Docker笔记

搜索镜像

进入目录

文件操作

设置密码

切换到root

查看已启动的容器

拉取仓库

获取Root管理员权限

获取镜像

使用示例：

查看镜像列表

使用示例：

利用 Dockerfile 来创建镜像

上传镜像

运行实例：

创建容器

查看本地images列表

用仓库 + 标签

使用image - id

可以使用 docker ps查看一件存在的容器列表,不加参数默认只显示当前运行的容器

启动容器

通过名字启动

通过容器ID启动

进入容器

停止容器

删除容器

运行容器

运行示例:

什么是Docker?

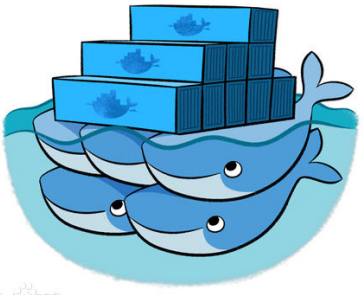
Docker 是一个开源的应用容器引擎，基于 Go 语言 并遵从Apache2.0协议开源。

Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。

容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。

Container

Docker Container（容器） 即Docker将宿主机隔开的一个个空间



Image

Docker Image（镜像）可以看作是一个特殊的文件系统，即对某一时刻容器状态的备份

相关链接:

Docker 官网：http://www.docker.com
Github Docker 源码：https://github.com/docker/docker

Docker的应用场景

- Web 应用的自动化打包和发布。
- 自动化测试和持续集成、发布。
- 在服务型环境中部署和调整数据库或其他的后台应用。
- 从头编译或者扩展现有的OpenShift或Cloud Foundry平台来搭建自己的PaaS环境。

Docker 的优点

- **1、简化程序：**

Docker 让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，便可以实现虚拟化。Docker改变了虚拟化的方式，使开发者可以直接将自己的成果放入Docker中进行管理。方便快捷已经是 Docker的最大优势，过去需要用数天乃至数周的 任务，在Docker容器的处理下，只需要数秒就能完成。

- **2、避免选择恐惧症：**

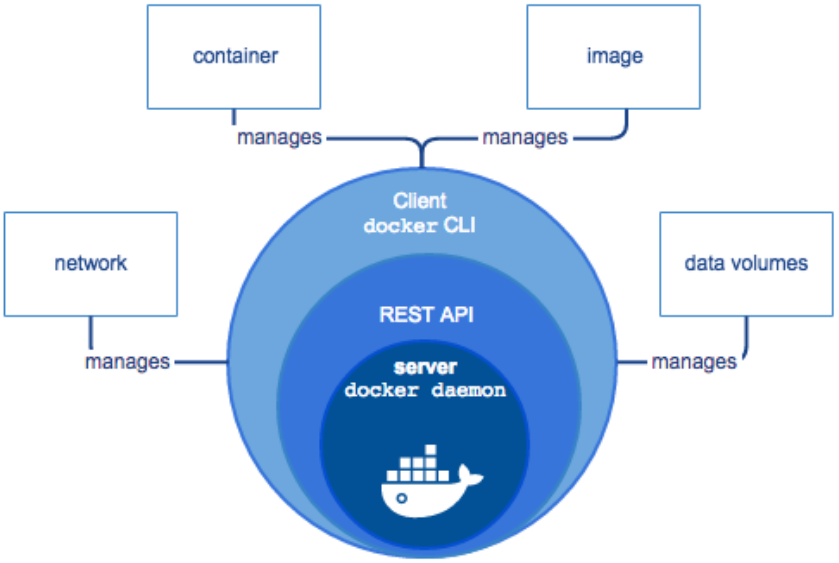
如果你有选择恐惧症，还是资深患者。Docker 帮你 打包你的纠结！比如 Docker 镜像；Docker 镜像中包含了运行环境和配置，所以 Docker 可以简化部署多种应用实例工作。比如 Web 应用、后台应用、数据库应用、大数据应用比如 Hadoop 集群、消息队列等等都可以打包成一个镜像部署。

- **3、节省开支：**

一方面，云计算时代到来，使开发者不必为了追求效果而配置高额的硬件，Docker 改变了高性能必然高价格的思维定势。Docker 与云的结合，让云空间得到更充分的利用。不仅解决了硬件管理的问题，也改变了虚拟化的方式。

