

Docker中文教程

Letong

Published
with GitBook



Table of Contents

1. [前言](#)
2. [Docker安装](#)
 - i. [Centos](#)
 - ii. [Ubuntu](#)
3. [镜像相关](#)
 - i. [获取镜像](#)
 - ii. [查看镜像](#)
 - iii. [导入和导出](#)
 - iv. [删除镜像](#)
 - v. [自定义镜像](#)
4. [容器](#)
 - i. [运行容器](#)
 - ii. [导入和导出](#)
 - iii. [终止与删除](#)
 - iv. [容器数据管理](#)
5. [Dockerfile](#)
 - i. [常用指令](#)
 - ii. [简单示例](#)
6. [网络配置](#)
 - i. [端口映射](#)
 - ii. [容器互联](#)
7. [实例大全](#)
 - i. [Supervisord管理Docker进程](#)
 - ii. [HECD架构](#)
8. [Python API操作](#)
 - i. [获取镜像](#)
 - ii. [查看镜像](#)
 - iii. [详细信息查询](#)
 - iv. [创建与移除](#)
 - v. [启动与终止](#)
 - vi. [端口映射](#)
 - vii. [使用Dockerfile](#)

Docker中文教程

本书基于docker官方文档，以及自身的一些实践经验编写而成。

#

EMAIL : a@letong.me

Blog : <http://letong.me>

If you need my help, please leave a message or send e-mail.

Docker安装

安装说明：

1. 只支持Centos 6.5以上的版本
2. Ubuntu测试环境为14.04，如果您的Ubuntu版过旧，请确保安装了AUFS。

Centos

首先安装epel https://fedoraproject.org/wiki/EPEL#How_can_I_use_these_extra_packages.3F

然后直接yum安装即可

```
yum -y install docker-io
```

Ubuntu

cat /etc/apt/sources.list.d/docker.list

```
deb https://get.docker.com/ubuntu docker main
```

直接安装

```
sudo apt-get update  
sudo apt-get install docker-io
```

获取镜像

通过pull获取镜像

```
docker pull centos
```

默认为官方源，速度较慢，可以使用国内源

```
docker pull dl.dockerpool.com:5000/centos  
docker pull docker.cn/docker/centos
```

或者修改配置文件

ubuntu中的配置文件：/etc/default/docker

Centos中的配置文件：/etc/sysconfig/docker

以下为Ubuntu中的配置文件示例

```
DOCKER_OPTS="--dns 8.8.8.8 --dns 8.8.4.4 --insecure-registry dl.dockerpool.com:5000"
```

查看镜像

显示本地已有的镜像

```
[root@Mysql ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL S
centos	latest	34943839435d	3 weeks ago	224 MB

参数注释

- REPOSITORY：仓库来源
- TAG：镜像标
- IMAGE ID：镜像ID，具有唯一性
- CREATED：镜像创建的时间
- VIRTUAL SIZE：镜像大小

有时会出现相同ID的不同TAG，不用担心那是同一镜像。

导入和导出

分别用命令 `docker save` 和 `docker load` 进行导入导出操作，以下为简单的实例。

#

导出镜像

```
docker save -o centos7.tar centos
```

#

导入本地镜像

```
sudo docker load --input centos7.tar
```

删除镜像

查看现有的镜像

```
[root@Mysql dock]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL S
centos	latest	34943839435d	3 weeks ago	224 MB



#

使用 `docker rmi` 删除镜像，可以是仓库源或者镜像ID

```
docker rmi centos
docker rmi 34943839435d
```

#

如遇无法删除，请删除依赖相关容器

```
docker rm `docker ps -aq`
```

自定义镜像

推荐使用Dockerfile，请参考相关章节。

运行容器

使用 `docker run` 运行容器

```
[root@Mysql dock]# docker run -ti centos /bin/bash  
[root@28c830652ec9 /]#
```

后台启动容器，echo命令一瞬间就结束了，只能使用ps -a查看停止的容器

```
[root@Mysql dock]# docker run -d centos /bin/echo "hhaha"  
52a3ed388019ff6a2094d1c5c8e2ae08a51a3308ffcfa3ac5b5661836ee9e627
```

