

# 架构师分享 Docker 新手入门完全指南

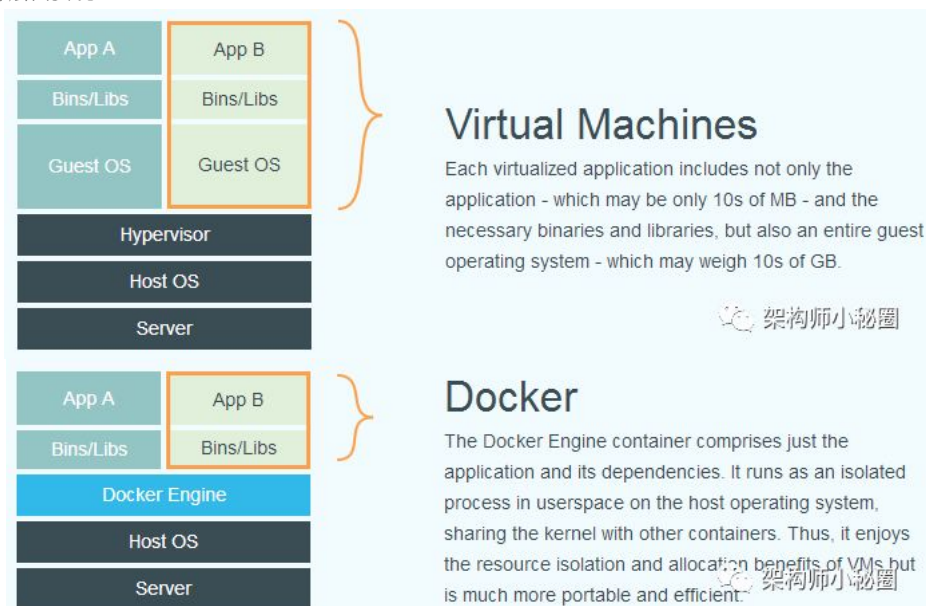
每日一个Linux、Python干货 ▲ 关注的人都加薪了

来源：架构师小秘圈

ID: seexmq

- Docker 最初 dotCloud 公司内部的一个业余项目
- Docker 基于 Go 语言
- Docker 项目的目标是实现轻量级的操作系统虚拟化解决方案
- Docker 的基础是 Linux 容器 (LXC) 等技术
- Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多
- Docker 对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器

下面的图片比较了 Docker 和传统虚拟化方式的不同之处，可见容器是在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，而传统方式则是在硬件层面实现。



容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行 10 个不同的应用就要起 10 个虚拟机，而 Docker 只需要启动 10 个隔离的应用即可。

主要优势为：

- 更快速的交付和部署 - 容器成为了最小单位
- 更高效的虚拟化 - 内核级虚拟化
- 更轻松的迁移和拓展
- 更简单的管理

官方网站提供了 Mac, Linux 和 Windows 版本的安装教程。我们只要跟着官方文档即可，这里不再赘述。

不过需要提一下 Kitematic 这个图形化工具（官方给出的定义是 Visual Docker Container Management on Mac & Windows），对于熟悉和了解 Docker 是很好的帮助，大家可以体验一下。

## 守护进程

运行 Docker 守护进程时，可以用 `-H` 来改变绑定接口的方式，比如

```
sudo /usr/bin/docker -d -H tcp://0.0.0.0:2375,
```

如果不想每次都输入这么长的命令，需要加入以下环境变量

```
export DOCKER_HOST="tcp://0.0.0.0:2375"
```

## 图形用户界面

虽然我们可以用命令来控制 docker，但是如果能有一个 web 管理界面，操作什么的会方便很多，比较常见的有

- Shipyard
- Potainer

基本概念主要有三个：

- 镜像(Image)
  - 一个只读的模板，镜像可以用来创建 Docker 容器
  - 用户基于镜像来运行自己的容器。镜像是基于 Union 文件系统的层式结构
  - 可以简单创建或更新现有镜像，或者直接下载使用其他人的。可以理解为生成容器的『源代码』
- 容器(Container)
  - 容器是从镜像创建的运行实例，在启动的时候创建一层可写层作为最上层（因为镜像是只读的）
  - 可以被启动、开始、停止、删除。每个容器都是相互隔离的、保证安全的平台
  - 可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序
- 仓库(Registry)
  - 集中存放镜像文件的场所，可以是公有的，也可以是私有的
  - 最大的公开仓库是 Docker Hub
  - 国内的公开仓库包括 Docker Pool 等
  - 当用户创建了自己的镜像之后就可以使用 `push` 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库上 `pull` 下来就可以了
  - Docker 仓库的概念跟 Git 类似，注册服务器可以理解为 GitHub 这样的托管服务

