docker入门

计算机科学与技术17-1 陈巍

写在前面

一、基本概念

docker使用容器技术，每个容器是互相隔离的，每一个容器都有一个属于自己的文件系统。

将项目打包成一个镜像。

镜像image：模版，通过模版来创建容器服务。一个镜像可以创建多个容器。

> 镜像 --- run --- 容器

```
运行镜像：
    docker run 「镜像名称」
```

容器container： 利用docker容器技术，可以独立运行一个或者一组应用（通过「镜像」创建的）。

> 启动
>
> 停止
>
> 删除

简单理解成一个简单的linux。

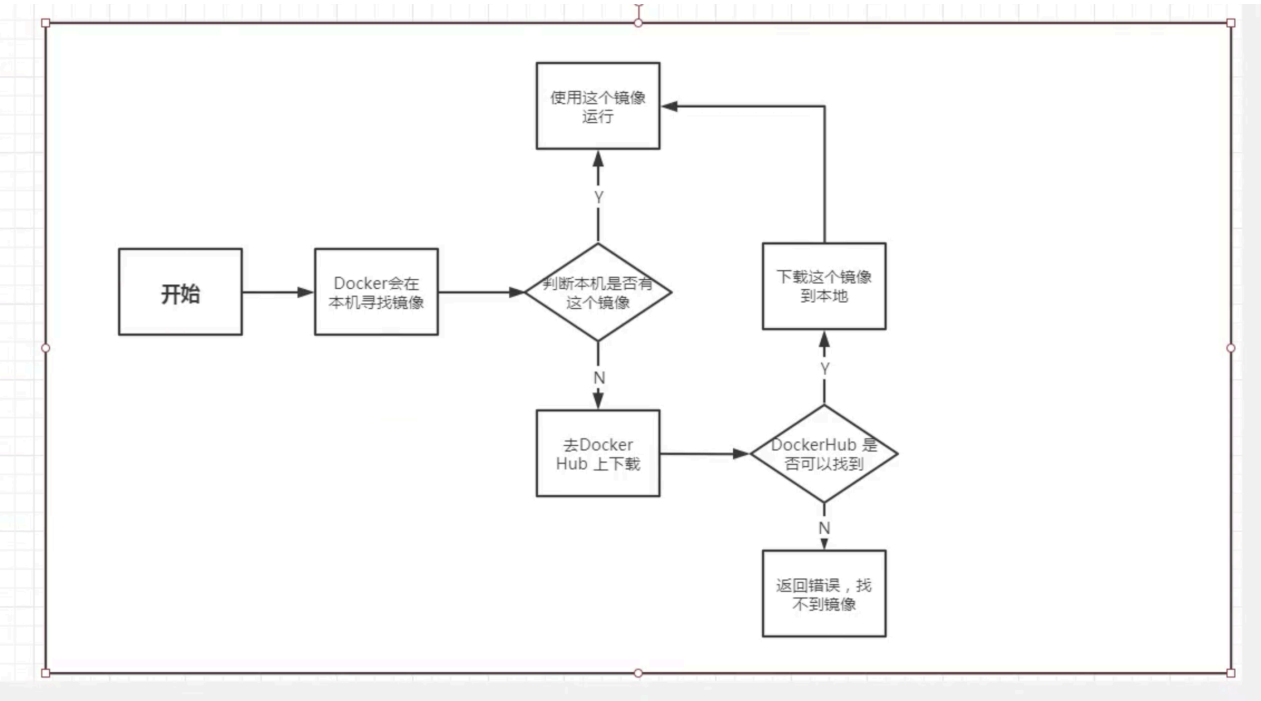
仓库repository： 存放镜像

共有仓库 & 私有仓库

默认是 docekr hub（国外）

二、docker-run的流程和运行原理

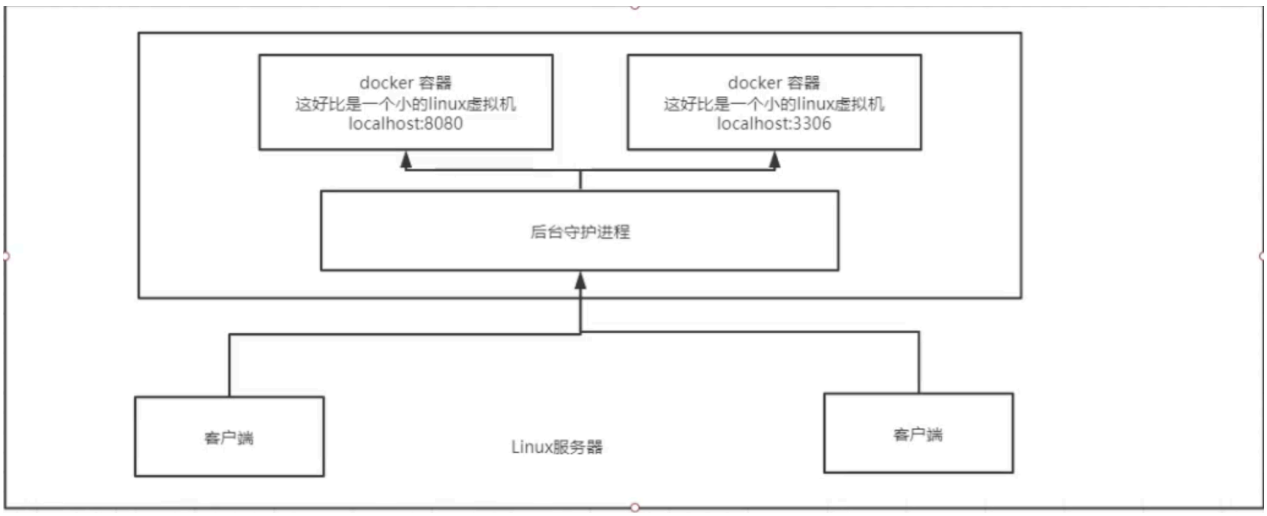
docker run的流程：



docker 工作原理：

docker是一个Client-Server结构的系统，docker的守护进程运行在主机上。通过Socket从客户端访问！

DockerServer接收到Docker-Client的指令，就回去执行这个命令。



docker客户端通过命令操作「守护进程」，「守护进程」再对「容器」进行操作。

【补充】「容器」想要和「Linux服务器」进行通信，还需要指定「Linux服务器」访问「容器」的端口。

三、Docker常用命令

3.1 帮助命令

```
docker version                          # 查看「ocker版本」
docker info                             # 显示「ocker的系统信息包括，「镜像」和「容器」数量
docker [命令] --help                     # 帮助命令
```

