马哥教育
IT人的高薪职业学院

企业级容器管理 Docker

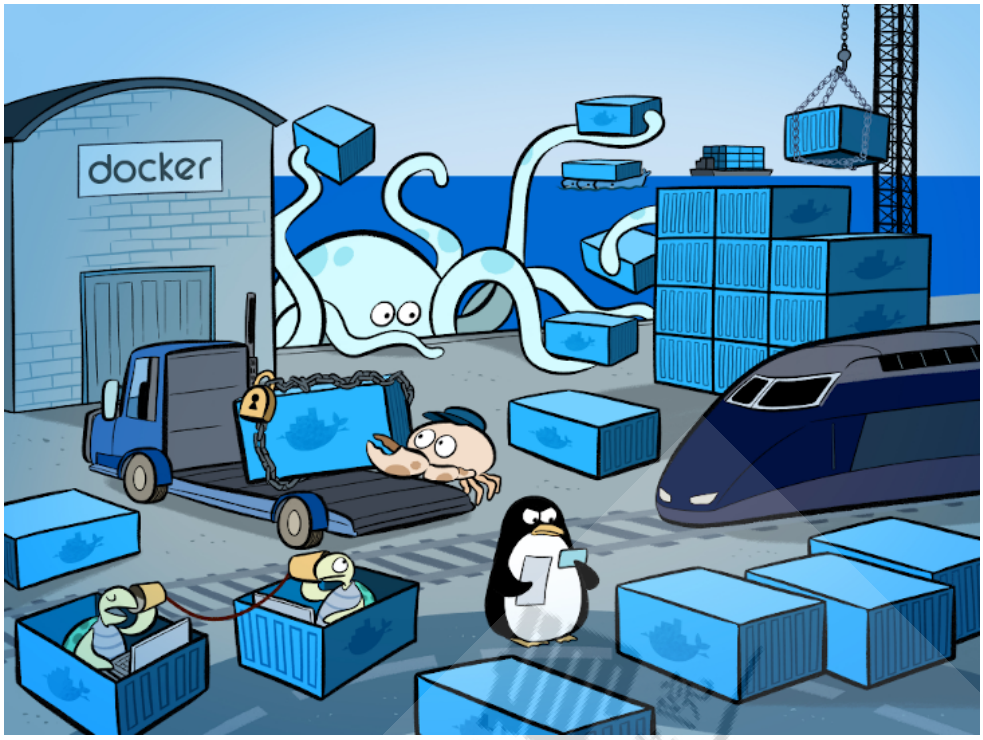
讲师：王晓春

企业级容器技术 Docker

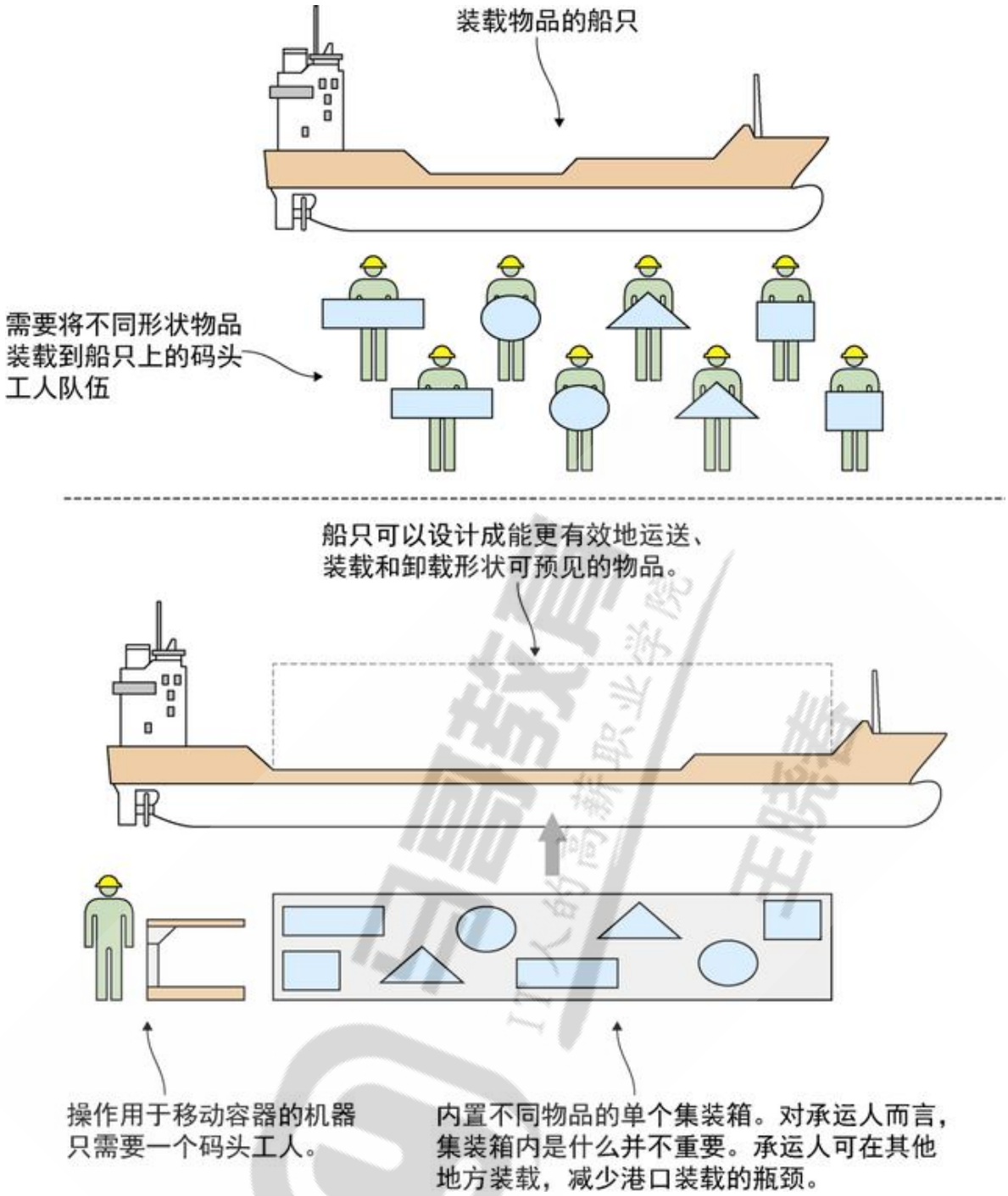
内容概述

- **docker 介绍和基本操作**
- **容器镜像制作和管理**
- **数据卷管理**
- **网络管理**
- **镜像仓库管理**
- **容器编排管理工具docker compose**
- **容器资源限制**
- **可视化容器管理工具Portainer**

1 Docker 介绍和基础操作



Container 即容器，平时生活中指的是可以装下其它物品的工具， 以方便人类归纳放置物品 、存储和异地运输 ，比如人类使用的衣柜 、行李箱、 背包等可以成为容器，Container 除了容器以外，另一个意思是集装箱, 很多码头工人将很多装有不同物品但却整齐划一的箱子装载到停靠在岸边大船，然后方便的运来运去。



为什么这些集装箱可以很方便的运来运去呢？因为它们大小一致标准化尺寸的箱子，并且可以安全的隔离开,所以当我们使用 Container 来形容容器的时候，就是我们想要让容器达到一个可以打包，符合标准的状态。