使用 `docker start` 启动已经停止的容器

```
[root@Mysql dock]# docker start 28c8  
28c8  
[root@Mysql dock]# docker ps
```

以上使用的是容器的ID号，可简写

导入和导出

使用 `docker export` 导出

```
[root@Mysql dock]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
52a3ed388019	centos:latest	"/bin/echo hhaha"	3 minutes ago	Exited

```
[root@Mysql dock]# docker export 52a3 > centos7.tar
[root@Mysql dock]# ll centos7.tar
-rw-r--r--. 1 root root 232501760 12月 25 13:59 centos7.tar
```

使用 `docker import` 导入

```
[root@Mysql dock]# docker import
Usage: docker import URL|- [REPOSITORY[:TAG]]

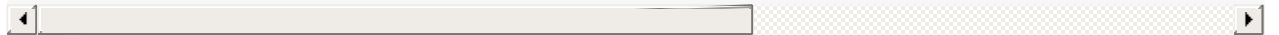
[root@Mysql dock]# cat centos7.tar | docker import - letong/centos:v1
2a32c43f599401817a5fb7b82b0a18d9898a1d3145f541b0bea65cf40611c121
[root@Mysql dock]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL
letong/centos	v1	2a32c43f5994	About a minute ago	224 MB

终止与删除容器

使用 `docker stop` 终止容器

```
[root@Mysql dock]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS
28c830652ec9        centos:latest      "/bin/bash"        25 minutes ago     Up 2 seco
[root@Mysql dock]# docker stop 28c8
28c8
```



使用 `docker rm` 删除容器

```
[root@Mysql dock]# docker rm 28c8
28c8
```

可使用 `-f` 参数强制删除容器

容器数据卷管理

使用-v参数将本地目录挂载到容器中

```
docker run -d -P --name web -v /src/webapp:/opt/webapp training/webapp python app.py
```

如果你想让该容器对数据拥有只读权

```
docker run -d -P --name web -v /src/webapp:/opt/webapp:ro training/webapp python app.py
```

#

创建一个挂在数据卷的容器 如果你想要容器之间数据共享，或者从非持久化容器中使用一些持久化数据，最好创建一个名为数据卷的容器，然后用它挂载数据。

让我们创建一个数据共享命名的容器。

```
docker run -d -v /dbdata --name dbdata training/postgres
```

你可以使用--volumes-from标识来再另外一个容器挂载/dbdata卷。

```
docker run -d --volumes-from dbdata --name db1 training/postgres
docker run -d --volumes-from dbdata --name db2 training/postgres
```

您也可以使用多个--volumes-from参数来将多个数据卷桥接到多个容器中。

也可以重复挂载

```
docker run -d --name db3 --volumes-from db1 training/postgres
```

当你删除挂载卷的dbdata容器，包括初始化数据化容器，或者随后的db1容器和db2容器，该卷将不会被删除直到没有容器使用该卷。这允许你升级，或者把有效的数据卷在容器之间迁移。

#

备份数据卷

```
docker run --volumes-from dbdata -v /backup ubuntu tar cvf /backup/backup.tar /dbdata
```

恢复数据卷，先运行一个容器，再新的容器的卷标中解压备份文件

```
docker run -v /dbdata --name dbdata2 ubuntu /bin/bash
docker run --volumes-from dbdata2 -v /backup busybox tar xvf /backup/backup.tar
```