3.2 镜像命令

3.2.1 docker images 查看本机上的镜像

```
[will@master ~]$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
hello-world         latest              bf756fb1ae65        6 months ago        13.3kB

# 解释
REPOSITORY   镜像的仓库源
TAG          镜像的标签
IMAGE ID     镜像的id
CREATED      镜像的创建时间
SIZE         镜像的大小

# 可选项
  -a, --all             # 列出所有的镜像
  -q, --quiet           # 只显示镜像的id

```

3.2.2 docker serach 搜索镜像

```
[will@master ~]$ docker search mysql
NAME                              DESCRIPTION                                     STARS               OFFICIAL            AUTOMATED
mysql                             MySQL is a widely used, open-source relation…   9704                [OK]
mariadb                           MariaDB is a community-developed fork of MyS…   3534                [OK]

# 可选项，通过「收藏/STARS」来过滤
  --filter=STARS=3000   # 搜索出来的镜像是收藏 > 3000
```

3.2.3 docker pull 下载镜像

```
# 下载镜像 docker pull 镜像名[:tag]
[will@master ~]$ docker pull mysql
Using default tag: latest                   # 如果不写tag，就默认下载latest版本/最新版本
latest: Pulling from library/mysql
8559a31e96f4: Pulling fs layer              # 分层下载，docker image的核心，联合文件系统
d51ce1c2e575: Pulling fs layer
c2344adc4858: Pull complete
fcf3ceff18fc: Pull complete
16da0c38dc5b: Pull complete
b905d1797e97: Pull complete
4b50d1c6b05c: Pull complete
c75914a65ca2: Pull complete
1ae8042bdd09: Pull complete
453ac13c00a3: Pull complete
9e680cd72f08: Pull complete
a6b5dc864b6c: Pull complete
Digest: sha256:8b7b328a7ff6de46ef96bcf83af048cb00a1c86282bfca0cb119c84568b4caf6 # 签名
```

```
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest  # 真实地址

# 等价命令
docker pull mysql
docker pull docker.io/library/mysql:latest

# 指定版本下载
[will@master ~]$ docker pull mysql:5.7
5.7: Pulling from library/mysql
8559a31e96f4: Already exists
d51ce1c2e575: Already exists
c2344adc4858: Already exists
fcf3ceff18fc: Already exists
16da0c38dc5b: Already exists
b905d1797e97: Already exists
4b50d1c6b05c: Already exists
d85174a87144: Pull complete
a4ad33703fa8: Pull complete
f7a5433ce20d: Pull complete
3dcd2a278b4a: Pull complete
Digest:
sha256:32f9d9a069f7a735e28fd44ea944d53c61f990ba71460c5c
183e610854ca4854
Status: Downloaded newer image for mysql:5.7
docker.io/library/mysql:5.7
```

3.2.4 docker rmi 删除镜像

```
docker rmi -f 镜像id                                    # 删除指定镜像

docker rmi -f 镜像id 镜像id 镜像id                      # 删除多个镜像

docker rmi -f $(docker images -aq)                      # 递归删除所有镜像，通过镜像id
```

四、容器命令

创建容器的前提是已经拥有镜像。

4.1 下载centos镜像

```
docekr pull centos
```

4.2 新建容器并启动

```
docker run [可选参数] [镜像名称]

# 参数说明
 --name="Name"        # 容器名字  tomcat01 tomcat02，用来区分容器
 -d                   # 后台方式运行
 -it                  # 使用交互方式运行，进入容器查看内容
 -p                   # 指定容器的端口 -p 8080:8080
    -p ip:主机端口:容器端口
    -p 主机端口:容器端口
    -p 容器端口
 -P                   # 随机指定端口

# 调试，测试并启动容器
[will@master ~]$ docker run -it centos /bin/bash
[root@2aabb466d68c /]# ls                    # 查看容器内的centos
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var

# 从容器中退出，回到主机
[root@2aabb466d68c /]# exit
exit
```

4.3 列出所有运行的容器

```
# docker ps 运行
 # 列出当前正在运行的容器
 -a # 列出当前正在运行的容器 + 曾经运行的容器
 -n # 显示最近创建的容器$(?)个
 -q # 只显示容器的编号
[wiaster ~]$ docker ps
CONER ID        IMAGE               COMMAND
CRE             STATUS              PORTS
NAM

[wiaster ~]$ docker ps -a
CONER ID        IMAGE               COMMAND
CRE             STATUS                    PORTS
NAMES
2aabb466d68c        centos              "/bin/bash"
5 minutes ago       Exited (0) 3 minutes ago
upbeat_leavitt
70a5b33fb3ec        bf756fb1ae65        "/hello"
17 hours ago        Exited (0) 17 hours ago
affectionate_sammet
a00884c700f         bf756fb1ae65        "/hello"
17 hours ago        Exited (0) 17 hours ago
serene_germain
```

4.4 退出容器

```
exit                # 容器停止工作，退出
Ctrl + P + Q        # 容器不停止工作，退出
```

4.5 删除容器

```
docker rm 容器id                          # 删除指定容器，不能删除正在运行的容器，使用 rm -f
docker rm -f $(docker ps -aq)             # 删除所有容器（使用参数传递）
docker ps -a -q|xargs docker rm           # 删除所有容器
```

4.6 启动和停止容器

```
docker start 容器id              # 「启动」指定容器
docker restart 容器id            # 「重启」指定容器
docker stop 容器id               # 「停止」正在运行的容器
docker kill 容器id               # 「杀掉」/「强制停止」正在运行的容器
```

五、其他常用命令

5.1 后台启动容器

```
# docker run -d [镜像名称]
[will@master ~]$ docker run -d centos
246fdfcaa911dabfafc61f2586107bbb479a9379bae77d62ab13716bc9402cf6

# 问题 docker ps后，centos停止

# 【注意】：docker 容器使用后台运行，就必须要有一个前台进程，docker发现没有应用，就会停止
#    nginx，容器启动后，发现自己没有提供服务，就会立刻停止，就是没有程序了
```

5.2 查看日志

```
# 显示指定容器的$(number)的日志
docker logs -f -t --tail [number] [容器id]

# 显示指定容器的所有日志
docekr logs -t -f [容器id]

# 显示日志
 -tf                  # 显示日志
 --tail number        # 要显示的日志条数
```