集装箱标准尺寸重量

类型	大小	内长m	内宽m	内高m	门高m	门宽m	容积m³	载重T	皮重T
普通箱	20GP	5.898	2.352	2.385	2.280	2.343	28.00	18	2.3
普通箱	40GP	12.032	2.352	2.385	2.280	2.343	57.00	28	3.4
高箱	40HC	12.032	352	2.690	2.585	2.343	67.00	28	4.0

但今天我们所说的容器是一种 IT 技术。容器其实是一种沙盒技术。顾名思义，沙盒就是能够像一个集装箱一样，把你的应用装起来。这样，应用与应用之间就有了边界而不会相互干扰;同时装在沙盒里面的应用，也可以很方便的被搬来搬去，这也是 PaaS 想要的最理想的状态(**可移植性,标准化,隔离性**)。

容器是软件工业上的集装箱的技术，集装箱的标准化，减少了包装成本，大大提高货物运输和装卸效率，是传统运输行业的重大变革。早期的软件项目中软件更新，发布低效，开发测试发布周期很长，很难敏捷。有了容器技术，就可以利用其标准化的特点，大幅提高生产效率。

容器技术是虚拟化、云计算、大数据之后的一门新兴的并且是炙手可热的新技术， 容器技术提高了硬件资源利用率、 方便了企业的业务快速横向扩容（可以达到秒级快速扩容）、 实现了业务宕机自愈功能（配合K8S可以实现，但OpenStack无此功能），因此未来数年会是一个容器愈发流行的时代，这是一个对于 IT 行业来说非常有影响和价值的技术，而对于IT行业的从业者来说， 熟练掌握容器技术无疑是一个很有前景的行业工作机会。

1.1 Docker 介绍

1.1.1 容器历史

虽然 docker 把容器技术推向了巅峰，但容器技术却不是从 docker 诞生的。实际上，容器技术连新技术都算不上，因为它的诞生和使用确实有些年头了。下面的一串名称可能有的你都没有听说过，但它们的确都是容器技术的应用:

1、Chroot Jail
就是我们常见的 chroot 命令的用法。它在 1979 年的时候就出现了，被认为是最早的容器化技术之一。它可以把一个进程的文件系统隔离起来。

2、The FreeBSD Jail
Freebsd Jail (监狱)实现了操作系统级别的虚拟化，它是操作系统级别虚拟化技术的先驱之一。2000年，伴随FreeBSD4.0版的发布

3、Linux VServer
使用添加到 Linux 内核的系统级别的虚拟化功能实现的专用虚拟服务器。允许创建许多独立的虚拟专用服务器（VPS），这些虚拟专用服务器在单个物理服务器上全速同时运行，从而有效地共享硬件资源。VPS提供与传统Linux服务器几乎相同的操作环境。可以在这样的VPS上启动所有服务（例如ssh，邮件，Web和数据库服务器），而无需（或者在特殊情况下只需进行很少的修改），就像在任何真实服务器上一样。

每个VPS都有自己的用户帐户数据库和root密码，并且与其他虚拟服务器隔离，但它们共享相同的硬件资源

2003年11月1日 VServer 1.0 发布

官网:http://linux-vserver.org/

4、Solaris Containers
它也是操作系统级别的虚拟化技术，专为 X86 和 SPARC 系统设计。Solaris 容器是系统资源控制和通过 "区域" 提供边界隔离的组合。

5、OpenVZ
OpenVZ 是一种 Linux 中操作系统级别的虚拟化技术。 它允许创建多个安全隔离的 Linux 容器，即 VPS。

6、Process Containers
Process 容器由 Google 的工程师开发，一般被称为 cgroups。

7、LXC
LXC为Linux Container的简写。可以提供轻量级的虚拟化，以便隔离进程和资源，而且不需要提供指令解释机制以及全虚拟化的其他复杂性。容器有效地将由单个操作系统管理的资源划分到孤立的组中，以更好地在孤立的组之间平衡有冲突的资源使用需求

Linux Container提供了在单一可控主机节点上支持多个相互隔离的server container同时执行的机制。Linux Container有点像chroot，提供了一个拥有自己进程和网络空间的虚拟环境，但又有别于虚拟机，因为lxc是一种操作系统层次上的资源的虚拟化。

8、Warden
在最初阶段，Warden 使用 LXC 作为容器运行时。 如今已被 CloudFoundy 取代。

9、LMCTFY
LMCTY 是 Let me contain that for you 的缩写。它是 Google 的容器技术栈的开源版本。Google 的工程师一直在与 docker 的 libertainer 团队合作，并将 libertainer 的核心概念进行抽象并移植到此项目中。该项目的进展不明，估计会被 libcontainer 取代。

10、Docker

Docker 是一个可以将应用程序及其依赖打包到几乎可以在任何服务器上运行的容器的工具。

11、RKT
RKT 是 Rocket 的缩写，它是一个专注于安全和开放标准的应用程序容器引擎。

综上所述正如我们所看到的，docker 并不是第一个容器化技术，但它的确是最知名的一个。

1.1.2 Docker 是什么

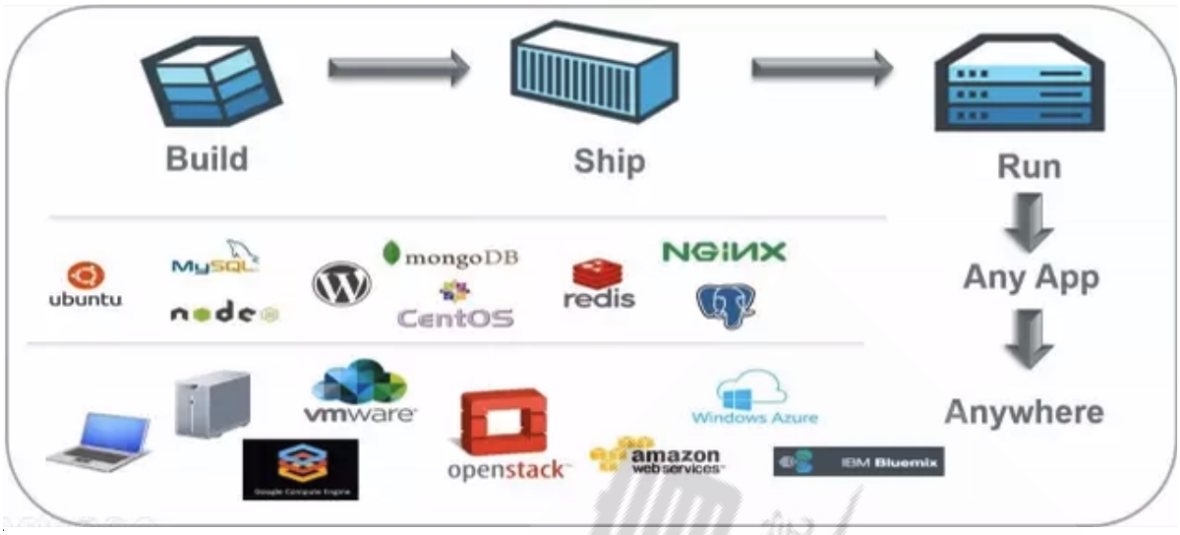
Docker （码头工人）是一个开源项目，诞生于 2013 年初，最初是 dotCloud 公司（后由于 Docker 开源后大受欢迎就将公司改名为 Docker Inc ，总部位于美国加州的旧金山）内部的一个开源的 PAAS 服务 (Platform as a ServiceService)的业余项目。它基于 Google 公司推出的 Go 语言实现。 项目后来加入了 Linux 基金会，遵从了 Apache 2.0 协议，项目代码在 GitHub 上进行维护。