Dockfile 指令介绍

以下为常用的Dockfile指令，可能不是那么全面。

FROM

格式为FROM 或 FROM <image>:<tag>

第一条指令必须为FROM指令。如果在同一个Dockerfile中创建多个镜像时，可以使用多个FROM指令。

MAINTAINER

格式为 MAINTAINER <name> ， 指定维护者信息。

RUN

格式为 RUN <command> 或RUN ["executable", "param1", "param2"]。

CMD

CMD ["executable","param1","param2"]使用exec执行，一个Dockfile只能有一条CMD，以最后一条为准。

EXPOSE

格式为 EXPOSE <port> ， 容器暴露的端口号。

ENV

格式为 ENV <key> <value> ， 指定一个环境变量。

ADD

格式为 ADD <src> <dest> ， src可以是URL或者tar文件。

COPY

格式为 COPY <src> <dest> ， src为本地目录的时候推荐使用

VOLUME

格式为VOLUME ["/data"], 这个就不用过多介绍了。

简单示例

以下是一段简单的sshd的示例，EXPOSE暴露的端口仅供互联系统使用。

```
# This is a base comment
FROM centos:latest
MAINTAINER letong <a@letong.me>

#yum install Package
RUN yum -y install openssh-server openssh-clients

#set sshd
RUN ssh-keygen -q -N "" -t dsa -f /etc/ssh/ssh_host_dsa_key
RUN ssh-keygen -q -N "" -t rsa -f /etc/ssh/ssh_host_rsa_key
RUN ssh-keygen -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key -N ""
RUN echo 'root:letong1' | chpasswd

#set port
EXPOSE 22

#set ENV
ENV LANG en_US.UTF-8
ENV LC_ALL en_US.UTF-8

#run sshd
CMD ["/usr/sbin/sshd", "-D"]
```

端口映射

将宿主机的2222端口映射到容器的22端口，并启动sshd服务。

```
docker run -d -p 127.0.0.1:2222:22 webserver /usr/sbin/sshd -D
```

容器互联

运行一个名web的容器，与名为db的容器互联。link格式为 name:alias

```
docker run -d -P --name web --link db:db webserver /bin/bash
```

Supervisord管理Docker进程

介绍

- 当进程中断或为启用的时候可自动重启

很简单，就直接上Dockerfile了

```
# This is a base comment
FROM centos:latest
MAINTAINER letong <a@letong.me>

#yum install Package
RUN yum -y install openssh-server openssh-clients
RUN yum -y install httpd
RUN yum -y install python-setuptools
RUN easy_install supervisor

#set sshd
RUN echo 'root:letong1' | chpasswd

#set supervisor
RUN mkdir -p /var/log/supervisor
ADD supervisord.conf /etc/supervisord.conf

#set port
EXPOSE 22
EXPOSE 80

#set ENV
ENV LANG en_US.UTF-8
ENV LC_ALL en_US.UTF-8

#run supervisor
CMD ["/usr/bin/supervisord -c /etc/supervisord.conf"]
```

supervisord的配置

```
[supervisord]
nodaemon=true

[program:sshd]
command=/usr/sbin/sshd -D

[program:httpd]
command=/usr/sbin/httpd -DFOREGROUND
```

HECD架构

容器创建步骤：

- 1.从etcd读取docker的host信息，包括appname与ip。
- 2.由confd生成haproxy配置文件，读取etcd的变量信息。
- 3.最后由Python API操作创建docker容器。

介绍

Etcd是一个高可用的 Key/Value 存储系统，主要用于分享配置和服务发现。简单：支持 curl 方式的用户API。

Confd介绍

Confd是一个轻量级的配置管理工具。通过查询Etcd，结合配置模板引擎，保持本地配置最新，同时具备定期探测机制，配置变更自动reload。

安装

宿主机安装

```
yum -y install docker-io
```

etcd安装

```
wget https://github.com/coreos/etcd/releases/download/v0.4.6/etcd-v0.4.6-linux-amd64.tar.  
tar -zxvf etcd-v0.4.6-linux-amd64.tar.gz  
cd etcd-v0.4.6-linux-amd64  
cp etcd* /bin/
```

confd与haproxy

```
yum -y install haproxy  
  
wget https://github.com/kelseyhightower/confd/releases/download/v0.6.3/confd-0.6.3-linux-  
mv confd /usr/local/bin/confd  
chmod +x /usr/local/bin/confd
```

etcd部分

启动

```
#-peer-addr：与其他节点通讯socket  
#-addr：服务监听socket  
#-data-dir：存储目录
```

```
/bin/etcd -name etcdserver -peer-addr 192.168.1.21:7001 -addr 192.168.1.21:4001 -data-dir
```