5.3 查看容器信息

```
# 命令
docker inspect [容器id]
```

5.4 进入「正在运行」的容器

```
# 容器通常都是使用后台方式运行，需要进入容器，修改一些配置

# 命令
docker exec -it [容器id] bashShell

docker attach [容器id]

# 测试
docker exec -it 容器id /bin/bash           # 进入centos

docker attach 容器id

# 总结
docker exec                                # 进入容器开启一个新的终端，可以在里面操作
docker attach                              # 进入容器正在执行的终端，不会启动新的进程
```

5.5 从容器拷贝文件到主机上

文件：从「容器」到「主机」

```
# 命令
docker cp [容器id:文件路径] [主机路径]

# 测试
# 创建centos容器，并进入容器内部
[will@master ~]$ docker run -it centos /bin/bash
[root@e70304a9b2b3 /]# ls
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
[root@e70304a9b2b3 /]# cd home
```

```
[root@e70304a9b2b3 home]# ls -a
.  ..
# 编写测试文件
[root@e70304a9b2b3 home]# touch test.java
[root@e70304a9b2b3 home]# ls
test.java
[root@e70304a9b2b3 home]# exit
exit
[will@master ~]$ docker ps -a
CONTAINER ID        IMAGE               COMMAND
CREATED             STATUS                      PORTS
NAMES
e70304a9b2b3        centos              "/bin/bash"
39 seconds ago      Exited (0) 14 seconds ago
pensive_mirzakhani
# 将文件拷贝到「主机」
[will@master ~]$ docker cp e70304a9b2b3:/home/test.java
/home/will/
# 查看文件
[will@master ~]$ ls /home/will/
Desktop  Documents  Downloads  Music  Pictures  Public
Templates  test.java  Videos

# 【注】：「拷贝」是一个手动过程，可以使用「-v」「数据卷技术」实现自动同步  /home  /home
```

5.6 查看docker使用cpu的状态

```
docker stats
```

5.7 限制容器使用的内存大小

```
# 修改配置文件 or 使用-e 环境配置修改
-e ES_JAVA_OPTS="-Xms64m -Xmx512m"      # 内存范围[64m, 512m]
```

六、docker可视化工具

- portainer

  docker的图形化管理工具

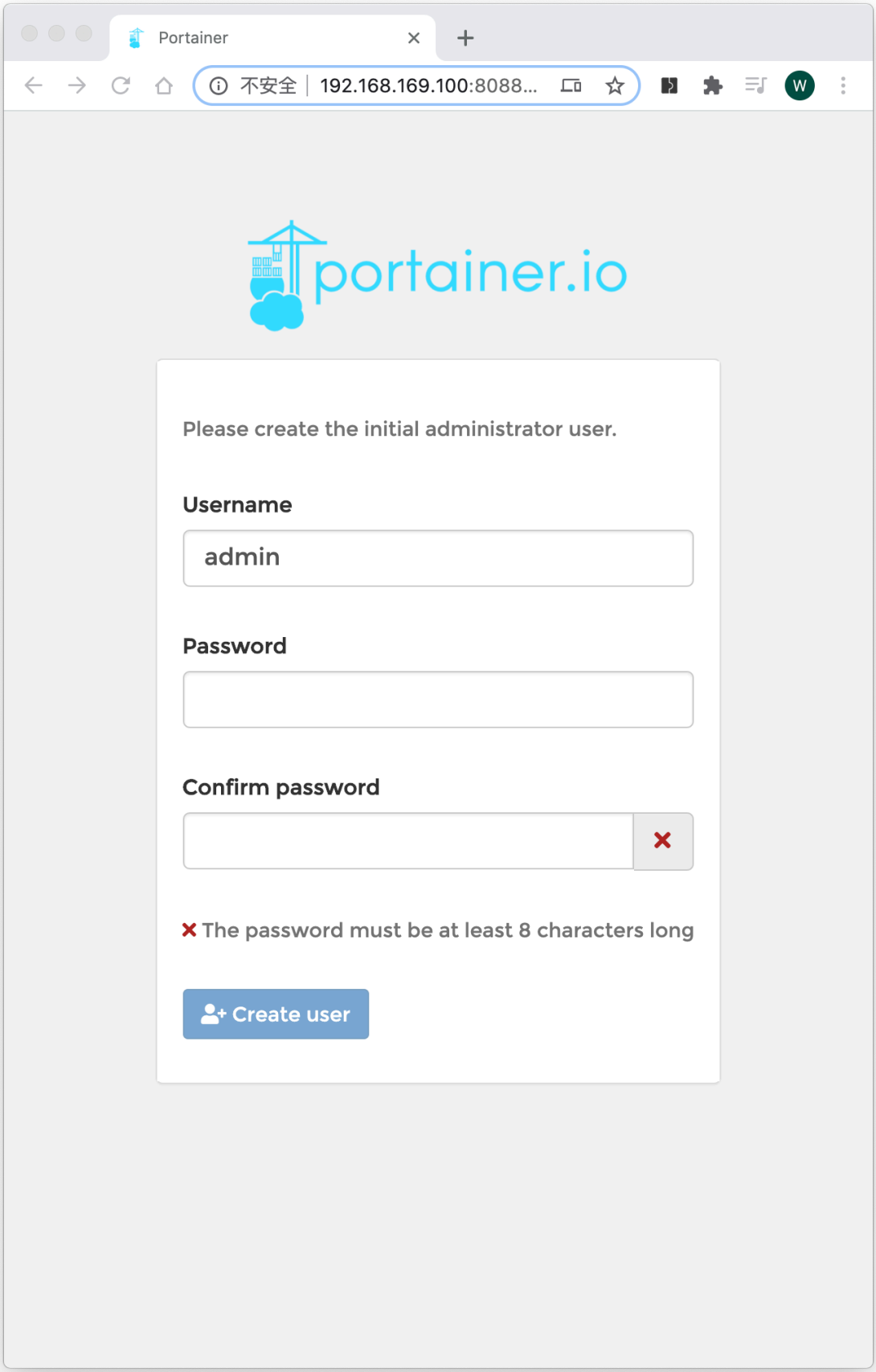
  安装

  ```
  docker run -d -p 8088:9000 --restart=always -v
  /var/run/docker.sock:/var/run/docker.sock --
  privileged=true portainer/portainer
  ```