Docker 是基于 linux 内核实现，Docker 最早采用 LXC 技术， LXC 是 Linux 原生支持的容器技术 ，可以提供轻量级的虚拟化 ，可以说 docker 就是基于 LXC 发展起来 的，提供 LXC 的高级封装，标准的配置方法，在LXC的基础之上，docker提供了一系列更强大的功能。而虚拟化技术 KVM(KernelKernel-based Virtual Machine Machine) 基于 模块实现， 后来Docker 改为自己研发并开源的 runc 技术运行容器，彻底抛弃了LXC。

Docker 相比虚拟机的交付速度更快，资源消耗更低，Docker 采用客户端/服务端架构，使用远程API来管理和创建容器，其可以轻松的创建一个轻量级的、可移植的、自给自足的容器，docker 的三大理念是build(构建)、ship(运输)、 run(运行)，Docker遵从apache 2.0协议，并通过（namespace及cgroup等）来提供容器的资源隔离与安全保障等，所以Docke容器在运行时不需要类似虚拟机（空运行的虚拟机占用物理机6-8%性能）的额外资源开销，因此可以大幅提高资源利用率,总而言之Docker是一种用了

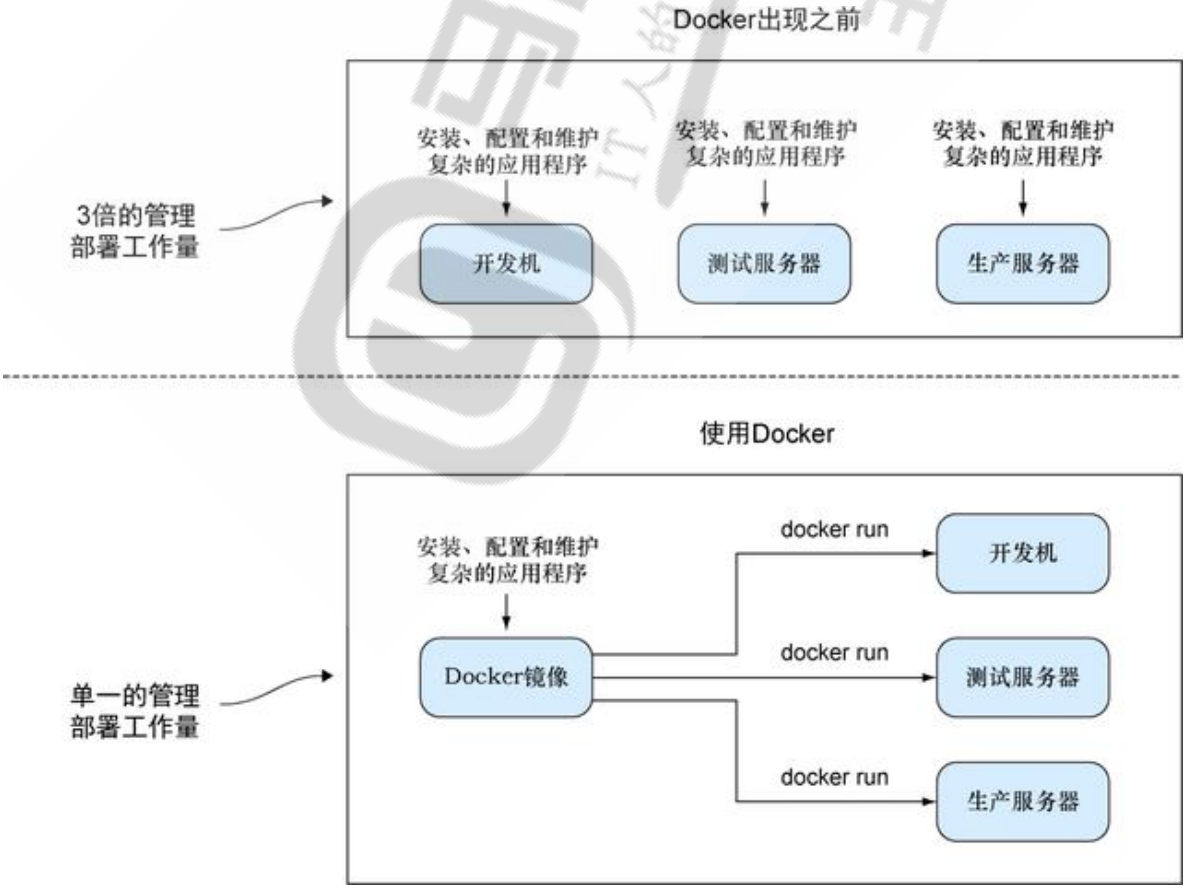
新颖方式实现的轻量级虚拟机.类似于VM但是在原理和应用上和VM的差别还是很大的，并且docker的专业叫法是应用容器(Application Container)。

Docker的主要目标



Build, Ship and Run Any App, Anywhere，即通过对应用组件的封装（Packaging）、分发（Distribution）、部署（Deployment）、运行（Runtime）等生命周期的管理，达到应用组件级别的“一次封装，到处运行”。这里的应用组件，既可以是一个Web应用，也可以是一套数据库服务，甚至是一个操作系统。将应用运行在Docker 容器上，可以实现跨平台，跨服务器，只需一次配置准备好相关的应用环境，即可实现到处运行，保证研发和生产环境的一致性，解决了应用和运行环境的兼容性问题，从而极大提升了部署效率，减少故障的可能性

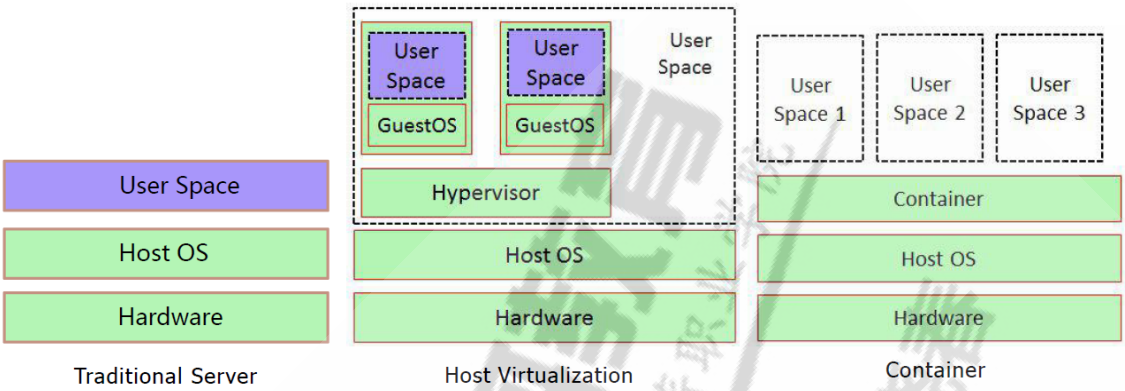
使用Docker 容器化封装应用程序的意义:



- 统一基础设施环境-docker环境
  - 硬件的组成配置
  - 操作系统的版本

    - 运行时环境的异构
- 统一程序打包（装箱）方式-docker镜像
    - java程序
    - python程序
    - nodejs程序
- 统一程序部署（运行）方式-docker容器
    - java-jar...→ docker run...
    - python manage.py runserver... → docker run...
    - npm run dev ... → docker run...

