在前面的第 5 小节《Docker 镜像介绍》中，我们简单构建了一个 Golang 的 http server 的 Docker 应用。在日常开发或者生产环境中，很多情况下，我们的系统都不是一个应用可以搞定的，而是由很多个部分组成，比如 webapp，数据库，缓存等。所以这一章的例子，我们就以一个 web 应用 + 缓存 redis 作为例子构建一个稍微复杂点的应用。

使用的语言和应用的版本如下：Python 3.8.1，Flask 库 1.1.1，Redis 库 3.4.1。

1. web 应用

这次我们使用 Python 来编写我们的 web 应用，上一次我们使用的是 Go 语言。由于 Go 语言部署直接使用二进制文件，Dockerfile 会极其的简单，为了让大家熟悉一下 Dockerfile 的应用，所以这里我们使用 Python 语言里编写我们的 web 应用。

Python 语言相信大家都很熟悉，不熟悉也没有关系，代码都很简单。基于 Python 的 web 网络应用框架比较出名的有 Django，Tornado，Flask 等。我们这里使用 Flask 来构建我们的应用，因为 Flask 是一种极其轻量的框架，正如作者所说：

> Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.
>
> Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy.

1.1 Flask 安装

Flask 安装很简单，和其他 Python 依赖安装基本没有区别。

1.2 Flask demo

我们前面说了 Flask 是一个非常轻量的 web 框架，那么有多轻量呢？轻量到我们使用下面几行代码就可以构建出来一个简单的 web 应用。

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, Flask'
```

启动应用。

```
$ env FLASK_APP=hello.py flask run
 * Serving Flask app "hello"
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

应用默认启动在 5000 端口，我们可以通过 `-p` 参数指定引用的启动端口。当然 Flask 还支持其他参数，我们可以通过 `flask run --help` 进行查看。

```
root@a36d1df88169:/
Usage: flask run [OPTIONS]

  Run a local development server.

  This server is for development purposes only. It does not provide the
  stability, security, or performance of production WSGI servers.

  The reloader and debugger are enabled by default if FLASK_ENV=development
  or FLASK_DEBUG=1.

Options:
  -h, --host TEXT                 The interface to bind to.
  -p, --port INTEGER              The port to bind to.
  --cert PATH                     Specify a certificate file to use HTTPS.
  --key FILE                      The key file to use when specifying a
                                  certificate.
  --reload / --no-reload          Enable or disable the reloader. By default
                                  the reloader is active if debug is enabled.
  --debugger / --no-debugger      Enable or disable the debugger. By default
                                  the debugger is active if debug is enabled.
  --eager-loading / --lazy-loader
                                  Enable or disable eager loading. By default
                                  eager loading is enabled if the reloader is
                                  disabled.
  --with-threads / --without-threads
                                  Enable or disable multithreading.
  --extra-files PATH              Extra files that trigger a reload on change.
                                  Multiple paths are separated by ':'.
  --help                          Show this message and exit.
```

应用启动了之后，我们可以访问 5000 端口来验证应用是不是正常的。

```
[root@docker ~]
Hello, Flask
```

1.3 Flask 使用

上面介绍了 Flask 最简单的使用 demo，下面我们使用 Flask 来编写我们应用和 Redis 进行交互。首先我们也要先安装 Python 依赖库：redis。

我们主要要实现三个功能：

1. redis 连接
2. 提供一个 route set 实现对 redis 中的值进行设置
3. 提供一个 route get 实现对 redis 中的值进行查询

redis 连接

redis 连接，我们直接使用 Python 的依赖库 Redis。

```
import redis

redis_client = redis.Redis(host=redis_host, port=redis_port, db=0)
```

其中连接 Redis 需要使用三个参数：

- host: redis 的 host
- port: redis 的端口
- db：redis 中的数据库，我们使用 db = 0 即可。

这里的一个核心问题是 redis 运行在另外一个 Docker 中，那我们在应用的 Docker 中如何连接 redis 实例呢？也就是如何发现 redis 的 host 和 port 呢？

在 Docker 技术中我们可以在启动 Docker 的时候指定参数 --link 将两个 Docker 的网络进行打通。在下面部署的时候我们再细说。

set route

编写一个 route，可以对 redis 进行写入。

```
@app.route('/set')
def set():
    key = request.args.get("key")
    value = request.args.get("value")
    redis_client.set(key, value)
    return 'OK. We have set ' + key + ' to be ' + value
```

其中 request.args 中可以获取到 url 中的参数。但是上面的代码没有做参数校验，key 和 value 可能是空，我们加一个参数校验的逻辑。

```
@app.route('/set')
def set():
    key = request.args.get("key")
    value = request.args.get("value")
    if key is None or value is None:
        return 'OOps, the key or value is NULL'
    redis_client.set(key, value)
    return 'OK. We have set ' + key + ' to be ' + value
```

get route

编写一个 route 对 redis 中的值查询

```
@app.route('/get')
def get():
    key = request.args.get('key')
    if key is None:
        return 'OOps, the key is null'
    value = redis_client.get(key)
    return value
```