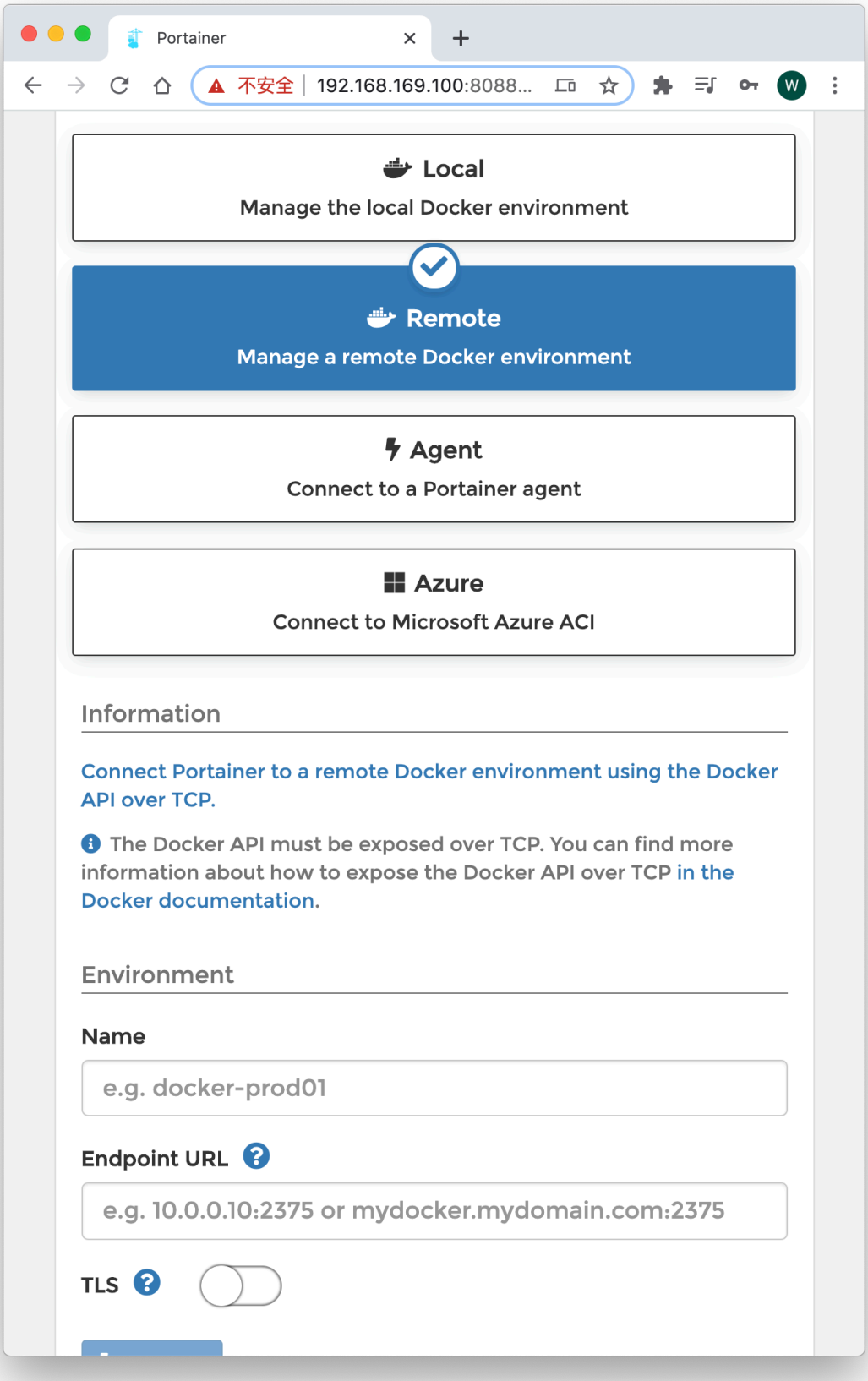
  访问

  http://ip:8088

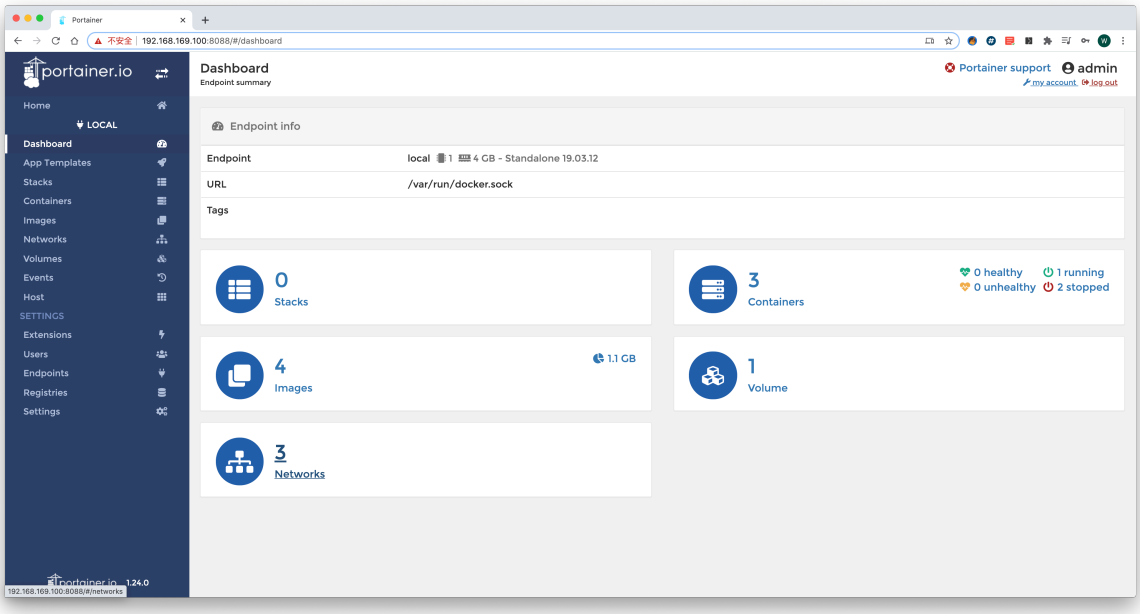
  设置登录密码


Portainer
不安全 | 192.168.169.100:8088...
portainer.io
Please create the initial administrator user.
Username
admin
Password
Confirm password
The password must be at least 8 characters long
Create user

选择本地（Local）


Portainer
不安全
192.168.169.100:8088...
Local
Manage the local Docker environment
Remote
Manage a remote Docker environment
Agent
Connect to a Portainer agent
Azure
Connect to Microsoft Azure ACI
Information
Connect Portainer to a remote Docker environment using the Docker API over TCP.
The Docker API must be exposed over TCP. You can find more information about how to expose the Docker API over TCP in the Docker documentation.
Environment
Name
e.g. docker-prod01
Endpoint URL
e.g. 10.0.0.10:2375 or mydocker.mydomain.com:2375
TLS