Docker引擎

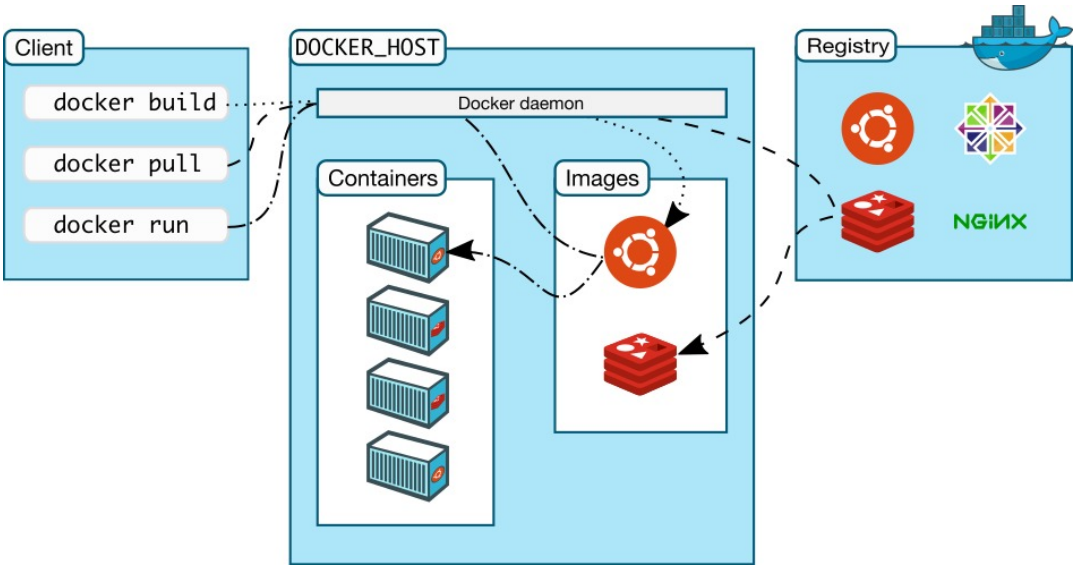
docker引擎是一个c/s结构的应用，主要组件见下图：



- Server是一个常驻进程
- REST API 实现了client和server间的交互协议
- CLI 实现容器和镜像的管理，为用户提供统一的操作界面

Docker构架

Docker使用C/S架构，Client 通过接口与Server进程通信实现容器的构建，运行和发布。client和server可以运行在同一台集群，也可以通过跨主机实现远程通信。

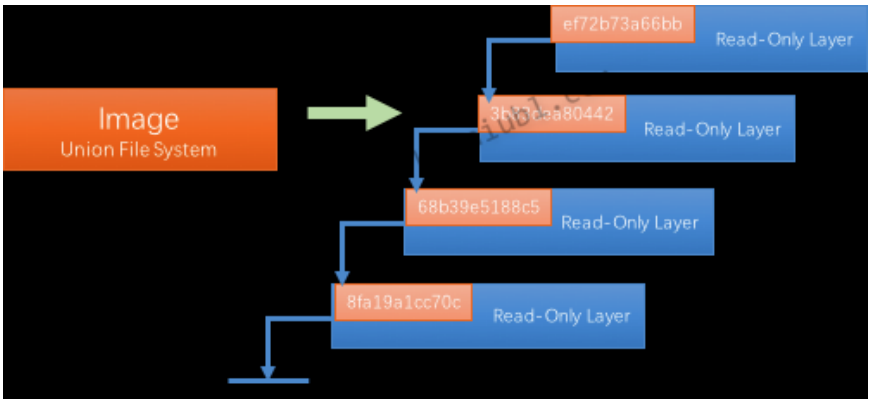


核心概念

镜像(image)

Docker 镜像（Image）就是一个只读的模板。例如：一个镜像可以包含一个完整的操作系统环境，里面仅安装了 Apache 或用户需要的其它应用程序。镜像可以用来创建 Docker 容器，一个镜像可以创建很多容器。Docker 提供了一个很简单的机制来创建镜像或者更新现有的镜像，用户甚至可以直接从其他人那里下载一个已经做好的镜像来直接使用。

