

Docker 容器使用

Docker 客户端

docker 客户端非常简单,我们可以直接输入 docker 命令来查看到 Docker 客户端的所有命令选项。

```
runoob@runoob:~# docker
```

```
runoob@runoob:/root$ docker
Usage: docker [OPTIONS] COMMAND [arg...]
       docker daemon [ --help | ... ]
       docker [ --help | -v | --version ]

A self-sufficient runtime for containers.

Options:
  --config=~/.docker          Location of client config files
  -D, --debug                 Enable debug mode
  -H, --host=[]              Daemon socket(s) to connect to
  -h, --help                  Print usage
  -l, --log-level=info       Set the logging level
  --tls                       Use TLS; implied by --tlsverify
  --tlscacert=~/.docker/ca.pem Trust certs signed only by this CA
  --tlscert=~/.docker/cert.pem Path to TLS certificate file
  --tlskey=~/.docker/key.pem  Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
  -v, --version               Print version information and quit

Commands:
  attach      Attach to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes on a container's filesystem
```

可以通过命令 **docker command --help** 更深入的了解指定的 Docker 命令使用方法。

例如我们要查看 **docker stats** 指令的具体使用方法：

```
runoob@runoob:~# docker stats --help
```

```
runoob@runoob:/root$ docker stats --help
Usage:  docker stats [OPTIONS] [CONTAINER...]

Display a live stream of container(s) resource usage statistics

  -a, --all          Show all containers (default shows just running)
  --help            Print usage
  --no-stream        Disable streaming stats and only pull the first result
```

运行一个web应用

前面我们运行的容器并没有一些什么特别的用处。

接下来让我们尝试使用 docker 构建一个 web 应用程序。

我们将在docker容器中运行一个 Python Flask 应用来运行一个web应用。

```
runoob@runoob:~# docker pull training/webapp # 载入镜像
runoob@runoob:~# docker run -d -P training/webapp python app.py
```

```
runoob@runoob:~$ docker run -d -P training/webapp python app.py
4ecb9ce5113ded09117a94462fee760c89f9db7bfc10365ca20f2d5a4430871b
runoob@runoob:~$
```

参数说明:

- **-d**:让容器在后台运行。
- **-P**:将容器内部使用的网络端口映射到我们使用的主机上。

查看 WEB 应用容器

使用 `docker ps` 来查看我们正在运行的容器：

```
runoob@runoob:~# docker ps
```

CONTAINER ID	IMAGE	COMMAND	...	PORTS
d3d5e39ed9d3	training/webapp	"python app.py"	...	0.0.0.0:32769->5000/tcp

这里多了端口信息。

```
PORTS
0.0.0.0:32769->5000/tcp
```

Docker 开放了 5000 端口（默认 Python Flask 端口）映射到主机端口 32769 上。

这时我们可以通过浏览器访问WEB应用

← → ↻ 192.168.239.130:32769

Hello world!

我们也可以通过 `-p` 参数来设置不一样的端口：

```
runoob@runoob:~$ docker run -d -p 5000:5000 training/webapp python app.py
```

docker ps查看正在运行的容器

```
runoob@runoob:~# docker ps
```

CONTAINER ID	IMAGE	...	PORTS	NAMES
bf08b7f2cd89	training/webapp	...	0.0.0.0:5000->5000/tcp	wizardly_chandrasekhar
d3d5e39ed9d3	training/webapp	...	0.0.0.0:32769->5000/tcp	xenodochial_hoov

容器内部的 5000 端口映射到我们本地主机的 5000 端口上。

网络端口的快捷方式

通过 **docker ps** 命令可以查看到容器的端口映射，**docker** 还提供了另一个快捷方式 **docker port**，使用 **docker port** 可以查看指定（ID 或者名字）容器的某个确定端口映射到宿主机的端口号。

上面我们创建的 web 应用容器 ID 为 **bf08b7f2cd89** 名字为 **wizardly_chandrasekhar**。

我可以使⤵用 **docker port bf08b7f2cd89** 或 **docker port wizardly_chandrasekhar** 来查看容器端口的映射情况。

```
runoob@runoob:~$ docker port bf08b7f2cd89
5000/tcp -> 0.0.0.0:5000
```

```
runoob@runoob:~$ docker port wizardly_chandrasekhar
5000/tcp -> 0.0.0.0:5000
```

查看 WEB 应用程序日志

docker logs [ID或者名字] 可以查看容器内部的标准输出。

```
runoob@runoob:~$ docker logs -f bf08b7f2cd89
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
192.168.239.1 - - [09/May/2016 16:30:37] "GET / HTTP/1.1" 200 -
192.168.239.1 - - [09/May/2016 16:30:37] "GET /favicon.ico HTTP/1.1" 404 -
```

-f: 让 **docker logs** 像使用 **tail -f** 一样来输出容器内部的标准输出。

从上面，我们可以看到应用程序使用的是 5000 端口并且能够查看到应用程序的访问日志。

查看WEB应用程序容器的进程

我们还可以使用 **docker top** 来查看容器内部运行的进程

```
runoob@runoob:~$ docker top wizardly_chandrasekhar
```

UID	PID	PPID	...	TIME	CMD
root	23245	23228	...	00:00:00	python app.py

检查 WEB 应用程序

使用 **docker inspect** 来查看 Docker 的底层信息。它会返回一个 JSON 文件记录着 Docker 容器的配置和状态信息。

```
runoob@runoob:~$ docker inspect wizardly_chandrasekhar
[
  {
    "Id": "bf08b7f2cd897b5964943134aa6d373e355c286db9b9885b1f60b6e8f82b2b85",
    "Created": "2018-09-17T01:41:26.174228707Z",
```

```
"Path": "python",
"Args": [
    "app.py"
],
"State": {
    "Status": "running",
    "Running": true,
    "Paused": false,
    "Restarting": false,
    "OOMKilled": false,
    "Dead": false,
    "Pid": 23245,
    "ExitCode": 0,
    "Error": "",
    "StartedAt": "2018-09-17T01:41:26.494185806Z",
    "FinishedAt": "0001-01-01T00:00:00Z"
},
.....
```

停止 WEB 应用容器

```
runoob@runoob:~$ docker stop wizardly_chandrasekhar
wizardly_chandrasekhar
```

重启WEB应用容器

已经停止的容器，我们可以使用命令 `docker start` 来启动。

```
runoob@runoob:~$ docker start wizardly_chandrasekhar
wizardly_chandrasekhar
```

`docker ps -l` 查询最后一次创建的容器：

```
# docker ps -l
```

CONTAINER ID	IMAGE	PORTS	NAMES
bf08b7f2cd89	training/webapp	0.0.0.0:5000->5000/tcp	wizardly_chandrasekhar

正在运行的容器，我们可以使用 `docker restart` 命令来重启

移除WEB应用容器

我们可以使用 `docker rm` 命令来删除不需要的容器

```
runoob@runoob:~$ docker rm wizardly_chandrasekhar
wizardly_chandrasekhar
```

删除容器时，容器必须是停止状态，否则会报如下错误

```
runoob@runoob:~$ docker rm wizardly_chandrasekhar
Error response from daemon: You cannot remove a running container bf08b7f2cd897b5964943134aa6d373e355c286db9b9885b1f60b6e8f82b2b85. Stop the container before attempting removal or force remove
```

[← Ubuntu Docker 安装](#)

[Docker 镜像使用 →](#)

[📝 点我分享笔记](#)