进入面板



- Rancher(主)

  略 :)

七、Docker镜像原理

镜像是一种轻量级、可执行的独立软件包，用来打包软件运行环境和机遇运行环境开发的软件，它包含运行某个软件所需的所有内容，包括代码、运行时、库、环境变量和配置文件。

所有的应用，可以直接打包docker镜像，就可以直接跑起来。

7.1 获取镜像：

- 从docker hub中获取
- 从他人那里拷贝
- 自己制作

7.2 镜像加载原理

UnionFS联合文件系统

Union文件系统是一种分层、轻量级并且高性能的文件系统。

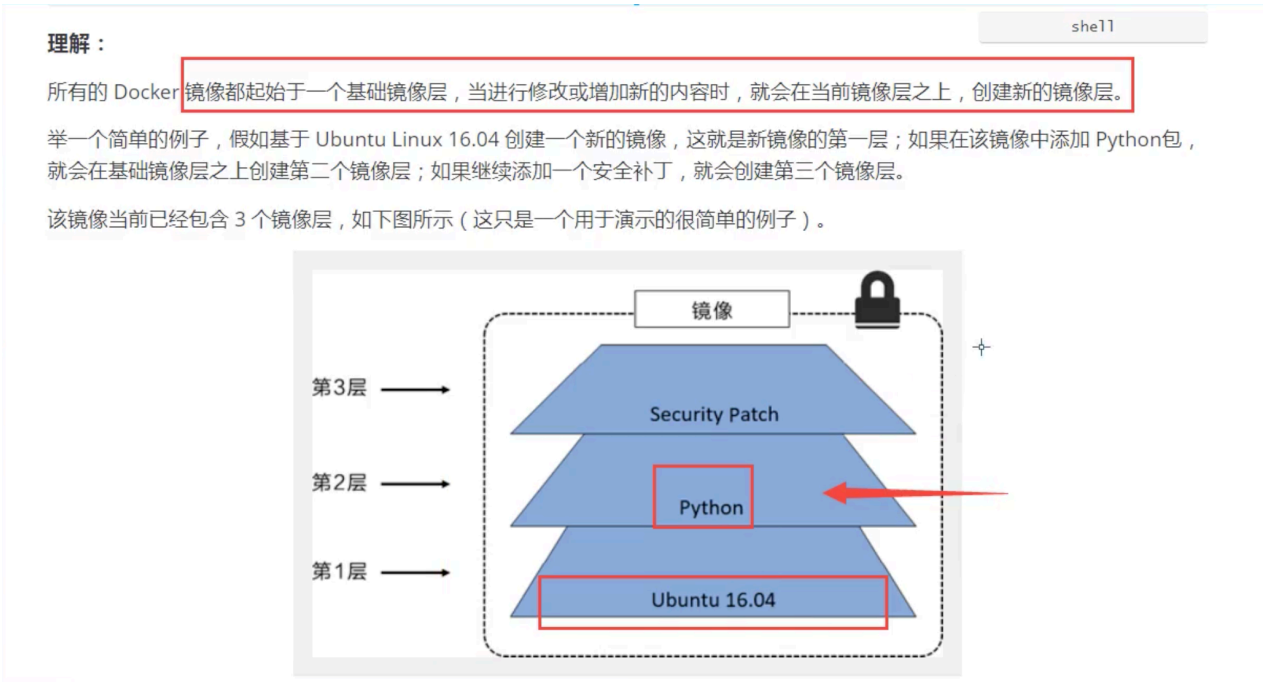
镜像加载原理

docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统是UnionFS。

容器就是一个小的Linux虚拟机环境，可以理解为一个CentOS。

我们可以创建一个「CentOS」容器，使用「CentOS」镜像。当我们需要使用其他容器而下载其他镜像时，如果这个容器在运行时需要用到CentOS，那么我们在下载容器的时候，也会下载CentOS镜像和其他用到的镜像。

镜像的分层



我们从docker hub中pull的镜像就是「未经修改的镜像层」。当我们使用了run命令将镜像运行起来后，就创建了一个容器。此时，就出现了第二层——「容器层」。我们的所有操作，都是基于容器层的操作，我们的修改都是在容器层上修改的，最后操作的结果成为了一个容器层。如下图：

七、Commit镜像

提交一个自己的镜像。

```
docker commit 提交容器，使其成为一个新的副本

docker commit -m="提交的信息" -a="作者" [容器id] 目标镜像名称:[tag/版本] # 目标镜像名称 = 自定义镜像名称
```

测试 - 提交一个自己的镜像

```
# 启动一个默认的tomcat
[will@master ~]$ docker run -it --name tomcat01 -p 3080:8080 tomcat

# 发现这个磨人的tomcat是没有webapps应用：官方默认镜像中webapps下面是没有文件的
root@29c525f703dc:/usr/local/tomcat# ls
BUILDING.txt     LICENSE  README.md  RUNNING.txt  conf
logs         temp     webapps.dist
CONTRIBUTING.md  NOTICE   RELEASE-NOTES  bin
lib   native-jni-lib  webapps  work
root@29c525f703dc:/usr/local/tomcat# cd webapps
root@29c525f703dc:/usr/local/tomcat/webapps# ls

# 将webapps.dist下的文件拷贝到 tomcat中
root@29c525f703dc:/usr/local/tomcat# cp -r webapps.dist/* webapps/
root@29c525f703dc:/usr/local/tomcat# cd webapps
root@29c525f703dc:/usr/local/tomcat/webapps# ls
ROOT  docs  examples  host-manager  manager
root@29c525f703dc:/usr/local/tomcat/webapps#

# 提交一个「新镜像」，经过上述操作后webapps目录下有文件的「新镜像」
[will@master ~]$ docker commit -m="add files into webapps" -a="will" 29c525f703dc tomcat_add_webapps:1.0
```

```
sha256:c6ef44f81d8de11cc221ce0a356610cda94a0d79bec62e0904f713f982e548cc

# 使用docker images查看刚刚 docker commit的镜像
```