镜像（Image）就是一堆只读层（read-only layer）的统一视角，也许这个定义有些难以理解，看看下面这张图：



右边我们看到了多个只读层，它们重叠在一起。除了最下面一层，其它层都会有一个指针指向下一层。这些层是Docker内部的实现细节，并且能够在docker宿主机的文件系统上访问到。统一文件系统（Union File System）技术能够将不同的层整合成一个文件系统，为这些层提供了一个统一的视角，这样就隐藏了多层的存在，在用户的角度看来，只存在一个文件系统。

仓库(repository)

仓库（Repository）是集中存放镜像文件的场所。有时候会把仓库和仓库注册服务器（Registry）混为一谈，并不严格区分。实际上，仓库注册服务器上往往存放着多个仓库，每个仓库中又包含了多个镜像，每个镜像有不同的标签（tag）。

仓库分为公开仓库（Public）和私有仓库（Private）两种形式。最大的公开仓库是 Docker Hub，存放了数量庞大的镜像供用户下载。国内的公开仓库包括 时速云 、网易云 等，可以提供大陆用户更稳定快速的访问。当然，用户也可以在本地网络内创建一个私有仓库。

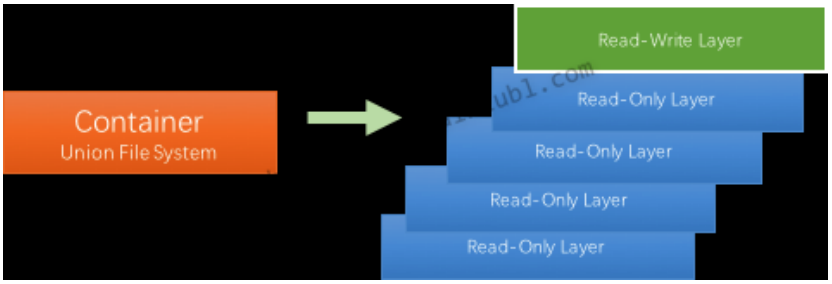
当用户创建了自己的镜像之后就可以使用 push 命令将它上传到公有或者私有仓库，这样下次在另外一台机器上使用这个镜像时候，只需要从仓库上 pull 下来就可以了。

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。

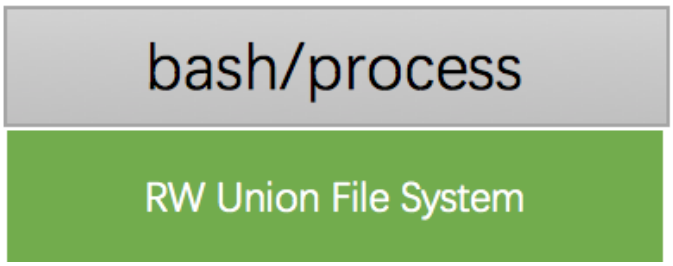
容器(container)

Docker 利用容器（Container）来运行应用。容器是从镜像创建的运行实例。它可以被启动、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。

容器的定义和镜像几乎一模一样，也是一堆层的统一视角，唯一区别在于容器的最上面那一层是可读可写的。



一个运行态容器被定义为一个可读写的统一文件系统加上隔离的进程空间和包含其中的进程。下面这张图片展示了一个运行中的容器。



正是文件系统隔离技术使得Docker成为了一个非常有潜力的虚拟化技术。一个容器中的进程可能会对文件进行修改、删除、创建，这些改变都将作用于可读写层。

网络和端口映射

- 网络配置
- host 模式
- container 模式
- none 模式
- bridge 模式，默认
- --net=（具体模式名称）

端口映射

- -p

常用命令

更新docker

```
sudo apt update
```

> 大多数开源软件、应用和工具在安装之后都会先执行“更新”操作。如果数据库没有自动升级，那么系统不会知道是不是有一个新的可替换package。所以在任何Linux系统中，**更新现有的库都是首先要做的**。
> 更新数据库需要超级用户权限，所以你需要运行“sudo”。

安装docker

```
sudo apt-get install -y docker.io
```

> 如果你已经知道了自己需要安装的package的名字，那么就可以直接执行命令sudo apt install <package_name>。当然，你只需要将<package_name>替换成你真正需要的名字就好，比如你想安装mplayer就可以输入命令：sudo apt install mplayer。