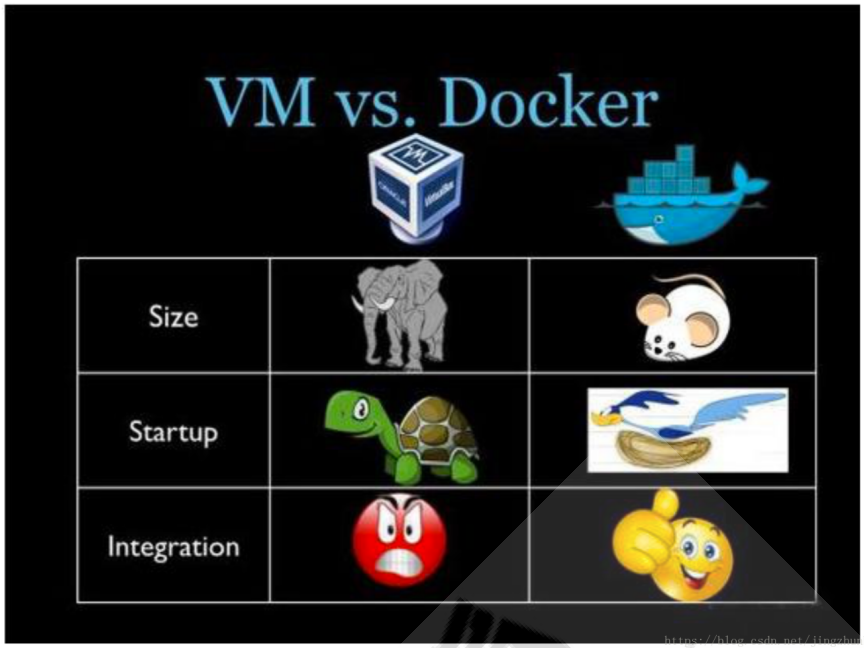
1.1.3 Docker 和虚拟机，物理主机



容器和虚拟机技术比较

- 传统虚拟机是虚拟出一个主机硬件,并且运行一个完整的操作系统 ,然后在这个系统上安装和运行软件
- 容器内的应用直接运行在宿主机的内核之上,容器并没有自己的内核,也不需要虚拟硬件,相当轻量化
- 每个容器间是互相隔离,每个容器内都有一个属于自己的独立文件系统,独立的进程空间,网络空间,用户空间等,所以在同一个宿主机上的多个容器之间彼此不会相互影响

容器和虚拟机表现比较



- 资源利用率更高：开销更小,不需要启动单独的虚拟机OS内核占用硬件资源,可以将服务器性能压榨至极致.虚拟机一般会有5-20%的损耗,容器运行基本无损耗,所以生产中一台物理机只能运行数十个虚拟机，但是一般可以运行数百个容器
- 启动速度更快：可以在数秒内完成启动
- 占用空间更小: 容器一般占用的磁盘空间以MB为单位,而虚拟机以GB
- 集成性更好：和CI/CD（持续集成/持续部署）相关技术结合性更好，实现打包镜像发布测试可以一键运行,做到自动化并快速的部署管理,实现高效的开发生命周期

使用虚拟机是为了更好的实现服务运行环境隔离，每个虚拟机都有独立的内核，虚拟化可以实现不同操作系统的虚拟机，但是通常一个虚拟机只运行一个服务，很明显资源利用率比较低且造成不必要的性能损耗，我们创建虚拟机的目的是为了运行应用程序，比如Nginx、PHP、Tomcat等web程序，使用虚拟机无疑带来了一些不必要的资源开销，但是容器技术则基于减少中间运行环节带来较大的性能提升。

根据实验，一个运行着CentOS的KVM虚拟机启动后，在不做优化的情况下，虚拟机自己就需要占用100~200 MB内存。此外，用户应用运行在虚拟机里面，它对宿主机操作系统的调用就不可避免地要经过虚拟化软件的拦截和处理，这本身又是一层性能损耗，尤其对计算资源、网络和磁盘I/O的损耗非常大。

比如：一台96G内存的物理服务器，为了运行java程序的虚拟机一般需要分配8G内存/4核的资源，只能运行13台左右虚拟机，但是改为在docker容器上运行Java程序,每个容器只需要分配4G内存即可，同样的物理服务器就可以运行25个左右容器，运行数量相当于提高一倍，可以大幅节省IT支出，通常情况下至少可节约一半以上的物理设备