八、容器数据卷

数据持久化、数据可以存储在本地、容器之间数据共享

目录挂载。讲容器内的目录，挂在到Linux上面。

容器的持久化和数据同步，容器间的数据共享

8.1 使用数据卷

挂载数据卷，可以理解为C++中的引用，「双向绑定」。或者理解成，通过地址映射，实现的一种数据同步技术。

实现了数据卷的挂载后，可以通过linux向容器中传文件，也可以把docker的文件同步到linux中。

> 方式一： 直接使用命令来挂载 -v

```
docker run -it -v 主机目录:容器目录
```

> 使用「数据卷」的好处：
>
> 以后对容器的修改只需要在主机本地完成，容器内会自动同步。

8.2 docker volume命令

> 挂载

```
# 匿名挂载、具名挂载、指定路径挂载
 -v 容器内路径                    # 匿名挂载
 -v 卷名:容器内路径               # 具名挂载
 -v 宿主机路径:容器内路径         # 指定路径挂载

# 读写权限
-v 宿主机路径:容器内路径:[ro/rw]
ro   readonly    # 只读：只能通过主机来操作，容器内无法操作修改
rw   readwrite   # 可读可写
```

8.3 数据卷与DockerFile（挂载数据卷的第二种方式）

> 挂载数据卷
>
> 1. -v 命令
> 2. dockerfile

DockerFile用来构建docker镜像，编写命令脚本实现镜像。

```
# dockerfile文件，名字可以随机，但是最好使用 dockerfile
# dockerfile编写，指令（大写）+ 参数
FROM centos


VOLUME ["volume01","volume02"]

CMD echo "--- end ---"
CMD /bin/bash
```

8.3 数据卷容器

功能：两个mysql同步数据、容器之间数据同步和共享

容器B通过「数据卷挂载」至容器A，实现两个容器的数据同步，「被挂载的容器」为「父容器」。

```
# 挂载「父容器/数据卷容器」
 --volumes-from [容器id/容器名称]
```

> 【数据拷贝】只要有一个还在用共享的数据卷，那么「删除父容器/子容器」都是不会造成「数据丢失」。

> 【数据持久化】只要「父容器/子容器」将数据挂载到了本地，那么删除所有容器也不会造成数据丢失。

九、DockerFile

9.1 说明

dockerfile时用来构建docker镜像文件。

构建步骤：

1. 编写一个dockerfile文件
2. docker build构建成为一个镜像
3. docker run 运行镜像
4. docker push 发布镜像

> centos7-官方镜像的dockerfile

```
FROM scratch
ADD centos-7-x86_64-docker.tar.xz /

LABEL \
    org.label-schema.schema-version="1.0" \
    org.label-schema.name="CentOS Base Image" \
    org.label-schema.vendor="CentOS" \
    org.label-schema.license="GPLv2" \
    org.label-schema.build-date="20200504" \
    org.opencontainers.image.title="CentOS Base Image" \
    org.opencontainers.image.vendor="CentOS" \
    org.opencontainers.image.licenses="GPL-2.0-only" \
```

```
    org.opencontainers.image.created="2020-05-04 00:00:00+01:00"

CMD ["/bin/bash"]
```

9.2 dockerfile构建

9.2.1 docker file指令

> 所有的指令都是「大写字母」

```
FROM                    # 基础镜像，一切从这里开始构建
MAINTAINER              # 镜像作者，姓名 + 邮箱
RUN                     # 镜像构建时，需要运行的命令
ADD                     # 步骤：如果添加tomcat镜像，添加一个压缩包。tomcat压缩包就是添加的内容
WORKDIR                 # 镜像的工作目录
VOLUME                  # 挂载的目录
EXPOSE                  # 端口配置
CMD                     # 指定这个容器启动时，需要运行的命令，只有最后一个会生效可被替代
ENTRYPOINT              # 指定容器启动的时候要运行的命令，可以追加命令
ONBUILD                 # 当构建一个被继承的dockerfile时，会运行 ONBUILD指令。「触发指令」
COPY                    # 类似ADD，将文件拷贝到镜像中
ENV                     # 构建镜像时，设置环境变量 （key-value形式）
```

> dockerfile-编写：
>
> 1. 添加「基准镜像」
> 2. 配置「运行环境」
>   - 下载&安装软件
>   - 添加「环境软件」
>   - 执行命令

> 3. 设置「容器端口」
> 4. 运行「相关指令」

9.3 docker build构建镜像

```
# 命令 doker build -f [dockerfile文件路径] -t [镜像名称:[tag]]
```

9.4 CMD 和 ENTRYPOINT区别

```
# dockerfile
FROM centos
CMD ["ls","-a"]                 # 执行: ls -a

# 镜像构建命令 - 略 得到 cetostest

# 使用 docker run centostest -l
# 启动容器时，并执行 -l命令（追加 -l）
# 报错:
    # 原因:  启动时，追加的「-l」命令，替换了CMD中的「ls -a」；但是仅仅「-l」并不是是一个命令
```

> 【CMD 】在docker run运行容器时，如果追加了命令，会替换镜像中原本的CMD 指令。如果想要「追加命令」，必须使用「完全的命令」。

```
# dockerfile
FROM centos
ENRTYPOINT ["ls", "-a"]

# 略 - docker build ==> centostest02

# docker run centostest02 -l
# 可追加命令，追加后，不会报错
```

9.5 将镜像发布到docker hub中

略

9.6 将镜像发不到「阿里云容器服务」中

1. 登陆阿里云
2. 找到「容器镜像服务」
3. 创建「命名空间」
4. 创建「镜像仓库」（推荐：私有）
5. 浏览「阿里云文档」，尝试上传

十、【小结】

dockerfile ---build---> **image** ---run---> **container** ---commit---
>**new_image**

docker hub---pull--->**image**---run--->**container**---commit---
>**new_image**

.....

十一、Docker 网络（容器互联）

11.1 Docker0

> 查看主机ip

```
ip addr
# 得到三种结果
lo          # 本地回环地址
enth        # 本机地址
docker0     # docker0 地址
```

> docker 如何处理网络访问?