另外 Docker 采用的是客户端/服务器架构，客户端只需要向 Docker 服务器或守护进程发出请求即可完成各类操作。那么问题来了，我们能 Docker 来做什么呢？我们可以：

- 统一、优化和加速本地开发和构建流程
- 保证不同的环境中可以得到相同的运行结果
- 创建隔离环境用于测试

Docker 可以提供的隔离有：

- 文件系统隔离：每个容器都有自己的 root 文件系统

- 进程隔离：每个容器都运行在自己的进程环境中
- 网络隔离：容器间的虚拟网络接口和 IP 地址都是分开的
- 资源隔离和分组：使用 cgroups 将 CPU 和内存之类的资源独立分配给每个 Docker 容器

- 查看 docker 状态 `sudo docker info`
- 查看系统中正在运行的容器的列表 `docker ps`
  - 加上 `-a` 可以列出所有容器
  - 加上 `-l` 可以列出最后一次运行的容器

一个简单的例子

接下来我们用一个简单的例子来体验下 docker

容器是独立运行的一个或一组应用，以及它们的运行态环境。对应的，虚拟机可以理解为模拟运行的一整套操作系统（提供了运行态环境和其他系统环境）和跑在上面的应用。

启动容器有两种方式，一种是基于镜像新建一个容器并启动，另外一个是在终止状态（stopped）的容器重新启动。因为 Docker 的容器实在太轻量级了，很多时候用户都是随时删除和新创建容器（对于初级应用来说后者更方便）。

当利用 `docker run` 来创建容器时，Docker 在后台运行的标准操作包括：

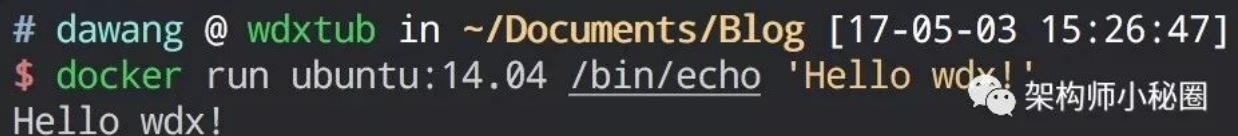
- 检查本地是否存在指定的镜像，不存在就从公有仓库下载
- 利用镜像创建并启动一个容器
- 分配一个文件系统，并在只读的镜像层外面挂载一层可读写层
- 从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去
- 从地址池配置一个 ip 地址给容器
- 执行用户指定的应用程序
- 执行完毕后容器被终止

可以利用 `docker start` 命令，直接将一个已经终止的容器启动运行。

现在，我们来创建一个 `ubuntu:14.04` 的容器

```
docker run ubuntu:14.04 /bin/echo 'Hello wdx!'
```

（结果如下图所示）



```
# dawang @ wdxsub in ~/Documents/Blog [17-05-03 15:26:47]
$ docker run ubuntu:14.04 /bin/echo 'Hello wdx!'
Hello wdx!
```

可以看到正确输出了我们的 “Hello wdx!”