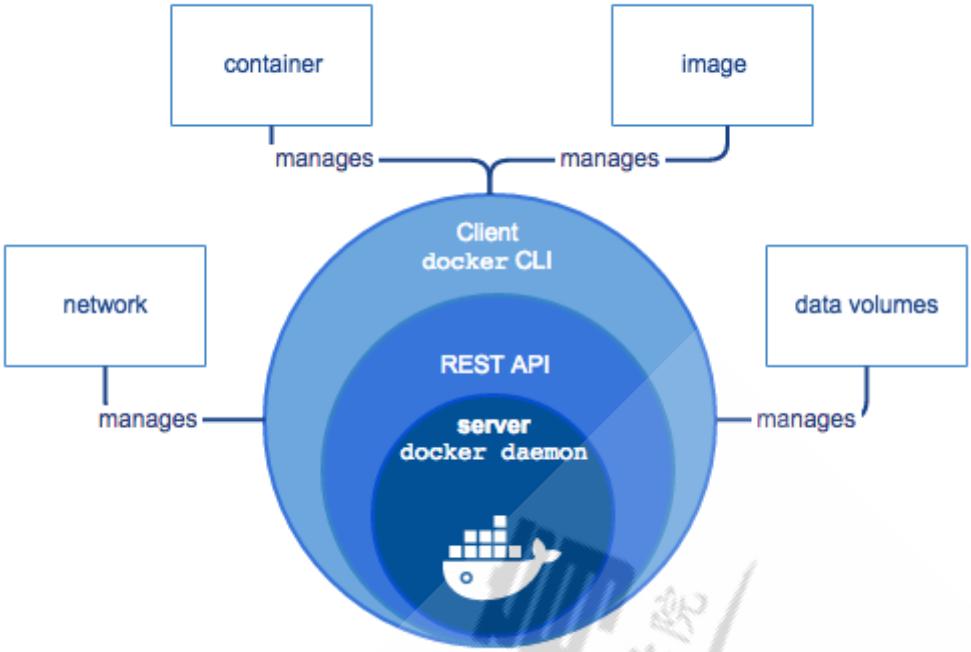
1.1.4 Docker 的组成

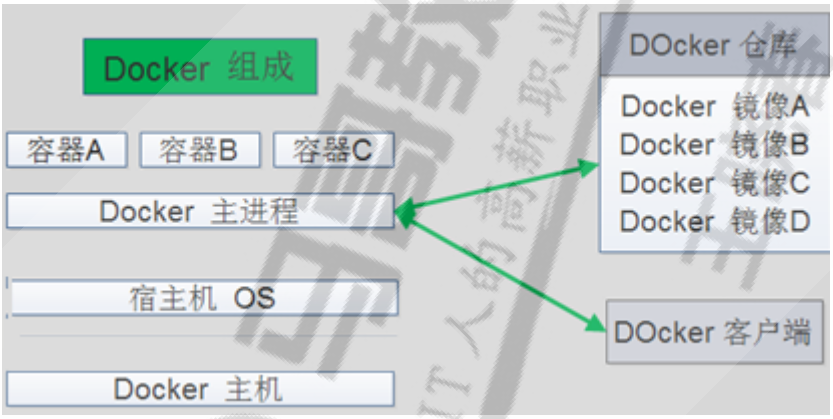
docker 官网: http://www.docker.com

帮助文档链接: https://docs.docker.com/

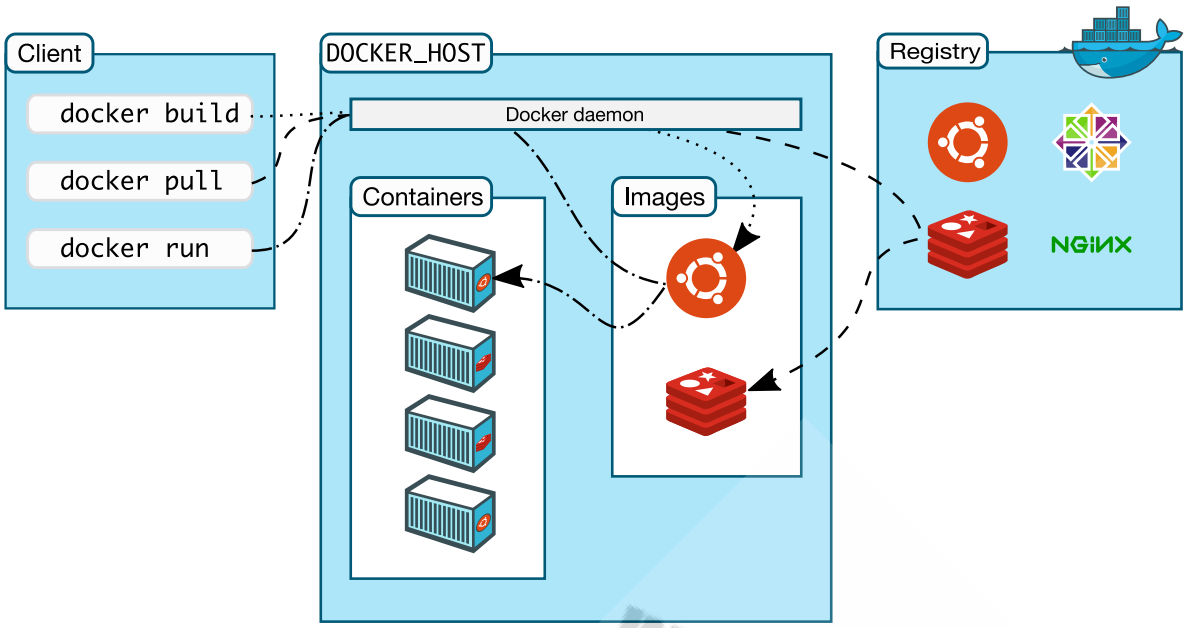
docker 镜像: https://hub.docker.com/

docker 中文网站: http://www.docker.org.cn/





- Docker 主机(Host): 一个物理机或虚拟机，用于运行Docker服务进程和容器，也称为宿主机，node节点
- Docker 服务端(Server): Docker守护进程，运行docker容器
- Docker 客户端(Client): 客户端使用 docker 命令或其他工具调用docker API
- Docker 镜像(Images): 镜像可以理解为创建实例使用的模板,本质上就是一些程序文件的集合
- Docker 仓库(Registry): 保存镜像的仓库，官方仓库: https://hub.docker.com/，可以搭建私有仓库harbor
- Docker 容器(Container): 容器是从镜像生成对外提供服务的一个或一组服务,其本质就是将镜像中的程序启动后生成的进程



1.1.5 Namespace

一个宿主机运行了N个容器，多个容器共用一个 OS，必然带来的以下问题:

- 怎么样保证每个容器都有不同的文件系统并且能互不影响?
- 一个docker主进程内的各个容器都是其子进程，那么如果实现同一个主进程下不同类型的子进程？各个容器子进程间能相互通信(内存数据)吗?
- 每个容器怎么解决IP及端口分配的问题?
- 多个容器的主机名能一样吗?
- 每个容器都要不要有root用户？怎么解决账户重名问题?

namespace是Linux系统的底层概念，在内核层实现，即有一些不同类型的命名空间被部署在核内，各个docker容器运行在同一个docker主进程并且共用同一个宿主机系统内核，各docker容器运行在宿主机的用户空间，每个容器都要有类似于虚拟机一样的相互隔离的运行空间，但是容器技术是在一个进程内实现运行指定服务的运行环境，并且还可以保护宿主机内核不受其他进程的干扰和影响，如文件系统空间、网络空间、进程空间等，目前主要通过以下技术实现容器运行空间的相互隔离:

隔离类型	功能	系统调用参数	内核版本
MNT Namespace(mount)	提供磁盘挂载点和文件系统的隔离能力	CLONE_NEWNS	2.4.19
IPC Namespace(Inter-Process Communication)	提供进程间通信的隔离能力,包括信号量,消息队列和共享内存	CLONE_NEWIPC	2.6.19
UTS Namespace(UNIX Timesharing System)	提供内核,主机名和域名隔离能力	CLONE_NEWUTS	2.6.19
PID Namespace(Process Identification)	提供进程隔离能力	CLONE_NEWPID	2.6.24
Net Namespace(network)	提供网络隔离能力,包括网络设备,网络栈,端口等	CLONE_NEWNET	2.6.29
User Namespace(user)	提供用户隔离能力,包括用户和组	CLONE_NEWUSER	3.8

1.1.5.1 MNT Namespace

每个容器都要有独立的根文件系统有独立的用户空间，以实现在容器里面启动服务并且使用容器的运行环境，即一个宿主机是ubuntu的服务器，可以在里面启动一个centos运行环境的容器并且在容器里面启动一个Nginx服务，此Nginx运行时使用的运行环境就是centos系统目录的运行环境，但是在容器里面是不能访问宿主机的资源，宿主机是使用了chroot技术把容器锁定到一个指定的运行目录里面。

例如:

```
/var/lib/containerd/io.containerd.runtime.v1.linux/moby/容器ID
```

根目录:

```
/var/lib/docker/overlay2/ID
```

范例:

```
#启动三个容器用于以下验证过程:
[root@ubuntu1804 ~]#docker version
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:29:52 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
```

```
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:28:22 2019
 OS/Arch:           linux/amd64
 Experimental:      false
containerd:
 Version:           1.2.10
 GitCommit:         b34a5c8af56e510852c35414db4c1f4fa6172339
runc:
 Version:           1.0.0-rc8+dev
 GitCommit:         3e425f80a8c931f88e6d94a8c831b9d5aa481657
docker-init:
 Version:           0.18.0
 GitCommit:         fec3683



[root@ubuntu1804 ~]# docker run  -d --name nginx-1 -p 80:80 nginx
[root@ubuntu1804 ~]# docker run  -d --name nginx-2 -p 81:80 nginx
[root@ubuntu1804 ~]# docker run  -d --name nginx-3 -p 82:80 nginx
```

范例: 查看存储

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                   COMMAND             CREATED
    STATUS              PORTS               NAMES
d2d79c1d3695        centos:centos8.1.1911   "/bin/bash"         14 minutes ago
   Up 14 minutes                           boring_carson
17ff44b1dbff        centos:centos8.1.1911   "/bin/bash"         17 minutes ago
   Up 17 minutes                           interesting_austin
[root@ubuntu1804 ~]#ls /var/lib/containerd/io.containerd.runtime.v1.linux/moby/
17ff44b1dbff94e3578b3d3b74daae54527c1f65a279bb07f00641bda24ba580
d2d79c1d36954642dbab35e19bf75075dc94b66c11626c72ac52910add710204
[root@ubuntu1804 ~]#ls
/var/lib/docker/overlay2/0c45e9ac63195a4562a1b5fcd4089a2ad604418d381557e7c1165da
70263b75b/merged/
bin  dev  etc  home  lib  lib64  lost+found  media  mnt  opt  proc  root  run
sbin  srv  sys  tmp  usr  var
```

范例: 验证容器的根文件系统