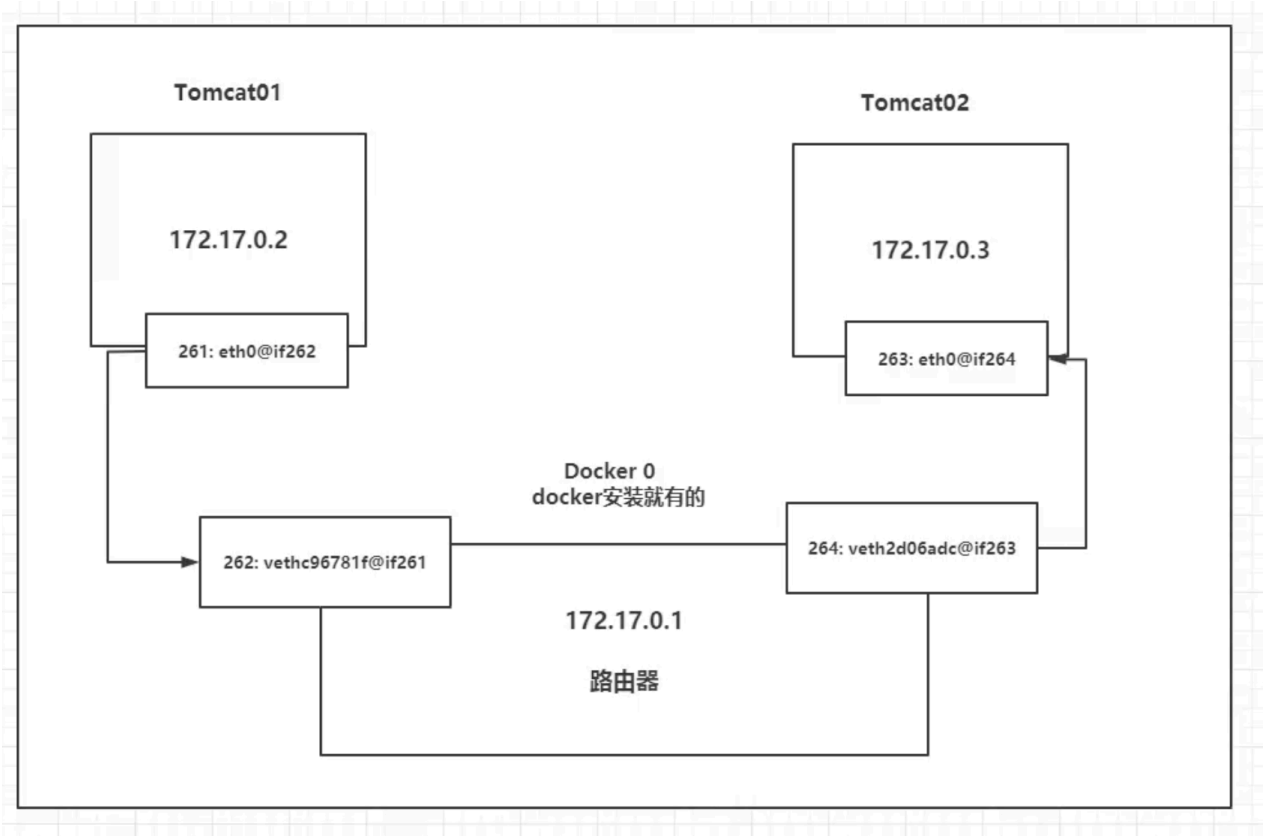
```
# 以tomcat容器为例子
docker run -d -P --name tomcat01 tomcat

# ip addr 查看容器的内部网络地址
docker exec -it tomcat01 ip addr

# 通过ip addr发现，容器启动的时候，会得到
 eth0@if7
 inet 172.17.0.2

# 使用linux本地主机，ping tomcat01容器，结果是可以成功的

```

> 原理

每启动一个docker容器，docker就会给docker容器分配一个ip。我们只要安装了docker，就会有一个网卡`「docker0 」以桥接模式工作，使用「veth-pair技术」。

```
# 我们通过测试发现，每启动一个容器，本地linux主机和容器内部都会多一个网卡，两个网卡是成对出现的。
# veth-pair 就是一对的虚拟设备接口，成对出现，一端连接协议，一端彼此相连
# 通过veth-pair特性，可以使其充当桥梁，连接各种虚拟网络设备
```

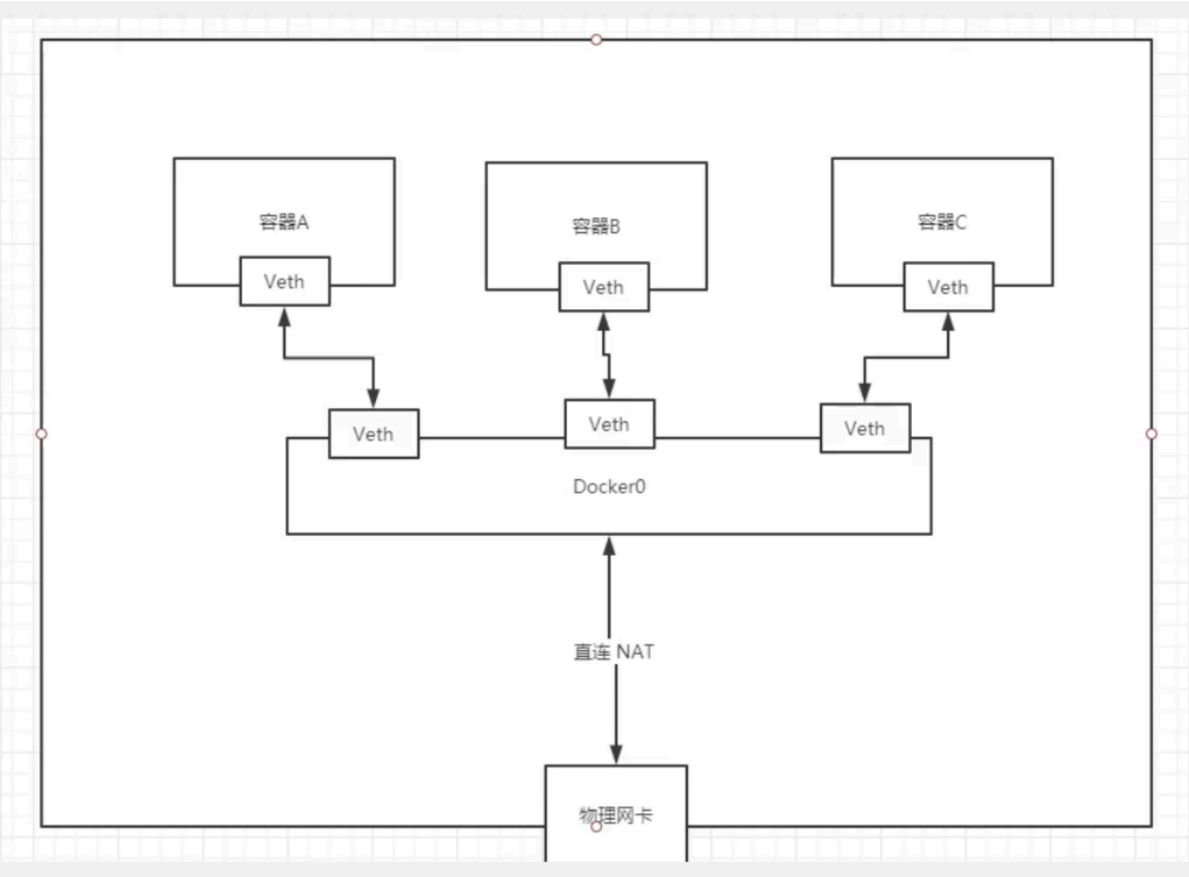
容器和容器之间也是可以互相ping通的。



结论：tomcat01和tomcat02共用一个路由器，也就是docker0。

所有的容器，在不指定网络的情况下，都是docker0路由的，docker会给我们的容器分配一个默认的可用ip

> 小结
>
> Docker使用的是Linux的桥接，宿主机是一个docker容器的网桥 --- docker0。
>
> 只要删除了容器，对应的网桥也就消失了。