接下来，我们用 `docker run -t -i ubuntu:14.04 /bin/bash` 可以启动一个 `bash` 终端用来交互。参数的意思是：

- `-t` 选项让Docker分配一个伪终端（pseudo-tty）并绑定到容器的标准输入上
- `-i` 则让容器的标准输入保持打开

我们可以输入一些命令来测试

```
# dawang @ wdxsub in ~/Documents/Blog [17-05-03 15:38:48]
$ docker run -t -i ubuntu:14.04 /bin/bash
root@e7eee4ccab6d:/# ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp
root@e7eee4ccab6d:/# ll
total 72
drwxr-xr-x 1 root root 4096 May 3 06:40 ./
drwxr-xr-x 1 root root 4096 May 3 06:40 ../
-rwxr-xr-x 1 root root 0 May 3 06:40 .dockerenv*
drwxr-xr-x 2 root root 4096 Feb 14 23:29 bin/
drwxr-xr-x 2 root root 4096 Apr 10 2014 boot/
drwxr-xr-x 5 root root 360 May 3 06:40 dev/
drwxr-xr-x 1 root root 4096 May 3 06:40 etc/
drwxr-xr-x 2 root root 4096 Apr 10 2014 home/
drwxr-xr-x 12 root root 4096 Feb 14 23:29 lib/
drwxr-xr-x 2 root root 4096 Feb 14 23:29 lib64/
drwxr-xr-x 2 root root 4096 Feb 14 23:28 media/
drwxr-xr-x 2 root root 4096 Apr 10 2014 mnt/
drwxr-xr-x 2 root root 4096 Feb 14 23:28 opt/
dr-xr-xr-x 118 root root 0 May 3 06:40 proc/
drwx----- 2 root root 4096 Feb 14 23:29 root/
drwxr-xr-x 1 root root 4096 Feb 27 19:41 run/
drwxr-xr-x 1 root root 4096 Feb 27 19:40 sbin/
drwxr-xr-x 2 root root 4096 Feb 14 23:28 srv/
dr-xr-xr-x 13 root root 0 May 3 06:40 sys/
drwxrwxrwt 2 root root 4096 Feb 14 23:29 tmp/
drwxr-xr-x 1 root root 4096 Feb 27 19:40 usr/
drwxr-xr-x 1 root root 4096 Feb 27 19:40 var/
root@e7eee4ccab6d:/# ps
  PID TTY          TIME CMD
   1 ?            00:00:00 bash
  16 ?            00:00:00 ps
```



容器的核心为所执行的应用程序，所需要的资源都是应用程序运行所必需的。除此之外，并没有其它的资源。我们用 `ps` 或 `top` 在伪终端中查看进程信息，可以看到只有我们运行的进程，没有其他花里胡哨的（上图最后一条命令）

试一试如下命令

- `cat /etc/hosts`
- `ip a`
- `ps -aux`
- `cd ~ && echo "hello wdx" > hello.txt && cat hello.txt`

（细心的同学可能会发现这里的输出暗藏玄机）

操作完成后，输入 `exit` 便可以退出这个 `ubuntu` 容器。退出之后这个容器依然存在，我们可以用 `docker ps -l`来看看：

```
# dawang @ wdxsub in ~/Documents/Blog [17-05-03 15:56:30]
$ docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
e7eee4ccab6d        ubuntu:14.04       "/bin/bash"        About an hour ago   Exited (0) 4 minutes ago
```

每个容器有一个 `Container ID` 和 `Name`，我们一般就是通过这俩来定位一个容器的。

我们可以使用 `docker pull` 命令从仓库中获取所需要的镜像。比如说

`sudo docker pull ubuntu:12.04,`

相当于

```
sudo docker pull registry.hub.docker.com/ubuntu:12.04,
```

即从注册服务器 registry.hub.docker.com中的 ubuntu 仓库来下载标记为12.04 的镜像。

如果想从其他仓库注册服务器下载，需要输入完整的地址，例如：

```
sudo docker pull dl.dockerpool.com:5000/ubuntu:12.04
```

下载完成之后就可以使用该镜像了，比如下面的语句就会创建容器，其中运行 bash：

```
sudo docker run -t -i ubuntu:12.04 /bin/bash
```

可以使用 `docker images` 来显示本地已有的镜像，如下

```
parallels@ubuntu:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
cmusvsc/apacheonda  1.1                f389a55e30e1       36 hours ago       1.847 GB
hello-world         latest             690ed74de00f       5 months ago       1.847 GB
parallels@ubuntu:~$
```

具体字段的意思一目了然，这里不再赘述。然后我们来运行官方例子 whalesay 镜像。

打开浏览器，进入 Docker Hub



搜索 whalesay 这个镜像，就可以看到结果，点进去可以看到详细内容（基于 Ubuntu）



然后我们来运行一下，使用命令

```
docker run docker/whalesay cowsay boo
```

其中 cowsay 是要运行的命令，后面的 boo 是参数。

Docker 会先在本地查找有没有镜像，如果没有就从仓库中下载，具体的运行结果是：





架构师小秘圈

架构师小秘圈

架构师小秘圈

架构师小秘圈

```
docker login --username=vourhubusername --email=yourmail@company.com,
```