查看版本

```
docker --version
```

查看镜像

```
sudo docker images
```

查看所有容器

```
docker ps  -a
```

查询容器

```
docker ps  -a | grep 查找的名称
```

停止容器

```
docker stop containerId/containerName
```

启动容器

```
docker start containerId/containerName
```

删除容器

```
docker rm containerId/containerName
```

拷贝文件

```
docker cp 宿主机目录及文件 容器名称:容器目录
```

进入root管理员

```
sudo su
```

退出

```
exit
```

查看镜像

```
sudo docker images
```

搜索镜像

```
docker search Name
```

进入目录

```
 cd /xxx/xxx
```

文件操作

```
sudo vi sshd_config
[i] 进入编辑状态
[esc] 退出编辑状态
[shift + :] 键入文件操作命令
[文件操作] (w)保存 (q)退出 (wq)保存退出    (q!)强制退出
```

设置密码

```
sudo passwd
```

切换到root

```
su root
```

创建一个新的容器并运行一个命令

```
docker run [OPTIONS] IMAGE [COMMAND] [ARG...]
OPTIONS说明：
 -a stdin: 指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；
 -d: 后台运行容器，并返回容器ID；
 -i: 以交互模式运行容器，通常与 -t 同时使用；
 -p: 端口映射，格式为：主机(宿主)端口:容器端口
 -t: 为容器重新分配一个伪输入终端，通常与 -i 同时使用；
 --name="nginx-lb": 为容器指定一个名称；
 --dns 8.8.8.8: 指定容器使用的DNS服务器，默认和宿主一致；
 --dns-search example.com: 指定容器DNS搜索域名，默认和宿主一致；
 -h "mars": 指定容器的hostname；
 -e username="ritchie": 设置环境变量；
 --env-file=[]: 从指定文件读入环境变量；
 --cpuset="0-2" or --cpuset="0,1,2": 绑定容器到指定CPU运行；
 -m :设置容器使用内存最大值；
 --net="bridge": 指定容器的网络连接类型，支持 bridge/host/none/container: 四种类型；
 --link=[]: 添加链接到另一个容器；
 --expose=[]: 开放一个端口或一组端口；

实例：
 docker run -d -p 8888:8080 --name tomcat tomcat
```

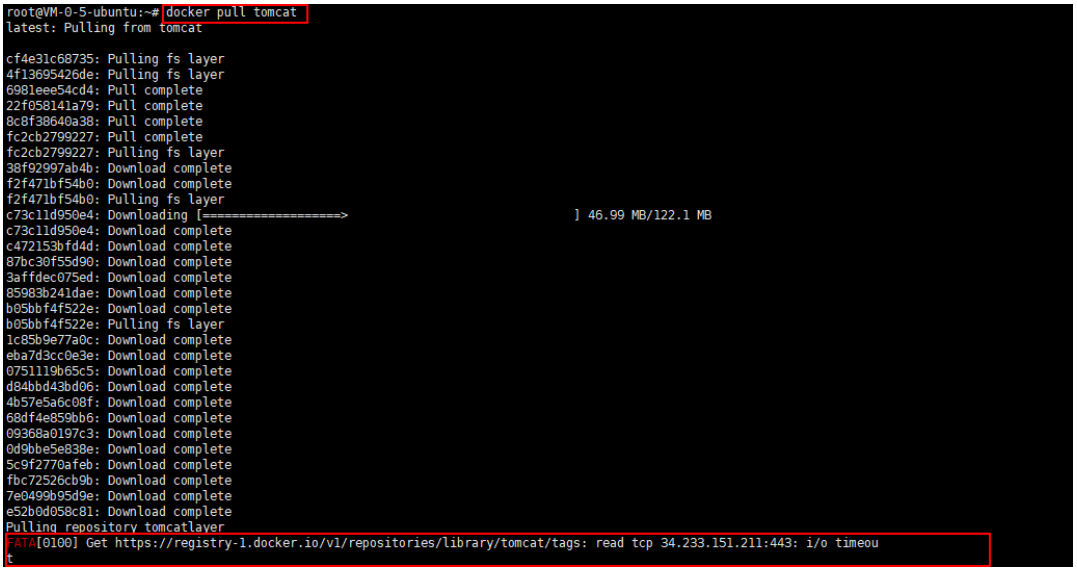
查看已启动的容器

```
docker ps
```

拉取仓库

```
tomcat docker pull tomcat
```

若拉去失败，如下图：



使用如下命令：

```
curl -sSL https://get.daocloud.io/daotools/set_mirror.sh | sh -s http://ef017c13.m.daocloud.io
please upgrade your docker to v1.9 or later
```