实例

```
#设置key
curl -L http://192.168.1.21:4001/v2/keys/testkey -XPUT -d value="this is testkey"
#获取key
curl -L http://192.168.1.21:4001/v2/keys/testkey
#删除key
curl -L http://192.168.1.21:4001/v2/keys/testkey -XDELETE
```

confd + haproxy部分

/etc/confd/conf.d/haproxy.toml

```
[template]
src = "haproxy.cfg.tpl"
dest = "/etc/haproxy/haproxy.cfg"
keys = [
    "/app/servers",
]
reload_cmd = "/etc/init.d/haproxy reload"
```

/etc/confd/templates/haproxy.cfg.tpl

```
global
    log 127.0.0.1 local3
    maxconn 5000
    uid 99
    gid 99
    daemon

defaults
    log 127.0.0.1 local3
    mode http
    option dontlognull
    retries 3
    option redispatch
    maxconn 2000
    timeout 5000
    clitimeout 50000
    srvtimeout 50000

listen frontend 0.0.0.0:80
    mode http
    balance roundrobin
    maxconn 2000
    option forwardfor
    {{range gets "/app/servers/*"}}
    server {{base .Key}} {{.Value}} check inter 5000 fall 1 rise 2
    {{end}}

    stats enable
    stats uri /admin-status
    stats auth admin:123456
```

```
stats admin if TRUE
```

语法见<http://golang.org/pkg/text/template/>

启动

```
#interval探测频率, node监控socket, confdir配置目录
/usr/local/bin/confd -verbose -interval 10 -node '192.168.1.21:4001' -confdir /etc/confd
```

docker启动程序实例 (by 刘天斯)

```
import docker
import etcd
import sys

Etcd_ip="192.168.1.21"
Server_ip="192.168.1.22"
App_port="80"
App_protocol="tcp"
Image="yorko/webserver:v3"

Port=""
Name=""

idict={}
rinfo={}
try:
    c = docker.Client(base_url='tcp://'+Server_ip+':2375',version='1.14',timeout=15)
except Exception,e:
    print "Connection docker server error:"+str(e)
    sys.exit()

try:
    rinfo=c.create_container(image=Image,stdin_open=True,TTY=True,command="/usr/bin/super
', '/etc/localtime'],ports=[80,22],name=None)
    containerId=rinfo['Id']
except Exception,e:
    print "Create docker container error:"+str(e)
    sys.exit()

try:
    c.start(container=containerId, binds={'/data':{'bind': '/data','ro': False},'/etc/htt
pd/conf.d','ro': True},'/etc/localtime':{'bind': '/etc/localtime','ro': True}}, port_bind
alse,dns='172.17.42.1', dns_search=None, volumes_from=None, network_mode=None,restart_pol
except Exception,e:
    print "Start docker container error:"+str(e)
    sys.exit()

try:
    idict=c.inspect_container(containerId)
    Name=idict["Name"][1:]
    skey=App_port+'/'+App_protocol
    for _key in idict["NetworkSettings"]["Ports"].keys():
        if _key==skey:
            Port=idict["NetworkSettings"]["Ports"][skey][0]["HostPort"]
except Exception,e:
    print "Get docker container inspect error:"+str(e)
```



```
sys.exit()

if Name!="" and Port!="":
    try:
        client = etcd.Client(host=Etcd_ip, port=4001)
        client.write('/app/servers/'+Name, Server_ip+": "+str(Port))
        print Name+" container run success!"
    except Exception,e:
        print "set etcd key error:"+str(e)
else:
    print "Get container name or port error."
```



docker-py 文档

简易安装

```
pip install docker-py
```

完整的文档在 /docs/ 目录

GitHub项目链接：

<https://github.com/docker/docker-py>

启动远程socket

Centos

```
OPTIONS=--selinux-enabled -H fd:// -H tcp://0.0.0.0:2375
```

Ubuntu

```
DOCKER_OPTS="-H tcp://0.0.0.0:2375 -H unix:///var/run/docker.sock"
```