至此，我们的 web 应用代码编写完成，完整的代码如下，其中 redis-host 现在还是一个占位符，我们部署的时候会把这个变量注入进来。

```
from flask import Flask, request
import redis

redis_client = redis.Redis(host='redis-host', port=6379, db=0)
app = Flask(__name__)

@app.route('/set')
def set():
    key = request.args.get('key')
    value = request.args.get('value')
    if key is None or value is None:
        return 'OOps, the key or value is NULL'
    redis_client.set(key, value)
    return 'OK. We have set ' + key + ' to be ' + value

@app.route('/get')
def get():
    key = request.args.get('key')
    if key is None:
        return 'OOps, the key is null'
    value = redis_client.get(key)
    return value
```

2. Dockerfile

下面开始编写我们的 Dockerfile。回忆一下我们上面编写 web 应用过程中，主要安装了依赖 flask 和 redis 依赖。我们可以很简单写出来我们的 Dockerfile 如下，并命名为 Dockerfile。

```
from python:3

RUN pip install flask
RUN pip install redis
RUN mkdir /data

COPY hello.py /data/
WORKDIR /data

EXPOSE 5000
ENV FLASK_APP=/data/hello.py
ENTRYPOINT ["flask", "run", "-h", "0.0.0.0"]
```

我们对这个 Dockerfile 进行一个简单解释：

- from：表示基础镜像是 python:3；
- RUN：表示在 docker build 的时候会执行后面的几个命令；
- COPY：拷贝文件或者目录都可以；
- WORKDIR：表示启动容器之后，当前的工作目录；
- EXPOSE：表示容器要暴露 5000 端口；
- ENV：环境变量；
- ENTRYPOINT：表示 Docker 容器的启动进程。这里 entrypoint 中的 flask run 我们增加了参数 -h 0.0.0.0。如果不加这个参数的话，进程默认绑定到 127.0.0.1，外面是没有办法访问的。

通过该 dockerfile 来构建镜像。基本每一个命令都会对应一个 step，如下。

```
[root@docker web]
Sending build context to Docker daemon  3.584kB
Step 1/8 : from python:3
 ---> efdecc2e377a
Step 2/8 : RUN pip install flask
 ---> Running in c4dfe7b3e466
Collecting flask
  Downloading Flask-1.1.1-py2.py3-none-any.whl (94 kB)
Collecting itsdangerous>=0.24
  Downloading itsdangerous-1.1.0-py2.py3-none-any.whl (16 kB)
Collecting Jinja2>=2.10.1
  Downloading Jinja2-2.11.1-py2.py3-none-any.whl (126 kB)
Collecting Werkzeug>=0.15
  Downloading Werkzeug-1.0.0-py2.py3-none-any.whl (298 kB)
  .......
Step 8/8 : ENTRYPOINT ["flask", "run"]
 ---> Running in 25594e1de72f
Removing intermediate container 25594e1de72f
 ---> d18b55e4d1fd
Successfully built d18b55e4d1fd
Successfully tagged web:v1
```

构建成功之后，我们可以通过 `docker images` 查看到我们刚才 build 出来的镜像 web。

```
[root@docker demo]
web                 v1                  02cc264143dc        6 minutes ago
943MB
```

3. 部署

我们先来部署一个 Redis Docker，`-d` 参数表示以 daemon 的方式运行。`-p` 表示端口映射。`-name` 表示 Docker 容器的名字叫 redis-test。

```
docker run --name redis-test -p 6379:6379 -d redis:latest
```

下面部署我们的 web 应用。

```
[root@docker ~]
64eef1f67c3934b6257510f47b587c59cee635188a4043b749966e71d2bc8c08
[root@docker ~]
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                    NAMES
64eef1f67c39        web:v1              "flask run -h 0.0.0.0"   2 seconds ago
    Up 1 second         0.0.0.0:5000->5000/tcp   web
```

其中有一个运行参数需要进行简单说明，也就是 --link。link 后面跟一对映射的值，左侧的为已经存在的 Docker 容器，右侧的为该容器映射到我们启动的 Docker 应用中的 host 名字，这里也就是 web 这个 Docker 容器。我们下面通过 `docker exec` 进入到容器中看一下 link 是怎么做的。

```
[root@docker ~]
root@64eef1f67c39:/data
```

我们查看一下 hosts 文件。

```
root@64eef1f67c39:/data
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.5  redis-host 0d748e8ce766 redis-test
172.17.0.6  64eef1f67c39
```

我们可以看到 redis-host 已经被写到 hosts 中，所以我们在 web 这个 Docker 容器中就可以通过 redis-host 这个主机名访问到 Redis 容器了，这也是我们的应用代码的写法。

4. 验证

部署完成，我们下面进行一个简单的验证。在宿主机上执行下面命令去设置一对 kv: <imooc, imooc.com> 写入到 Redis 中。

```
[root@docker ~]
OK. We have set imooc to be imooc.com
```

第二个请求去读取该值，如下。

```
[root@docker ~]
imooc.com
```

5. 总结

至此，我们第一个动手实践的 Docker 应用已经完成。本来想弄一个更复杂的应用，但是限于篇幅，只能做了一下取舍。虽然简单，还是建议各位同学进行动手实践。毕竟纸上得来终觉浅，绝知此事要躬行。

}