然后重新拉去，显示如下图既成功：

```
root@VM-0-5-ubuntu:~# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             VIRTUAL SIZE
<none>              <none>              fc2cb2799227        47 hours ago        133.7 MB
root@VM-0-5-ubuntu:~# docker pull tomcat
latest: Pulling from tomcat

b22161d3a38f: Pulling fs layer
38f92997ab4b: Pulling fs layer
649a4894845c: Pull complete
c73c11d950e4: Pull complete
4afa8175a669: Pull complete
c472153bfd4d: Pull complete
87bc30f55d90: Pull complete
3affdec075ed: Pull complete
85983b241dae: Pull complete
b05bbf4f522e: Pull complete
5687a5236f27: Pull complete
1c85b9e77a0c: Pull complete
eba7d3cc0e3e: Pull complete
0751119b65c5: Pull complete
d84bbd43bd06: Pull complete
4b57e5a6c08f: Pull complete
68df4e859bb6: Pull complete
09368a0197c3: Pull complete
0d9bbe5e838e: Pull complete
5c9f2770afeb: Pull complete
fbc72526cb9b: Pull complete
7e0499b95d9e: Pull complete
e52b0d058c81: Pull complete
e52b0d058c81: Pulling fs layer
cf4e31c68735: Already exists
4f13695426de: Already exists
a6931ed97645: Already exists
6981eee54cd4: Already exists
22f058141a79: Already exists
8c8f38640a38: Already exists
Digest: sha256:6f479831a2f814f6e953a9dd7b3aaf6f1b08d691d68c7a154f99f66cbf250617
Status: Downloaded newer image for tomcat:latest
```

获取Root管理员权限

```
sudo su
```

获取镜像

从仓库获取所需要的镜像。

```
docker pull
```

使用示例：

```
docker pull centos:centos6
```

实际上相当于 docker pull registry.hub.docker.com/centos:centos6

有时候官方仓库注册服务器下载较慢，可以从其他仓库下载。 从其它仓库下载时需要指定完整的仓库注册服务器地址。

查看镜像列表

```
docker images
```

列出了所有顶层（top-level）镜像。实际上，在这里我们没有办法区分一个镜像和一个只读层，所以我们提出了top-level镜像。只有创建容器时使用的镜像或者是直接pull下来的镜像能被称为顶层（top-level）镜像，并且每一个顶层镜像下面都隐藏了多个镜像层。

使用示例：

```
$ docker images
REPOSITORY TAG IMAGE ID CREATED SIZE
centos centos6 6a77ab6655b9 8 weeks ago 194.6 MB
ubuntu latest 2fa927b5cdd3 9 weeks ago 122 MB
```

> 在列出信息中，可以看到几个字段信息
> 来自于哪个仓库，比如 ubuntu
> 镜像的标记，比如 14.04
> 它的 ID 号（唯一）
> 创建时间
> 镜像大小

利用 Dockerfile 来创建镜像

```
docker build
```

> 使用 docker commit 来扩展一个镜像比较简单，但是不方便在一个团队中分享。我们可以使用
> docker build 来创建一个新的镜像。为此，首先需要创建一个 Dockerfile，包含一些如何创建镜像的
> 指令。新建一个目录和一个 Dockerfile。

```
mkdir hainiu
```

```
cd hainiu
touch Dockerfile
Dockerfile 中每一条指令都创建镜像的一层，例如：

FROM centos:centos6
MAINTAINER sandywei <sandy@hainiu.tech>
# move all configuration files into container

 RUN yum install -y httpd
 EXPOSE 80
 CMD ["sh","-c","service httpd start;bash"]
 Dockerfile 基本的语法是
```

上传镜像

```
docker push
```

> 用户可以通过 docker push 命令，把自己创建的镜像上传到仓库中来共享。例如，用户在 Docker Hub 上完成注册后，可以推送自己的镜像到仓库中。

运行实例：

```
$ docker push hainiu/httpd:1.0
```

创建容器

```
docker create <image-id>
```

> docker create 命令为指定的镜像（image）添加了一个可读写层，构成了一个新的容器。注意，这个容器并没有运行。
>
> docker create 命令提供了许多参数选项可以指定名字，硬件资源，网络配置等等。