对于我来说就是

```
docker login --username=wdx tub --email=dacrocodilee@gmail.com
```

成功之后大概是这样的:

```
parallels@ubuntu:~/Documents/wdxtub$ docker login --username=wdx tub --email=dacrocodilee@gmail.com
Password:
WARNING: login credentials saved in /home/parallels/.docker/config.json
Login Succeeded
```

然后就可以 push 上去了 `docker push wdx tub/wdx-whale`, 像下面这样

```
parallels@ubuntu:~/Documents/wdxtub$ docker push wdx tub/wdx-whale
The push refers to a repository [docker.io/wdxtub/wdx-whale]
0d48c73f3962: Pushed
5f70bf18a086: Pushed
d061ee1340ec: Pushed
d511ed9e12e1: Pushed
091abc5148e4: Pushed
b26122d57afa: Pushed
37ee47034d9b: Pushed
528c8710fd95: Pushed
1154ba695078: Pushed
latest: digest: sha256:a33b8602a8d00d59fec693da9609859a118e5bf777efd2b97375bac866e4c3f2 size: 8090
```

为了测试 pull 自己的镜像, 我们先把本地上的 whale 镜像删掉:

```
docker rmi -f wdx tub/wdx-whale; docker rmi -f wdx-whale
```

(如果有其他的用不着的也都删掉), 最后剩下(上课要用的镜像):

```
parallels@ubuntu:~/Documents/wdxtub$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE
cmusvsc/apachecn da 1.1          f389a55e30e1     38 hours ago    1.847 GB
```

接着来运行一下

```
docker run wdx tub/wdx-whale
```

我们可以把镜像导出到本地文件, 使用 `docker save` 命令即可, 比如针对我现在有的镜像 `wdx tub/wdx-whale(id:26ac9649d7da)`, 可以这样:

```
docker save -o wdx-local-whale.tar wdx tub/wdx-whale.
```

如果要载入的话, 使用下面的命令即可(会载入相关的元数据信息)

```
docker load --input wdx-local-whale.tar# 或者docker
load < wdx-local-whale.tar
```

在删除镜像之前要先用 `docker rm` 删掉依赖于这个镜像的所有容器.

```
sudo docker rmi $(docker images -q -f "dangling=true")
```

## 镜像的实现原理

Docker 镜像是怎么实现增量的修改和维护的? 每个镜像都由很多层次构成, Docker 使用 Union FS 将这些不同的层结合到一个镜像中去。

通常 Union FS 有两个用途, 一方面可以实现不借助 LVM、RAID 将多个 disk 挂到同一个目录下, 另一个更常用的就是将一个只读的分支和一个可写的分支联合在一起, Live CD 正是基于此方法可以允许在镜像不变的基础上允许用户在其上进行一些写操作。Docker 在 AUFS 上构建的容器也是利用了类似的原理。



举个例子，

后台运行

更多的时候，需要让 Docker在后台运行而不是直接把执行命令的结果输出在当前宿主机下。此时，可以通过添加 `-d` 参数来实现。

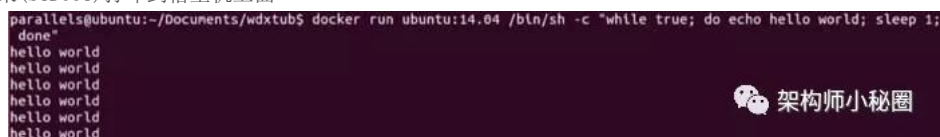
下面举两个例子来说明一下。

如果不使用 `-d` 参数运行容器

```
docker run ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
```

容器会把输出的结果(STDOUT)打印到宿主主机上面

```
parallels@ubuntu:~/Documents/wdxtub$ docker run ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```

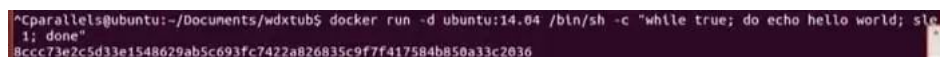


如果使用了 `-d` 参数运行容器

