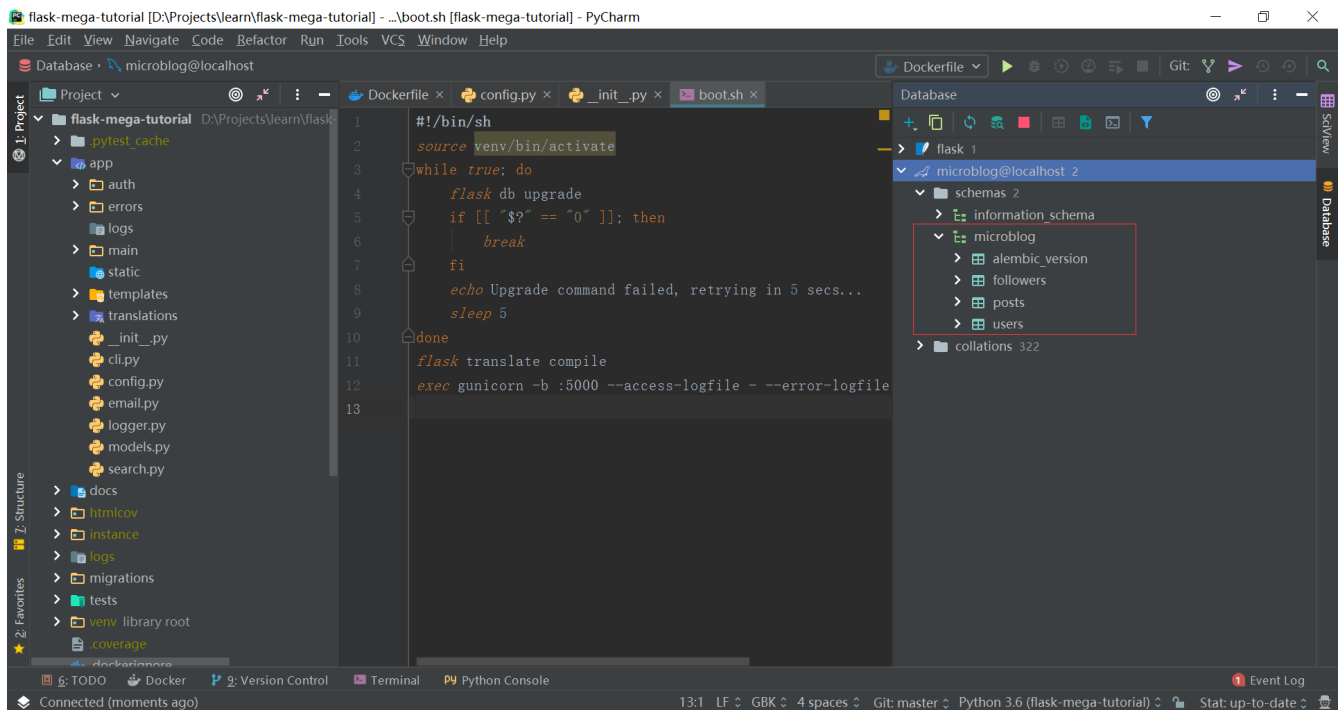
```
1 $ docker run --name microblog -d -p 8000:5000 --rm -e SECRET_KEY=my-secret-key \
2   -e MAIL_SERVER=smtp.googlemail.com -e MAIL_PORT=587 -e MAIL_USE_TLS=true \
3   -e MAIL_USERNAME=<your-gmail-username> -e MAIL_PASSWORD=<your-gmail-password> \
4   --link mysql:dbserver \
5   -e DATABASE_URL=mysql+pymysql://microblog:<database-password>@dbserver/microblog \
6   superwong/microblog:latest
```