11.2 --link

不根据docker容器ip，直接通过「容器名」实现容器间的ping通。

```
# 命令：  --link 容器名称
#    正向ping可以，但是反向ping却不行。
# --link 就是在/etc/hosts中增加列一个「 ip 容器名称 容器id」
```

--link 方法过时。

11.3 自定义网络

> docker network ls --- 查看所有的docker网络

```
[will@master]~% docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
305400d9307e        bridge              bridge              local
66e804107da2        host                host                local
f7bb4d8c0c5a        none                null                local
```

网络模式

bridge：桥接（默认 or 自己创建）。容器之间的通信通过docker0（ip为01）实现。

none： 不配置网络。

host：主机模式，和宿主机共享网络。

container：容器网络连通！（用的少！局限大！）

11.4 docker网络命令

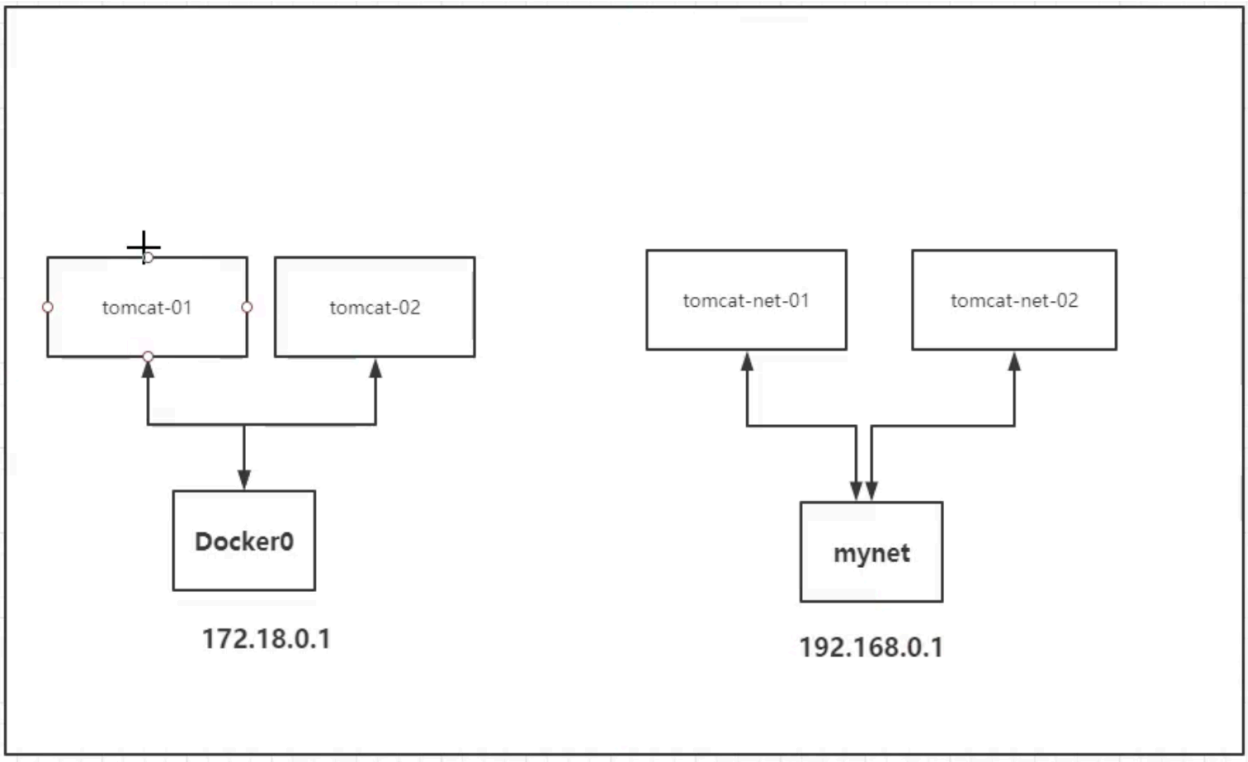
平时在启动docker容器的时候，会默认添加`--net bridge`和`docker0`实现互通（添加到默认网络中）。

```
# 创建自定义网络
# --driver bridge             设置「网络模式」（bridge桥接）
# --subnet 192.168.0.0/16     设置「子网掩码」（范围：192.168.0.2 ~ 192.168.255.255）
# --gatway 192.168.0.1        设置「网关」（所有的请求都经过的地方-网卡的地址）
docker network create --driver bridge --subnet 198.168.0.0/16 --gatway 192.168.0.1 [网络名称]
```

> 自定义网络中的容器，可以通过「容器名称」互相ping，而且不需要预先--link设置。

> 「好处」：不同的集群使用不同的网络，保证集群之间是安全健康的

11.5 网络互通



> 实现「docker0」和「mynet」之间互通

```
#「docker0」中所有容器，可以ping通「mynet」
#「mynet」中所有容器，可以ping通「docker01」中所有容器
# 并不是「docker0」和「mynet」之间可以ping通
```

```
# 命令 docker connect [网络名称] [容器名称]
docker connect mynet tomcat01
```

「tomcat01容器」和「mynet网络」的互通，就是将「tomcat01容器」加到了「mynet网络」中。做到了，一个容器两个ip，一个ip在「docker0网络」，一个ip在「tomcat01网络」。

```
# tomcat01 能够ping通 tomcat-net-01、tomcat-net-02
# tomcat02 无法ping通 tomcat-net-01、tomcat-net-02（tomcat02没加入mynet）
```

十二、docker-compose

```
# 命令：重新编译yml，然后再启动
docker-compose up --build
# -d 后台运行

# 查看docker-compose管理的进程
docker-compose ps

# 删除所有的docker-compose资源（网卡、容器....）
docekr-compose down

# 删除所有的docker-compose资源 + 镜像
docker-compose down --rmi all
```

参考

> https://www.bilibili.com/video/BV1og4y1q7M4