#

example

```
In [1]: from docker import Client
In [2]: c = Client(base_url='tcp://192.168.1.104:2375')
In [3]: c.info()
Out[3]:
{'Containers': 6,
 'Debug': 0,
 'Driver': u'devicemapper',
 'DriverStatus': [[u'Pool Name', u'docker-253:1-8812760-pool'],
 [u'Pool Blocksize', u'65.54 kB'],
 [u'Data file', u'/var/lib/docker/devicemapper/devicemapper/data'],
 [u'Metadata file', u'/var/lib/docker/devicemapper/devicemapper/metadata'],
 [u'Data Space Used', u'1.219 GB'],
 [u'Data Space Total', u'107.4 GB'],
 [u'Metadata Space Used', u'2.175 MB'],
 [u'Metadata Space Total', u'2.147 GB'],
 [u'Library Version', u'1.02.84-RHEL7 (2014-03-26)']],
 'ExecutionDriver': u'native-0.2',
 'IPV4Forwarding': 1,
 'Images': 18,
 'IndexServerAddress': u'https://index.docker.io/v1/',
 'InitPath': u'/usr/libexec/docker/dockerinit',
 'InitSha1': u'c906504aa058139c1d0569ecd0aa5f462a73440f',
 'KernelVersion': u'3.10.0-123.el7.x86_64',
 'MemoryLimit': 1,
 'NEventsListener': 0,
 'NFD': 12,
 'NGoroutines': 12,
 'OperatingSystem': u'CentOS Linux 7 (Core)',
 'SwapLimit': 1}
```


获取镜像

相当于 `docker pull`

Params:

- repository (str): 仓库
- tag (str): tag
- stream (bool): Stream the output as a generator
- insecure_registry (bool): Use an insecure registry

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.pull(repository='centos', tag='centos6')
```

查看镜像

相当于 `docker images`

Params:

- name (str): 匹配name
- quiet (bool): 只显示ID, 返回列表
- all (bool): 显示所有镜像
- viz: Depreciated

#

实例

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.images(name='centos')
[{'Created': 1417557342,
  'Id': 'u'34943839435dfb2ee646b692eebb06af13823a680ace00c0adc232c437c4f90c',
  'ParentId': 'u'5b12ef8fd57065237a6833039acc0e7f68e363c15d8abb5cacce7143a1f7de8a',
  'RepoTags': ['centos:latest'],
  'Size': 224015991,
  'VirtualSize': 224015991}]
```

详细信息查询

查看系统信息

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.info()
{'Containers': 1,
 u'Debug': 0,
 u'Driver': u'devicemapper',
 u'DriverStatus': [[u'Pool Name', u'docker-253:1-8812760-pool'],
 [u'Pool Blocksize', u'65.54 kB'],
 [u'Data file', u'/var/lib/docker/devicemapper/devicemapper/data'],
 [u'Metadata file', u'/var/lib/docker/devicemapper/devicemapper/metadata'],
 [u'Data Space Used', u'1.209 GB'],
 [u'Data Space Total', u'107.4 GB'],
 .....]
```

查看镜像底层信息

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.inspect_image(image_id='2a32')
{'Architecture': u'amd64',
 u'Author': u'',
 u'Comment': u'Imported from -',
 u'Config': None,
 u'Container': u'',
 u'ContainerConfig': {'AttachStderr': False,
 u'AttachStdin': False,
 u'AttachStdout': False,
 u'Cmd': None,
 u'CpuShares': 0,
 u'Cpuset': u'',
 .....}
```

查看容器底层信息

```
c.inspect_container(container='38aa')
```

容器的创建与移除

创建容器

相当与 `docker run`

Params:

- image (str): 要运行的镜像
- command (str or list): 容器运行的命令
- hostname (str): 容器的主机名
- user (str or int): 用户名或UID
- detach (bool): 分离模式：在后台运行的容器，并打印新的容器ID
- stdin_open (bool): 保持标准输入
- tty (bool): 分配一个tty
- mem_limit (float or str): 内存限制 (format: [number][optional unit], where unit = b, k, m, or g)
- ports (list of ints): 端口号列表
- environment (dict or list): A dictionary or a list of strings in the following format ["PASSWORD=xxx"] or {"PASSWORD": "xxx"}.
- dns (list): dns服务器
- volumes (str or list): volumes=['/data']
- volumes_from (str or list): 容器名称或ID的列表，从中获取卷。可选一个字符串用逗号连接容器的id
- network_disabled (bool): 禁止联网
- name (str): 容器名
- entrypoint (str or list): 入口
- cpu_shares (int or float): CPU 权重
- working_dir (str): 工作目录路径
- domainname (str or list): 设置自定义DNS搜索域
- memswap_limit: 内容swap限制