```
docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done",
```


则显示是这样：

```
parallels@ubuntu:~/Documents/wdxtub$ docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 1; done"
8ccc73e2c5d33e1548629ab5c693fc7422a826835c9f7f417584b850a33c2036
```



使用 `docker logs containerid` 可以查看输出，如：

```
parallels@ubuntu:~/Documents/wdxtub$ docker logs 8ccc73e2c5d33e1548629ab5c693fc7422a826835c9f7f417584b850a33c2036
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
hello world
```



使用 `-d` 参数启动后会返回一个唯一的 `id`，也可以通过 `docker ps` 命令来查看容器信息。容器是否会长久运行，是和`docker run`指定的命令有关，和 `-d` 参数无关

在使用 `-d` 参数时，容器启动后会进入后台。某些时候需要进入容器进行操作，有很多种方法，包括使用 `docker attach` 命令或 `nsenter` 工具等。具体参考[这里](#)

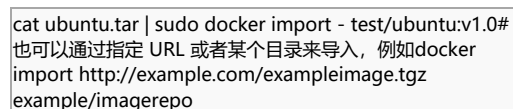
使用 `docker stop containerid` 来终止容器。终止状态的容器可以用 `docker ps -a` 命令看到。

另外，`docker restart containerid` 命令会将一个运行态的容器终止，然后再重新启动它。

如果要导出本地某个容器，可以使用 `docker export containerid` 命令。

可以使用 `docker import` 从容器快照文件中再导入为镜像，例如

```
cat ubuntu.tar | sudo docker import - test/ubuntu:v1.0#
也可以通过指定 URL 或者某个目录来导入，例如docker
import http://example.com/exampleimage.tgz
example/imagerepo
```



---

用户既可以使用 `docker load` 来导入镜像存储文件到本地镜像库，也可以使用 `docker import` 来导入一个容器快照到本地镜像库。这两者的区别在于容器快照文件将丢弃所有的历史记录和元数据信息（即仅保存容器当时的快照状态），而镜像存储文件将保存完整记录，体积也要大。此外，从容器快照文件导入时可以重新指定标签等元数据信息。

可以使用 `docker rm` 来删除一个处于终止状态的容器。如果要删除一个运行中的容器，可以添加 `-f` 参数。Docker 会发送 `SIGKILL` 信号给容器。

用 `docker ps -a` 命令可以查看所有已经创建的包括终止状态的容器，如果数量太多要一个个删除可能会很麻烦，用 `docker rm $(docker ps -a -q)` 可以全部清理掉。

注意：这个命令其实会试图删除所有的包括还在运行中的容器，不过就像上面提过的 `docker rm` 默认并不会删除运行中的容器。

仓库（Repository）是集中存放镜像的地方。

一个容易混淆的概念是注册服务器（Registry）。实际上注册服务器是管理仓库的具体服务器，每个服务器上可以有多个仓库，而每个仓库下面有多个镜像。从这方面来说，仓库可以被认为是一个具体的项目或目录。例如对于仓库地址 `dl.dockerpool.com/ubuntu` 来说，`dl.dockerpool.com` 是注册服务器地址，`ubuntu` 是仓库名。

作者：小土刀，责任编辑：帝都羊

<https://wxdxtub.com/2017/05/01/docker-guide/>

[《Linux云计算及运维架构师高薪实战班》2018年08月27日即将开课中，120天冲击Linux运维年薪30万，改变速约~~~~](#)

\*声明：推送内容及图片来源于网络，部分内容会有所改动，版权归原作者所有，如来源信息有误或侵犯权益，请联系我们删除或授权事宜。

免费好礼



欢豆

推荐一个福利包

## Python福利包

主讲人：上市公司十年开发经理

福利1：15册Python入门书籍

福利2：30集Python入门视频

福利3：50个Python商业项目源代码



长按识别二维码，即刻获取



每天精选技术干货，十万Linux人订阅

◀ **Linux人充电第一站**

长按识别二维码 关注马哥Linux运维

更多Linux好文请点击【[阅读原文](#)】哦

↓↓↓

[阅读原文](#)