上一节课我们学习了 Scrapy 和 Scrapyd 的用法，虽然它们可以解决项目部署的一些问题，但其实这种方案并没有真正彻底解决环境配置的问题。

比如使用 Scrapyd 时我们依然需要安装对应的依赖库，即使这样仍免不了还是会出现环境冲突和不一致的问题。因此，本节课我会再介绍另一种部署方案 —— Docker。

Docker 可以提供操作系统级别的虚拟环境，一个 Docker 镜像一般都会包含一个完整的操作系统，而这些系统内也有已经配置好的开发环境，如 Python 3.6 环境等。

我们可以直接使用此 Docker 的 Python 3 镜像运行一个容器，将项目直接放到容器里运行，就不用再额外配置 Python 3 环境了，这样就解决了环境配置的问题。

我们也可以进一步将 Scrapy 项目制作成一个新的 Docker 镜像，镜像里只包含适用于本项目的 Python 环境。如果要部署到其他平台，只需要下载该镜像并运行就好了，因为 Docker 运行时采用虚拟环境，和宿主机是完全隔离的，所以也不需要担心环境冲突问题。

如果我们能够把 Scrapy 项目制作成一个 Docker 镜像，只要其他主机安装了 Docker，那么只要将镜像下载并运行即可，而不必再担心环境配置问题或版本冲突问题。

因此，利用 Docker 我们就能很好地解决环境配置、环境冲突等问题。接下来，我们就尝试把一个 Scrapy 项目制作成一个 Docker 镜像。

本节目标

我们要实现把前文 Scrapy 的入门项目打包成一个 Docker 镜像的过程。项目爬取的网址为：http://quotes.toscrape.com/，本模块 Scrapy 入门一节已经实现了 Scrapy 对此站点的爬取过程，项目代码为：https://github.com/Python3WebSpider/ScrapyTutorial，如果本地不存在的话可以 Clone 下来。

准备工作

请确保已经安装好 Docker 并可以正常运行，如果没有安装可以参考 https://cuiqingcai.com/5438.html。

创建 Dockerfile

首先在项目的根目录下新建一个 requirements.txt 文件，将整个项目依赖的 Python 环境包都列出来，如下所示：

```
scrapy
pymongo
```

如果库需要特定的版本，我们还可以指定版本号，如下所示：

```
scrapy>=1.4.0
pymongo>=3.4.0
```

在项目根目录下新建一个 Dockerfile 文件，文件不加任何后缀名，修改内容如下所示：

```
FROM python:3.7
ENV PATH /usr/local/bin:$PATH
ADD . /code
WORKDIR /code
RUN pip3 install -r requirements.txt
CMD scrapy crawl quotes
```

第一行的 FROM 代表使用的 Docker 基础镜像，在这里我们直接使用 python:3.7 的镜像，在此基础上运行 Scrapy 项目。

第二行 ENV 是环境变量设置，将 /usr/local/bin:$PATH 赋值给 PATH，即增加 /usr/local/bin 这个环境的变量路径。

第三行 ADD 是将本地的代码放置到虚拟容器中。它有两个参数：第一个参数是“.”，代表本地当前路径；第二个参数是 /code，代表虚拟容器中的路径，也就是将本地项目所有内容放置到虚拟容器的 /code 目录下，以便于在虚拟容器中运行代码。

第四行 WORKDIR 是指定工作目录，这里将刚才添加的代码路径设置成工作路径。在这个路径下的目录结构和当前本地目录结构是相同的，所以我们可以直接执行库安装命令、爬虫运行命令等。

第五行 RUN 是执行某些命令来做一些环境准备工作。由于 Docker 虚拟容器内只有 Python 3 环境，而没有所需要的 Python 库，所以我们运行此命令在虚拟容器中安装相应的 Python 库如 Scrapy，这样就可以在虚拟容器中执行 Scrapy 命令了。