```
[root@centos8 ~]#podman exec nginx cat /etc/issue
Debian GNU/Linux 9 \n \l
[root@centos8 ~]#podman exec nginx ls /
bin
boot
data
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
```

```
run
sbin
srv
sys
tmp
usr
var
```

范例: 容器和宿主机共享内核

```
[root@centos8 ~]#podman exec nginx uname -r
4.18.0-147.el8.x86_64
[root@centos8 ~]#uname -r
4.18.0-147.el8.x86_64
```

1.1.5.2 IPC Namespace

一个容器内的进程间通信，允许一个容器内的不同进程的(内存、缓存等)数据访问，但是不能跨容器直接访问其他容器的数据

1.1.5.3 UTS Namespace

UTS namespace（UNIX Timesharing System包含了运行内核的名称、版本、底层体系结构类型等信息）用于系统标识，其中包含了主机名hostname 和域名domainname，它使得一个容器拥有属于自己主机名标识，这个主机名标识独立于宿主机系统和其上的其他容器。

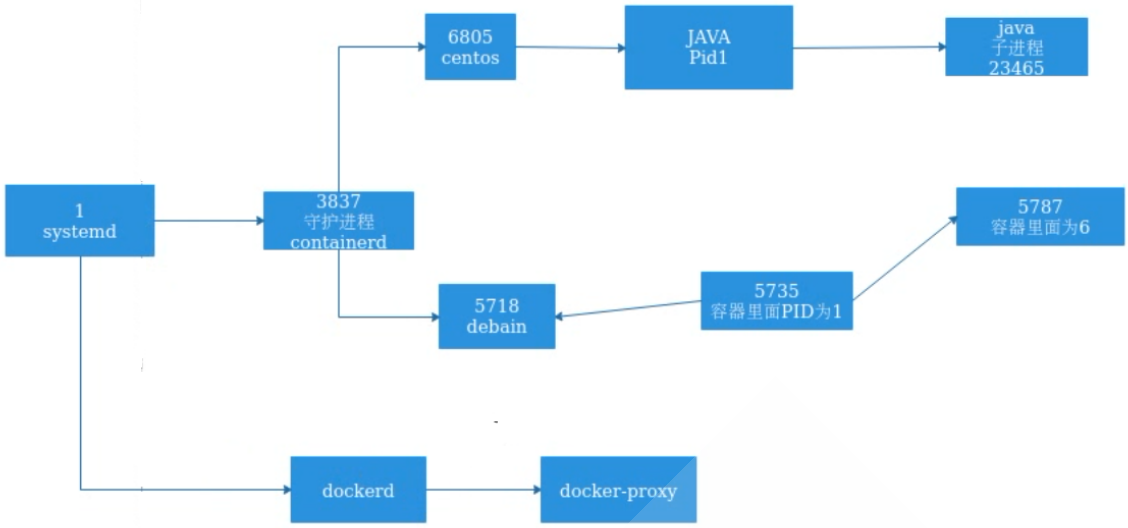
范例:

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                     COMMAND             CREATED
    STATUS              PORTS                NAMES
d2d79c1d3695        centos:centos8.1.1911   "/bin/bash"         34 minutes ago
   Up 34 minutes                           boring_carson
17ff44b1dbff        centos:centos8.1.1911   "/bin/bash"         37 minutes ago
   Up 37 minutes                           interesting_austin
[root@ubuntu1804 ~]#docker exec -it 17ff44b1dbff sh
sh-4.4# hostname
17ff44b1dbff
sh-4.4# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  17ff44b1dbff
sh-4.4# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
60: eth0@if61: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
```

```
       valid_lft forever preferred_lft forever
sh-4.4# uname -r
4.15.0-29-generic
sh-4.4# free -h
              total        used        free      shared  buff/cache   available
Mem:           962Mi       268Mi        81Mi       1.0Mi       612Mi       522Mi
Swap:          1.9Gi        17Mi       1.8Gi
sh-4.4# lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              1
On-line CPU(s) list: 0
Thread(s) per core:  1
Core(s) per socket:  1
Socket(s):           1
NUMA node(s):        1
Vendor ID:           GenuineIntel
CPU family:          6
Model:               60
Model name:          Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz
Stepping:            3
CPU MHz:             2494.237
BogoMIPS:            4988.47
Hypervisor vendor:   VMware
Virtualization type: full
L1d cache:           32K
L1i cache:           32K
L2 cache:            256K
L3 cache:            6144K
NUMA node0 CPU(s):   0
Flags:               fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon nopl xtopology tsc_reliable nonstop_tsc cpuid pni
pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt
tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm cpuid_fault
invpcid_single pti ssbd ibrs ibpb stibp fsgsbase tsc_adjust bmi1 avx2 smep bmi2
invpcid xsaveopt arat arch_capabilities
sh-4.4# exit
exit
[root@ubuntu1804 ~]#uname -r
4.15.0-29-generic
```

1.1.5.4 PID Namespace

Linux系统中，有一个PID为1的进程(init/systemd)是其他所有进程的父进程，那么在每个容器内也要有一个父进程来管理其下属的子进程，那么多个容器的进程通PID namespace进程隔离(比如PID编号重复、器内的主进程生成与回收子进程等)。



范例:

```
[root@ubuntu1804 ~]#docker exec -it 17ff44b1dbff sh
sh-4.4# ping 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.061 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.051 ms
64 bytes from 127.0.0.1: icmp_seq=5 ttl=64 time=0.050 ms
64 bytes from 127.0.0.1: icmp_seq=6 ttl=64 time=0.051 ms
^Z
[1]+  Stopped(SIGTSTP)        ping 127.0.0.1
sh-4.4# ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.3  12024  3172 pts/0    Ss+  10:41   0:00 /bin/bash
root          46  1.2  0.3  12024  3228 pts/1    Ss   11:24   0:00 sh
root          51  0.0  0.2  29460  2280 pts/1    T    11:24   0:00 ping 127.0.0.1
root          52  0.0  0.3  43960  3332 pts/1    R+   11:24   0:00 ps aux
sh-4.4#
[root@ubuntu1804 ~]#pstree -p
systemd(1)─┬─VGAuthService(816)
           ├─accounts-daemon(819)─┬─{accounts-daemon}(828)
           |                      └─{accounts-daemon}(839)
           ├─agetty(887)
           ├─atd(807)
           ├─blkmapd(512)
           ├─containerd(3371)─┬─containerd-shim(12233)─┬─bash(12259)
           |                  |                        ├─sh(13359)───ping(13395)
           |                  |                        ├─{containerd-shim}
(12234)
```

例如: 下图是在一个容器内使用top命令看到的PID为1的进程是nginx: :

```
top - 05:39:50 up 19:27,  0 users,  load average: 0.00, 0.01, 0.05
Tasks:   4 total,   1 running,   3 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.2 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1868692 total,   514628 free,   292976 used,  1061088 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  1418708 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
    1 root      20   0   32648   3244   2436 S   0.0  0.2   0:00.07 nginx
    6 nginx     20   0   33104   1600    340 S   0.0  0.1   0:00.00 nginx
 3839 root      20   0    4272    628    540 S   0.0  0.0   0:00.02 sh
 3844 root      20   0   41032   1776   1308 R   0.0  0.1   0:00.00 top
```

容器内的Nginx主进程与工作进程:

```
# ps -ef | grep nginx
root          1      0  0 Jun30 ?        00:00:00 nginx: master process nginx -g daemon off;
nginx         6      1  0 Jun30 ?        00:00:00 nginx: worker process
root       3846   3839  0 05:40 pts/0    00:00:00 grep nginx
#
```

那么宿主机的PID究竟与容器内的PID是什么关系?

