

Docker 的基本使...

这是本专栏的第一部分：Docker 入门，共 3 篇，带大家进入 Docker 的世界。上一节，我带大家了解了 Docker 和容器技术的发展历程，知道了它的基础技术发展路线。下面一起进入第二节的内容。

Docker 安装

Docker 支持 Linux、MacOS 和 Windows 等系统，且在 Linux 的各发行版中的安装步骤也都略有差异。

这里我不会列出它在各个系统平台上的具体安装步骤，因为 [Docker 的文档描述的很详细了](#)，没必要赘述。

这里我对在 Linux 平台下的安装多说一点，如果你使用的是比较常见的发行版，如 Debian、Ubuntu、CentOS、Fedora 等，可以直接通过 <https://get.docker.com/> 中提供的脚本来一键完成安装。

```
# 下载脚本
```

```
$ curl -fsSL https://get.docker.com -o get-docker.sh
```

[复制](#)

脚本中内置了使用国内源进行加速：

```
# 使用 Azure 进行加速
```

```
$ sh get-docker.sh --mirror AzureChinaCloud
```

[复制](#)

或

```
# 使用 Aliyun 进行加速
```

```
$ sh get-docker.sh --mirror Aliyun
```

[复制](#)

在安装完成后，强烈建议阅读[官方文档](#)，对已经安装的 Docker 进行配置，比如配置 Docker 的开机自启动。

第一个容器

在安装完成后，我们正式开始。

经过上一节的介绍，我们也知道 Docker 一开始能胜出，而且吸引无数开发者，与它的易用性是密不可分的。

使用 Docker 启动容器很简单，只需要一句 `docker run` 命令行即可搞定。

例如，当我想要运行最新版本的 Python 解释器时，只需要一句 `docker run -it python` 即可。

复制

```
(MoeLove) → ~ docker run -it python
Unable to find image 'python:latest' locally
latest: Pulling from library/python
4ae16bd47783: Pull complete
bbab4ec87ac4: Pull complete
2ealf7804402: Pull complete
96465440c208: Pull complete
6ac892e64b94: Pull complete
5b3ec9e84adf: Pull complete
317202007d7c: Pull complete
balee226143f: Pull complete
cba5f4ed3782: Pull complete
Digest: sha256:4432d65bd7da4693bb9365c3369ed5c7f0cb472195a294c54d1b766751098f7b
Status: Downloaded newer image for python:latest
Python 3.7.4 (default, Aug 14 2019, 12:09:51)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.version
'3.7.4 (default, Aug 14 2019, 12:09:51) \n[GCC 8.3.0]'
```

看吧真的很简单，现在已经运行了一个容器，并在容器中完成了操作。

你可能已经注意到了，我们在 `docker run` 命令与 `python` 镜像之间加了 `-it` 的参数。

这是一种简写，实际上这条命令的完整写法是 `docker run --interactive --tty python`。

其中：

- `--interactive` 就如同它字面上的意思，使用此选项表示会保持标准输入（STDIN）打开，从标准输入来接收输入信号；
- `--tty` 参数则表示分配一个伪终端给容器。

这两个选项常常一起使用，用于与容器内程序直接进行交互。

理解了上述内容，我们的第一个容器实验也就顺利结束了。

后台运行容器

我们并不总是希望一直保持与容器进行交互，如 Redis、NGINX 之类的这种可提供服务的容器，在多数情况下只是希望它可以运行在后台提供服务即可。

经过前面内容的介绍，也许会有人很自然的认为，既然增加 `--interactive` 和 `--tty` 参数可以让我们直接与容器进行交互，那如果去掉这两个参数，是否就可以保证容器不与我们交互，运行在后台呢？不妨来试试看：

复制

```
(MoeLove) → ~ docker run python
```

```
(MoeLove) → ~ docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	ST
7a803976d0a4	python	"python3"	10 seconds ago	Ex

执行完 `docker run python` 后没有任何输出或反馈，通过 `docker ps` 来查看容器的状态。其中，`-l` 参数是 `--latest` 的缩写，表示最新的容器（包含所有状态），可以看到该容器的状态是已经退出了。

这是为什么呢？

回忆下刚才带着 `--interactive` 和 `--tty` 的时候，启动容器后，直接进入了 Python 的可交互式终端内。而现在我们没有携带任何参数，那自然在启动时 Python 的终端知道即使等待也不会有任何结果，因此就退出了。

那我们有没有办法改变这一情况呢？

有，我们给命令的最后增加一些参数来解决。

复制