第六行 CMD 是容器启动命令。在容器运行时，此命令会被执行。在这里我们直接用 scrapy crawl quotes 来启动爬虫。

修改 MongoDB 连接

接下来我们需要修改 MongoDB 的连接信息。如果我们继续用 localhost 是无法找到 MongoDB 的，因为在 Docker 虚拟容器里 localhost 实际指向容器本身的运行 IP，而容器内部并没有安装 MongoDB，所以爬虫无法连接 MongoDB。

这里的 MongoDB 地址可以有如下两种选择。

- 如果只想在本机测试，我们可以将地址修改为宿主机的 IP，也就是容器外部的本机 IP，一般是一个局域网 IP，使用 ifconfig 命令即可查看。
- 如果要部署到远程主机运行，一般 MongoDB 都是可公网访问的地址，修改为此地址即可。

但为了保证灵活性，我们可以将这个连接字符串通过环境变量传递进来，比如修改为：

```
import os

MONGO_URI = os.getenv('MONGO_URI')
MONGO_DB = os.getenv('MONGO_DB', 'tutorial')
```

这样项目的配置就完成了。

构建镜像

接下来，我们便可以构建镜像了，执行如下命令：

```
docker build -t quotes:latest .
```

这样输出就说明镜像构建成功。这时我们查看一下构建的镜像，如下所示：

```
Sending build context to Docker daemon 191.5 kB
Step 1/6 : FROM python:3.7
 ---> 968120d8cbe8
Step 2/6 : ENV PATH /usr/local/bin:$PATH
 ---> Using cache
 ---> 387abbba1189
Step 3/6 : ADD . /code
 ---> a844ee0db9c6
Removing intermediate container 4dc41779c573
Step 4/6 : WORKDIR /code
 ---> 619b2c064ae9
Removing intermediate container bcd7cd7f7337
Step 5/6 : RUN pip3 install -r requirements.txt
 ---> Running in 9452c83a12c5
...
Removing intermediate container 9452c83a12c5
Step 6/6 : CMD scrapy crawl quotes
 ---> Running in c092b5557ab8
 ---> c8101aca6e2a
Removing intermediate container c092b5557ab8
Successfully built c8101aca6e2a
```

出现类似输出就证明镜像构建成功了，这时执行，比如我们查看一下构建的镜像：

```
docker images
```

返回结果中其中有一行就是：

```
quotes  latest  41c8499ce210    2 minutes ago   769 MB
```

这就是我们新构建的镜像。

运行

我们可以先在本地测试运行，这时候我们需要指定 MongoDB 的连接信息，比如我在宿主机上启动了一个 MongoDB，找到当前宿主机的 IP 为 192.168.3.47，那么这里我就可以指定 MONGO_URI 并启动 Docker 镜像：

```
docker run -e MONGO_URI=192.168.3.47 quotes
```

当然我们还可以指定 MONGO_URI 为远程 MongoDB 连接字符串。

另外我们也可以利用 Docker-Compose 来启动，与此同时顺便也可以使用 Docker 来新建一个 MongoDB。可以在项目目录下新建 docker-compose.yaml 文件，如下所示：

```
version: '3'
services:
  crawler:
    build: .
    image: quotes
    depends_on:
      - mongo
    environment:
      MONGO_URI: mongo:7017
  mongo:
    image: mongo
    ports:
      - 7017:27017
```

这里我们使用 Docker-Compose 配置了两个容器，二者需要配合启动。

首先是 crawler 这个容器，其 build 路径是当前路径，image 代表 build 生成的镜像名称，这里取名为 quotes，depends_on 代表容器的启动依赖顺序，这里依赖于 mongo 这个容器，environment 这里就是指定容器运行时的环境变量，这里指定为 `mongo:7017` 。

另外一个容器就是刚才的 crawler 这个容器所依赖的 MongoDB 数据库了，在这里我们直接指定了镜像为 mongo，运行端口配置为 `7017:27017` ，这代表容器内的 MongoDB 运行在 27017 端口上，这个端口会映射到宿主机的 7017 端口上，所以我们在宿主机通过 7017 端口就能连接到这个 MongoDB 数据库。