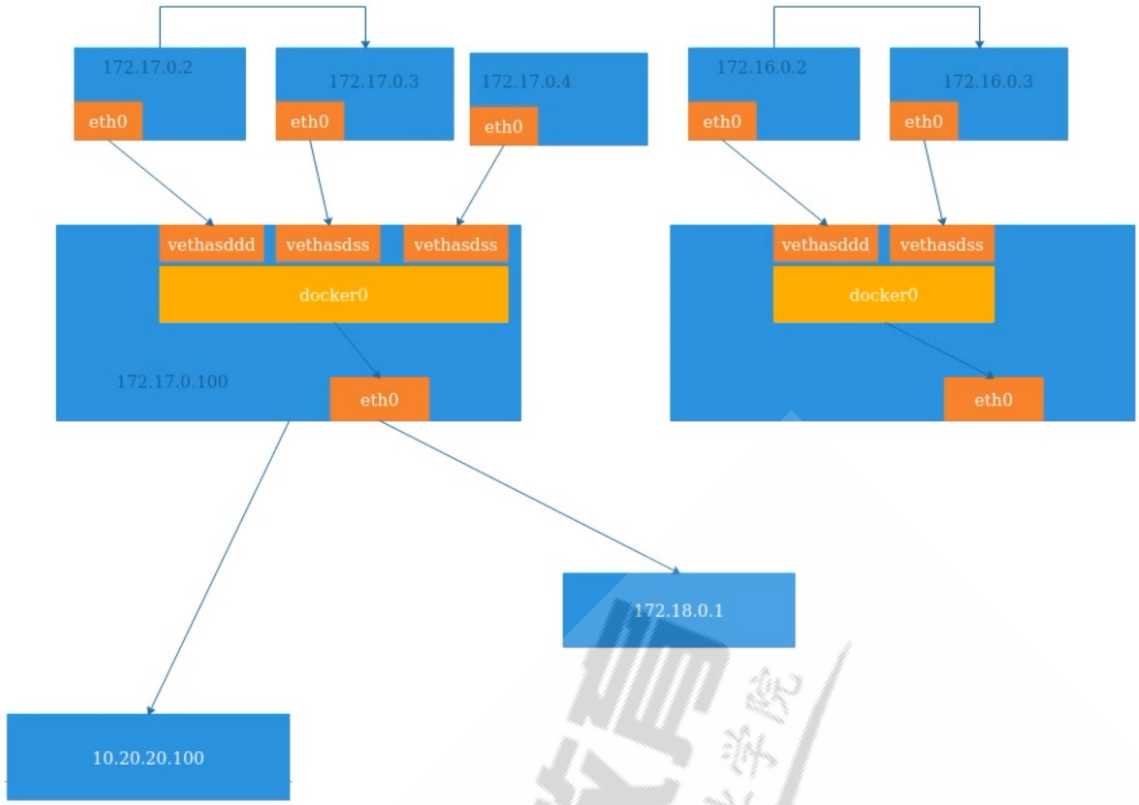
范例: 查看宿主机上的PID信息

```
[root@node1 ~]# ps -ef | grep docker          <-- 查看docker相关的进程
root       5064      1  0 Jun30 ?        00:00:58 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/container
d.sock                                        <-- docker-proxy进程，负责容器访问
root       5487   5064  0 Jun30 ?        00:00:00 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 80
-container-ip 172.17.0.2 -container-port 80
root       5492   5061  0 Jun30 ?        00:00:06 containerd-shim -namespace moby -workdir /var/lib/containerd/io.
containerd.runtime.v1.linux/moby/5a7c6090b8adae677c8cd176f9130ce83d27e60093f9424cbe9a99c86d69c690 -address /run/co
ntainerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
root      26180   5064  0 11:58 ?        00:00:00 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 81
-container-ip 172.17.0.3 -container-port 80
root      26186   5061  0 11:58 ?        00:00:00 containerd-shim -namespace moby -workdir /var/lib/containerd/io.
containerd.runtime.v1.linux/moby/c3f729d7be609297e8b2b1b57364257f286707b431ef5db1bfee899f4dec2d92 -address /run/co
ntainerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
root      26318   5064  0 11:58 ?        00:00:00 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 82
-container-ip 172.17.0.4 -container-port 80
root      26324   5061  0 11:58 ?        00:00:01 containerd-shim -namespace moby -workdir /var/lib/containerd/io.
containerd.runtime.v1.linux/moby/c981ef6a234c5f2201fd90663d06d3fa7331b64a7246da1a835abc8f689a6866 -address /run/co
ntainerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
root      27554  21259  0 13:39 pts/1    00:00:00 docker exec -it 5a7c6090b8adae677c8c sh
root      27797  26765  0 14:40 pts/2    00:00:00 grep --color=auto docker
[root@node1 ~]#
[root@node1 ~]# ps -ef | grep 5061            <-- 查看PID为5061是什么进程
root       5061      1  0 Jun30 ?        00:06:38 /usr/bin/containerd
root       5492   5061  0 Jun30 ?        00:00:06 containerd-shim -namespace moby -workdir /var/lib/containerd/io.
containerd.runtime.v1.linux/moby/5a7c6090b8adae677c8cd176f9130ce83d27e60093f9424cbe9a99c86d69c690 -address /run/co
ntainerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
root      26186   5061  0 11:58 ?        00:00:00 containerd-shim -namespace moby -workdir /var/lib/containerd/io.
containerd.runtime.v1.linux/moby/c3f729d7be609297e8b2b1b57364257f286707b431ef5db1bfee899f4dec2d92 -address /run/co
ntainerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
root      26324   5061  0 11:58 ?        00:00:01 containerd-shim -namespace moby -workdir /var/lib/containerd/io.
containerd.runtime.v1.linux/moby/c981ef6a234c5f2201fd90663d06d3fa7331b64a7246da1a835abc8f689a6866 -address /run/co
ntainerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
root      27799  26765  0 14:42 pts/2    00:00:00 grep --color=auto 5061
[root@node1 ~]#
[root@node1 ~]# ps -ef | grep 5492            <-- 查看其中一个容器的PID信息
root       5492   5061  0 Jun30 ?        00:00:06 containerd-shim -namespace moby -workdir /var/lib/containerd/io.
containerd.runtime.v1.linux/moby/5a7c6090b8adae677c8cd176f9130ce83d27e60093f9424cbe9a99c86d69c690 -address /run/co
ntainerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-root /var/run/docker/runtime-runc
root       5510   5492  0 Jun30 ?        00:00:00 nginx: master process nginx -g daemon off;
root      27574   5492  0 13:39 pts/0    00:00:00 sh
root      27801  26765  0 14:42 pts/2    00:00:00 grep --color=auto 5492
[root@node1 ~]#
```