实例

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.create_container(image='centos',command='echo "examle container"')
{'u'Id': u'30b51797743b7b8dd029899bf527797384864fdb3367c18c18748250017f8888',
 u'Warnings': None}
```

查看返回结果

[root@Mysql ~]# docker ps -a				
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
30b51797743b	centos:latest	"echo 'examle contai	27 seconds ago	silly

扩展实例

```
c.create_container(image="lelong/webserver",stdin_open=True,tty=True,command="/usr/bin/su
```

数据绑定实例

```
c.start(container_id, binds={
    '/home/user1/':
        {
            'bind': '/mnt/vol2',
            'ro': False
        },
    '/var/www':
        {
            'bind': '/mnt/vol1',
            'ro': True
        }
})
```

移除容器

remove_container

相当于 `docker rm`

Params:

- container (str): 容器ID
- v (bool): 删除与该容器相关联的卷
- link (bool): 删除指定的链接，而不是底层的容器
- force (bool): 强制删除容器

#

实例

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.remove_container(container='30b5')
```

移除镜像

remove_image

相当于 `docker rmi`

Params:

- image (str): 要移除的镜像
- force (bool): 强制移除镜像
- noprun (bool): 不删除未标记的

#

实例

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.remove_image(image='centos')
```

启动与终止

启动容器

Params:

- container (str): 启动的容器
- binds: 数据挂载绑定
- port_bindings (dict): 端口绑定
- lxc_conf (dict): LXC配置
- publish_all_ports (bool): 发布所有端口的主机
- links (dict or list of tuples): See note above
- privileged (bool): 给扩展权限到这个容器
- dns (list): 设置自定义DNS服务器
- dns_search (list): DNS搜索域
- volumes_from (str or list): 容器名称或ID的列表，从中获取卷。可选一个字符串用逗号连接容器的id
- network_mode (str): One of ['bridge', None, 'container:', 'host']
- restart_policy (dict): See note above. "Name" param must be one of ['on-failure', 'always']

#

实例

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> container = cli.create_container(image='busybox:latest', command='/bin/sleep 30')
>>> c.start(container=container.get('Id'))
```

停止容器

Params:

- container (str): 要移除的容器
- timeout (int): 超时时间，否则强制停止

#

实例

```
>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.stop(container='8290')
```

端口映射

Params:

- container (str): The container to look up
- private_port (int): The private port to inspect
- Returns (list of dict): The mapping for the host ports

#

实例

以下两个操作同等效果

```
docker run -d -p 80:80 centos /bin/sleep 30
7174d6347063a83f412fad6124c99cffd25ffe1a0807eb4b7f9cec76ac8cb43b

>>> from docker import Client
>>> c = Client(base_url='tcp://192.168.1.104:2375')
>>> c.port('7174d6347063', 80)
[{'HostIp': '0.0.0.0', 'HostPort': '80'}]
```

使用Dockerfile

build格式

build(self, path=None, tag=None, quiet=False, fileobj=None, nocache=False, rm=False, stream=False, timeout=None, custom_context=False, encoding=None)

Params:

- path (str): Path to the directory containing the Dockerfile
- tag (str): A tag to add to the final image
- quiet (bool): Whether to return the status
- fileobj: A file object to use as the Dockerfile. (Or a file-like object)
- nocache (bool): Don't use the cache when set to True
- rm (bool): Remove intermediate containers
- stream (bool): Return a blocking generator you can iterate over to retrieve build output as it happens
- timeout (int): HTTP timeout
- custom_context (bool): Optional if using fileobj
- encoding (str): The encoding for a stream. Set to gzip for compressing

实例

```
>>> from io import BytesIO
>>> from docker import Client
>>> dockerfile = '''
... # Shared Volume
... FROM centos:latest
... MAINTAINER letong <a@letong.me>
... VOLUME /data
... CMD ["/bin/sh"]
... '''
>>> f = BytesIO(dockerfile.encode('utf-8'))
>>> c = Client(base_url='tcp://127.0.0.1:2375')
>>> c.build(path=/dockerfile, tag='letong/webserver')
```