```
(MoeLove) → ~ docker run python sleep 60
```

加了一句 `sleep 60`，现在整个终端没有任何输出，也无法进行交互（毕竟我们没有传递过 `--interactive` 的参数，因此输入是无效的）。另外打开一个终端，执行刚才的 `docker ps -l` 命令进行查看。

复制

```
(MoeLove) → ~ docker ps -l
```

CONTAINER ID	IMAGE	COMMAND	CREATED	ST
82686b276ec4	python	"sleep 60"	6 seconds ago	Up

可以看到该容器是 Up 的状态，且正在执行 `sleep 60` 的命令，这说明我们在最后传递的命令是可执行的。

现在容器的行为并不符合我们的预期，那如何实现预期呢？答案是可以给 `docker run` 命令传递 `-d` 或者完整的 `--detach` 参数，这表示将容器运行于后台。来试试看。

复制

```
(MoeLove) → ~ docker run -d python sleep 60
```

```
9f2b81e85893b1f8402247867344c9ab6bde92f377ec9949bd491e857b570048
```

该命令执行后，输出了一行字符串，终端并没有被占用，来执行 `docker ps -l` 命令：

复制

```
(MoeLove) → ~ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             ST
9f2b81e85893        python             "sleep 60"         5 seconds ago      Up
```

可以看到容器也正在运行，符合我们预期的“后台运行容器”。

小结

使用 `docker run 镜像名 命令` 可以启动一个容器，在执行该命令时组合使用 `--interactive` 和 `--tty` 可直接与容器内应用程序进行交互。

容器启动时，在镜像名之后写命令，可传递至我们实验的容器内（具体原因会放在下一个部分的章节讲）。

同样地，当我们启动容器时，传递了 `--detach` 参数，并且容器内执行的程序无需等待交互，则容器可以启动在后台。

进入容器

刚才我们的容器已经启动在了后台，如果此时想要在容器内执行一条命令，或者想要运行 Python 的解释器该如何操作呢？

答案是用 `docker exec`：

复制

```
(MoeLove) → ~ docker run -d python sleep 60
10aad6e0af4fad2405c420a90fbf56f9689f033608e6f22d987c2f18d644eda9
(MoeLove) → ~ docker exec -it 10aad6e0af4fad2405c420a90fbf56f9689f033608e6f22d987c2f18d644eda9
Python 3.7.4 (default, Aug 14 2019, 12:09:51)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import sys
>>> sys.version
'3.7.4 (default, Aug 14 2019, 12:09:51) \n[GCC 8.3.0]'
```

如上所述，`-it` 仍然是 `--interactive` 和 `--tty` 的缩写，最后的 `python` 是我们预期要执行的命令，当然换成 `bash` 或者其他命令也可以。

```
(MoeLove) → ~ docker exec -it 10aad6e0af4fad2405c420a90fbf56f9689f033608e6f22d98
root@f396422ae58d:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv
```

总结

通过本节，我们介绍了使用 `docker run 镜像` 来启动一个容器，可以给此命令传递 `--interactive` 和 `--tty`（简写组合为 `-it`）参数，以达到直接与容器内应用程序直接交互的目的。

另外也可以在 `docker run 镜像` 之后传递命令和参数，以改变使用镜像启动容器后的默认行为（由于本篇是 Docker 入门的内容，为了让读者更易理解，此处的表述不是很严谨，在下一部分的章节中会进行补充）。

比如，我们可以传递 `sleep 60` 让上面例子中的 Python 容器启动后休眠 60s；配合着给 `docker run` 传递的 `--detach` 参数，可以实现将容器启动在后台的目的。

除了以上内容，还介绍了使用 `docker ps` 可列出容器记录，通过给它传递 `-l` 参数可得到最近的一条记录。

如果是一个正在运行的容器，可以通过 `docker exec -it 容器 ID/名称 命令` 的方式进入该容器内。

以上便是本节的全部内容，你可能会问，这就是“Docker 入门”吗？事实上，是的。

Docker 一直以易用性著称，且 Docker 也一直很注意用户体验，从 Docker 的首次面世到现在，一直都是用 `docker run 镜像` 这样简单的一句命令即可启动容器。

我认为，在你学习这个专栏的时候，以上内容便是“入门”的基础知识，后续章节中我会假设你已经知晓这些。

Docker 的其他知识，后续章节中会通过实践加深入原理的方式，逐层递进，带你掌握 Docker 的核心知识！