通过 `docker logs` 命令可以查看容器的日志信息，`-f` 用于跟踪日志输出，`-t` 用于显示日志时间戳。


```

1 $ docker logs -f microblog
2 [2019-05-15 01:47:23,255] INFO in logger: 博客已启动
3 INFO [alembic.runtime.migration] Context impl MySQLImpl.
4 INFO [alembic.runtime.migration] Will assume non-transactional DDL.
5 INFO [alembic.runtime.migration] Running upgrade -> 7c2bb80cfe7b, empty message
6 INFO [alembic.runtime.migration] Running upgrade 7c2bb80cfe7b -> 7bddf580d26a, empty
  message
7 INFO [alembic.runtime.migration] Running upgrade 7bddf580d26a -> 99a18f68c13a, '新增
  about_me、last_seen字段'
8 INFO [alembic.runtime.migration] Running upgrade 99a18f68c13a -> 251e4854ef4c,
  'followers'
9 [2019-05-15 01:47:25,324] INFO in logger: 博客已启动
10 compiling catalog app/translations/en/LC_MESSAGES/messages.po to
  app/translations/en/LC_MESSAGES/messages.mo
11 [2019-05-15 01:47:26 +0000] [1] [INFO] Starting gunicorn 19.9.0
12 [2019-05-15 01:47:26 +0000] [1] [INFO] Listening at: http://0.0.0.0:5000 (1)
13 [2019-05-15 01:47:26 +0000] [1] [INFO] Using worker: sync
14 [2019-05-15 01:47:26 +0000] [32] [INFO] Booting worker with pid: 32
15 [2019-05-15 01:47:26,871] INFO in logger: 博客已启动

```



在试用MySQL容器时注意到的一件事是，这个容器需要几秒钟才能完全运行并准备好接受数据库连接。如果启动MySQL容器，然后立刻启动应用容器，在`boot.sh`脚本尝试运行`flask db migrate`时，则可能会因数据库未准备好接受连接而失败。为了解决这个问题，可以在`boot.sh`中添加一个重试循环：

```

1  #!/bin/sh
2  source venv/bin/activate
3  while true; do
4      flask db upgrade
5      if [[ "$?" == "0" ]]; then
6          break
7      fi
8      echo Upgrade command failed, retrying in 5 secs...
9      sleep 5
10 done
11 flask translate compile
12 exec gunicorn -b :5000 --access-logfile - --error-logfile - microblog:app

```

此循环检查 `flask db upgrade` 命令的退出代码，如果它不为零，则认为出现了问题，因此它会等待5秒钟然后重试。

5.2. 添加Elasticsearch容器

[Elasticsearch Docker文档](#)演示了如何将该服务作为单一节点的开发模式服务，以及部署两个节点的生产环境服务。先使用单节点模式。此处使用的镜像是由Elasticsearch服务自行维护。

```

1  $ docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300 --rm \
2      -e "discovery.type=single-node" docker.elastic.co/elasticsearch/elasticsearch:7.0.1

```

这个 `docker run` 命令与Microblog和MySQL的命令有很多相似之处，但是有一些有趣的区别。首先，有两个 `-p` 选项，这意味着这个容器将在两个端口上而不是一个端口上进行监听。端口9200和9300都映射到主机中的相同端口。

现在已经启动并运行了Elasticsearch服务，可以修改Microblog容器的启动命令以创建指向它的链接并设置Elasticsearch服务URL：

```

1  $ docker run --name microblog -d -p 8000:5000 --rm \
2      -e MAIL_SERVER=smtp.googlemail.com -e MAIL_PORT=587 -e MAIL_USE_TLS=true \
3      -e MAIL_USERNAME=<your-gmail-username> -e MAIL_PASSWORD=<your-gmail-password> \
4      --link mysql:dbserver \
5      -e DATABASE_URL=mysql+pymysql://microblog:<database-password>@dbserver/microblog \
6      --link elasticsearch:elasticsearch \
7      -e ELASTICSEARCH_URL=http://elasticsearch:9200 \
8      superwong/microblog:latest

```

localhost:8000/search?q=44

微博

首页

发现

44

个人资料

退出

搜索结果



john 说 4 分钟前:
44

← 上一页

第1页

下一页 →

6. 总结

以上已经在Docker上使用三个容器来运行了完整的应用程序，其中两个容器来自公开的第三方镜像。在较为复杂的项目中可能会用到更多的容器，因此可以使用编排工具更加方便的管理容器部署。其中，`Docker Compose`是docker提供的一个通过YAML文件来定义多容器应用的编排工具；`Kubernetes`则是除了允许以简单的YAML格式文本文件描述多容器部署逻辑以外，还提供了更高级别的自动化和便利性，负载均衡，水平扩展，密钥的安全管理以及滚动升级和回滚。