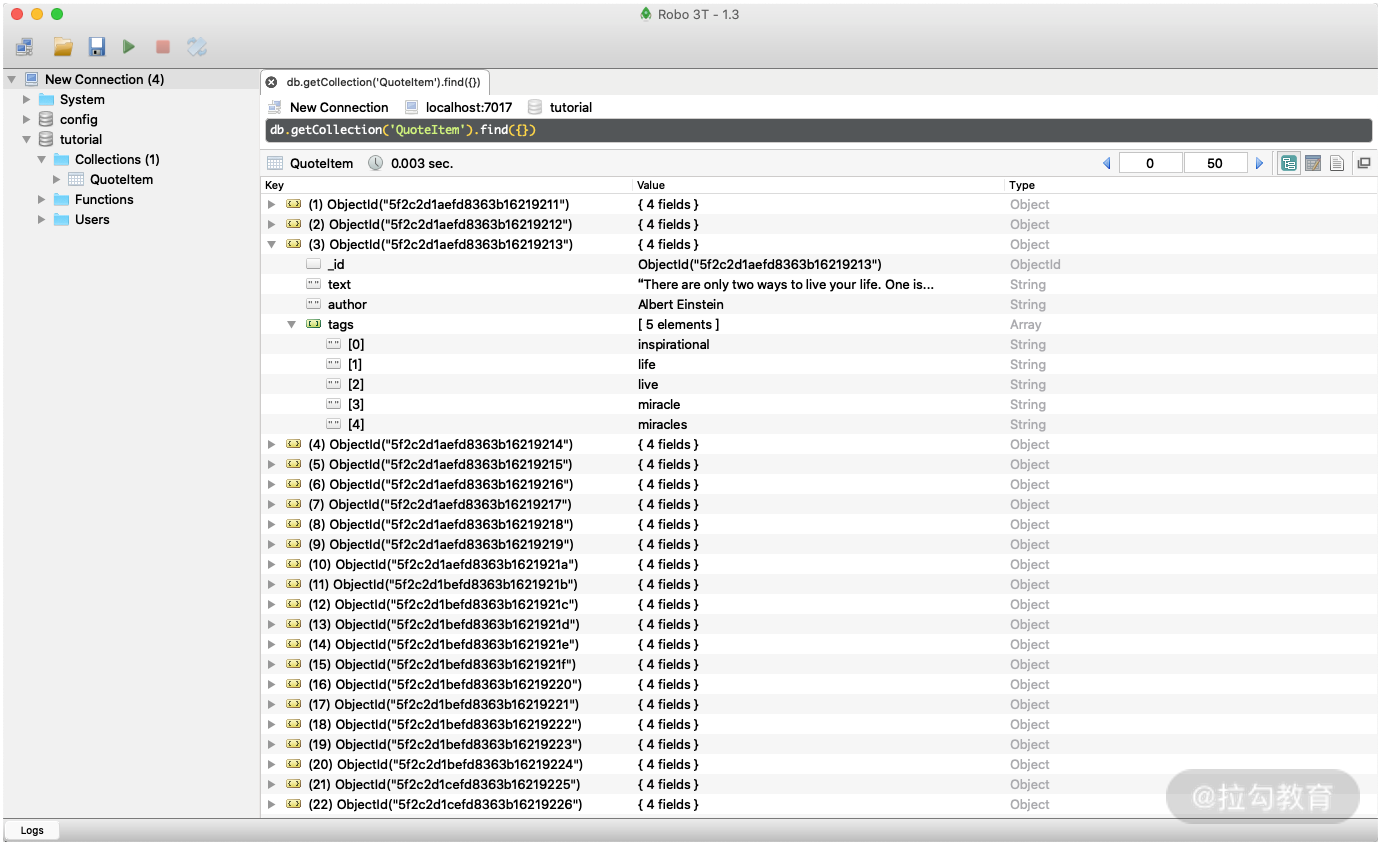
好，这时候我们运行一下：

```
docker-compose up
```

然后 Docker 会构建镜像并运行，运行结果如下：

```
Starting scrapytutorial_mongo_1 ... done
Recreating scrapytutorial_crawler_1 ... done
Attaching to scrapytutorial_mongo_1, scrapytutorial_crawler_1
mongo_1    | {"t":{"$date":"2020-08-06T16:18:05.310+00:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify
mongo_1    | {"t":{"$date":"2020-08-06T16:18:05.312+00:00"},"s":"W",  "c":"ASIO",     "id":22601,   "ctx":"main","msg":"No TransportLayer configured during NetworkInterface startup"}
mongo_1    | {"t":{"$date":"2020-08-06T16:18:05.312+00:00"},"s":"I",  "c":"NETWORK",  "id":4648601, "ctx":"main","msg":"Implicit TCP FastOpen unavailable. If TCP FastOpen is required,
...
crawler_1  | 2020-08-06 16:18:06 [scrapy.utils.log] INFO: Scrapy 2.3.0 started (bot: tutorial)