查看本地images列表

```
$ docker images
```

用仓库 + 标签

```
$ docker create -it --name centos6_container centos:centos6
```

使用image - id

$ docker create -it --name centos6_container 6a77ab6655b9 bash

b3cd0b47fe3db0115037c5e9cf776914bd46944d1ac63c0b753a9df6944c7a67

可以使用 docker ps查看一件存在的容器列表,不加参数默认只显示当前运行的容器

```
$ docker ps -a
```

可以使用 -v 参数将本地目录挂载到容器中。

$ docker create -it --name centos6_container -v /src/webapp:/opt/webapp centos:centos6

这个功能在进行测试的时候十分方便，比如用户可以放置一些程序到本地目录中，来查看容器是否正常工作。本地目录的路径必须是绝对路径，如果目录不存在 Docker 会自动为你创建它。

启动容器

```
docker start <container-id>
```

> Docker start命令为容器文件系统创建了一个进程隔离空间。注意，每一个容器只能够有一个进程隔离空间。

通过名字启动

```
$ docker start -i centos6_container
```

通过容器ID启动

```
$ docker start -i b3cd0b47fe3d
```

进入容器

```
docker exec <container-id>
```

> 在当前容器中执行新命令，如果增加 -it参数运行bash 就和登录到容器效果一样的。

停止容器

```
docker stop <container-id>
```

删除容器

```
docker rm <container-id>
```

运行容器

```
docker run <image-id>
```

docker run就是docker create和docker start两个命令的组合,支持参数也是一致的，如果指定容器名字是，容器已经存在会报错,可以增加 --rm 参数实现容器退出时自动删除。

运行示例:

```
docker create -it --rm --name centos6_container centos:centos6
```

查看容器列表

```
docker ps
```

docker ps 命令会列出所有运行中的容器。这隐藏了非运行态容器的存在，如果想要找出这些容器，增加 -a 参数。

删除镜像

```
docker rmi <image-id>
```

删除构成镜像的一个只读层。你只能够使用docker rmi来移除最顶层（top level layer)
（也可以说是镜像），你也可以使用-f参数来强制删除中间的只读层。

commit容器

```
docker commit <container-id>
```

将容器的可读写层转换为一个只读层，这样就把一个容器转换成了不可变的镜像。

镜像保存

```
docker save <image-id>
```

创建一个镜像的压缩文件，这个文件能够在另外一个主机的Docker上使用。和export命令不同，这个命令
为每一个层都保存了它们的元数据。这个命令只能对镜像生效。

保存centos镜像到centos_images.tar 文件

```
$ docker save -o centos_images.tar centos:centos6
```

或者直接重定向

```
$ docker save -o centos_images.tar centos:centos6 > centos_images.tar
```

容器导出

```
docker export <container-id>
```

创建一个tar文件，并且移除了元数据和不必要的层，将多个层整合成了一个层，只保存了当前统一视角看到

的内容。expoxt后的容器再import到Docker中，只有一个容器当前状态的镜像；
而save后的镜像则不同，
它能够看到这个镜像的历史镜像。

inspect
docker inspect <container-id> or <image-id>
docker inspect命令会提取出容器或者镜像最顶层的元数据

指定IP

- 动态IP
- 固定IP

创建网络，指定网段

```
docker network create --subnet=172.18.0.0/16 mynetwork
```

创建容器，指定IP

```
docker run -it -d --net mynetwork --ip 172.18.0.8 --name mytomcat tomcat
```

Dockerfile

内置命令

FROM： 依赖的底层镜像
MAINTAINER： 指定镜像创建者
ENV： 设置环境变量
RUN： 运行shell命令
COPY： 将编译机本地文件拷贝到镜像文件系统中
EXPOSE： 指定监听端口
ENTRYPOINT： 与执行命令，创建容器并启动后才执行

Dockerfile操作

编写Dockerfile
文件的名称必须为Dockerfile
执行命令生成镜像
docker build [OPTIONS] PATH | URL | -

容器可视化

- DockerUI
- Shipyard
- Portainer

总结

- Docker中涉及的概念
- 了解运行原理
- Docker的安装
- 镜像的操作
- 容器的操作
- 使用可视化管理工具