crawler_1  | 2020-08-06 16:18:06 [scrapy.utils.log] INFO: Versions: lxml 4.5.2.0, libxml2 2.9.10, cssselect 1.1.0, parsel 1.6.0, w3lib 1.22.0, Twisted 20.3.0, Python 3.7.8 (default, Ju
crawler_1  | 2020-08-06 16:18:06 [scrapy.utils.log] DEBUG: Using reactor: twisted.internet.epollreactor.EPollReactor
crawler_1  | 2020-08-06 16:18:06 [scrapy.crawler] INFO: Overridden settings:
crawler_1  | {'BOT_NAME': 'tutorial',
```

这时候就发现爬虫已经正常运行了，同时我们在宿主机上连接 localhost:7017 这个 MongoDB 服务就能看到爬取的结果了：



这就是用 Docker-Compose 启动的方式，其启动更加便捷，参数可以配置到 Docker-Compose 文件中。

推送至 Docker Hub

构建完成之后，我们可以将镜像 Push 到 Docker 镜像托管平台，如 Docker Hub 或者私有的 Docker Registry 等，这样我们就可以从远程服务器下拉镜像并运行了。

以 Docker Hub 为例，如果项目包含一些私有的连接信息（如数据库），我们最好将 Repository 设为私有或者直接放到私有的 Docker Registry 中。

首先在 https://hub.docker.com注册一个账号，新建一个 Repository，名为 quotes。比如，我的用户名为 germey，新建的 Repository 名为 quotes，那么此 Repository 的地址就可以用 germey/quotes 来表示，当然你也可以自行修改。

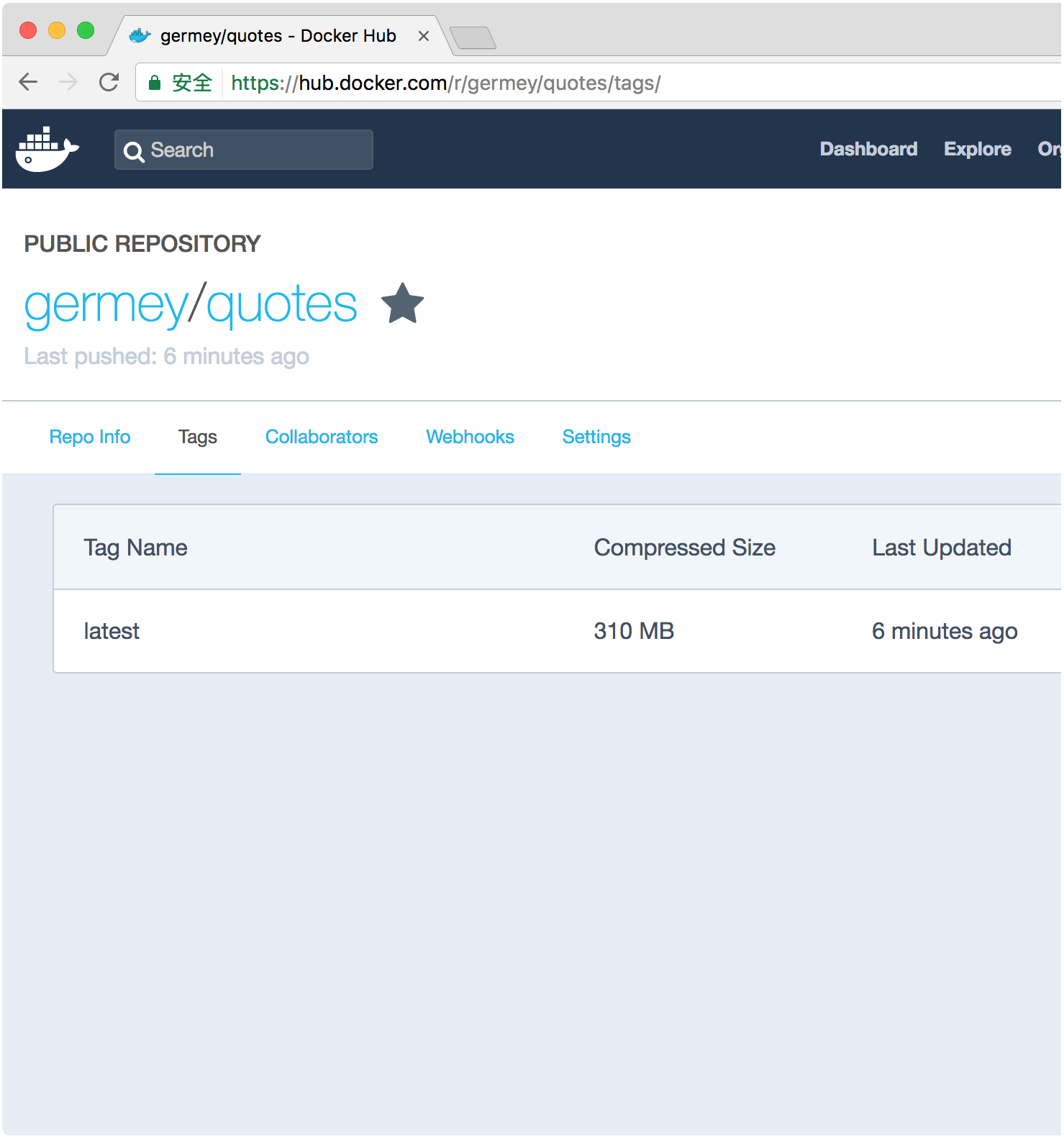
为新建的镜像打一个标签，命令如下所示：

```
docker tag quotes:latest germey/quotes:latest
```

推送镜像到 Docker Hub 即可，命令如下所示：

```
docker push germey/quotes
```

Docker Hub 中便会出现新推送的 Docker 镜像了，如图所示。



如果我们想在其他的主机上运行这个镜像，在主机上装好 Docker 后，可以直接执行如下命令：

```
docker run germey/quotes
```

这样就会自动下载镜像，然后启动容器运行，不需要配置 Python 环境，不需要关心版本冲突问题。

当然我们也可以使用 Docker-Compose 来构建镜像和推送镜像，这里我们只需要修改 docker-compose.yaml 文件即可：

```
version: '3'
services:
  crawler:
    build: .
    image: germey/quotes
    depends_on:
      - mongo
    environment:
      MONGO_URI: mongo:7017
  mongo:
    image: mongo
    ports:
      - 7017:27017
```

可以看到，这里我们就将 crawler 的 image 内容修改为了 `germey/quotes`，接下来执行：

```
docker-compose build
docker-compose push
```

就可以把镜像推送到 Docker Hub 了。

结语

本课时，我们讲解了将 Scrapy 项目制作成 Docker 镜像并部署到远程服务器运行的过程。使用此种方式，我们在本节课开始时所列出的问题都可以迎刃而解了。