1.1.5.5 NET Namespace

每一个容器都类似于虚拟机一样有自己的网卡、监听端口、TCP/IP协议栈等，

Docker使用network namespace启动一个vethX接口，这样你的容器将拥有它自己的桥接ip地址，通常是docker0，而docker0实质就是Linux的虚拟网桥,网桥是在OSI七层模型的数据链路层的网络设备，通过mac地址对网络进行划分，并且在不同网络直接传递数据。



查看宿主机的网卡信息:

```
veth1c35fa8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::6ca7:c0ff:fe34:b180  prefixlen 64  scopeid 0x20<link>
        ether 6e:a7:c0:34:b1:80  txqueuelen 0  (Ethernet)
        RX packets 1973  bytes 118913 (116.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2681  bytes 13127422 (12.5 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth2d4714c: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::7cd9:27ff:fec9:a979  prefixlen 64  scopeid 0x20<link>
        ether 7e:d9:27:c9:a9:79  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 648 (648.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

veth6a33bc3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet6 fe80::480a:19ff:fee3:9924  prefixlen 64  scopeid 0x20<link>
        ether 4a:0a:19:e3:99:24  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 648 (648.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@node1 ~]#
```

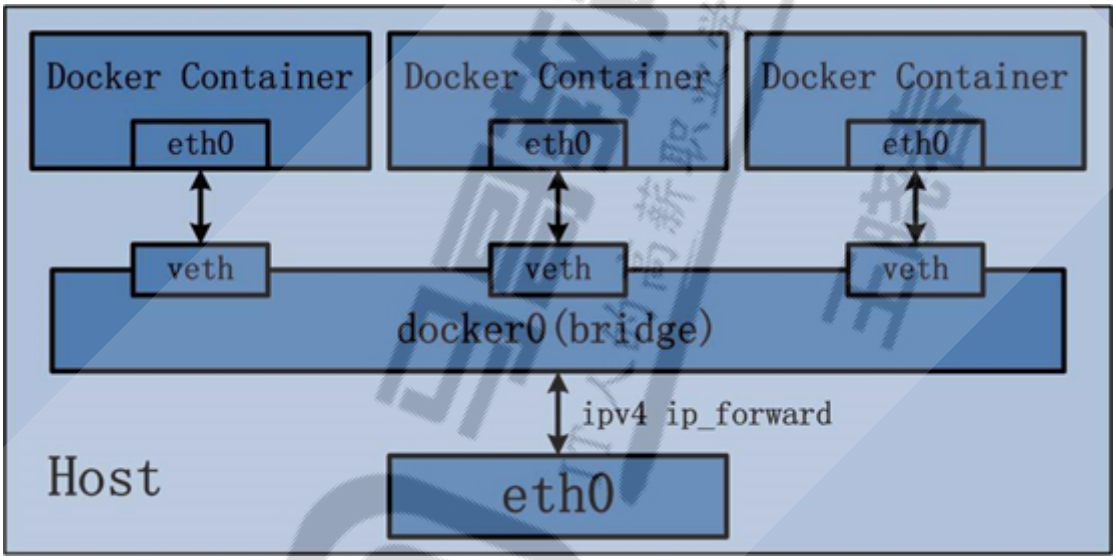
查看宿主机桥接设备:

通过brctl show命令查看桥接设备:

```
[root@node1 ~]# brctl  show
bridge name     bridge id               STP enabled     interfaces
docker0         8000.02429f98ec2b       no              veth1c35fa8
                                                        veth2d4714c
                                                        veth6a33bc3

[root@node1 ~]#
```

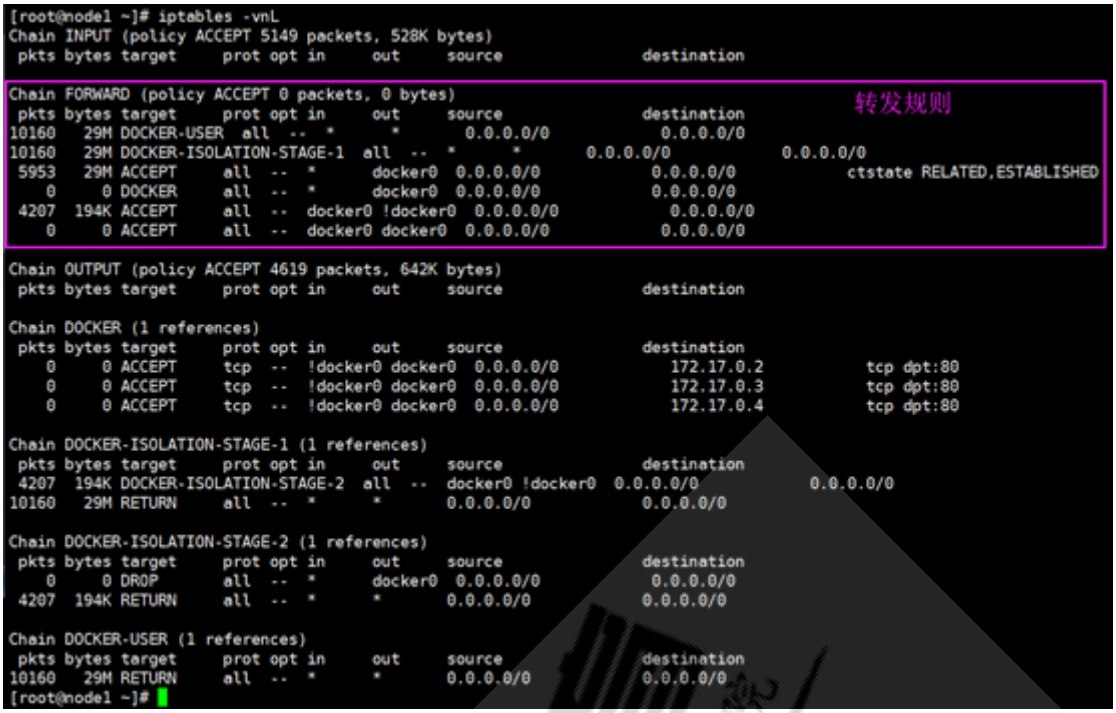
```
[root@node1 ~]# docker exec -it c981ef6a234c sh
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.4  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:04  txqueuelen 0  (Ethernet)
        RX packets 1757  bytes 9262963 (8.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1232  bytes 72606 (70.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 0  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

#
```

逻辑网络图:



宿主机iptables规则:

```
[root@node1 ~]# docker exec -it c981ef6a234c sh
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.4  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:04  txqueuelen 0  (Ethernet)
        RX packets 1757  bytes 9262963 (8.8 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1232  bytes 72606 (70.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 0  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

#
```




```
[root@node1 ~]# iptables -vnL
Chain INPUT (policy ACCEPT 5149 packets, 528K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
10160   29M DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0
10160   29M DOCKER-ISOLATION-STAGE-1  all  --  *      *       0.0.0.0/0            0.0.0.0/0
 5953   29M ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0           ctstate RELATED,ESTABLISHED
    0     0 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
 4207  194K ACCEPT     all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT 4619 packets, 642K bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain DOCKER (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  !docker0 docker0  0.0.0.0/0            172.17.0.2           tcp dpt:80
    0     0 ACCEPT     tcp  --  !docker0 docker0  0.0.0.0/0            172.17.0.3           tcp dpt:80
    0     0 ACCEPT     tcp  --  !docker0 docker0  0.0.0.0/0            172.17.0.4           tcp dpt:80

Chain DOCKER-ISOLATION-STAGE-1 (1 references)
 pkts bytes target     prot opt in     out     source               destination
 4207  194K DOCKER-ISOLATION-STAGE-2  all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
10160   29M RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER-ISOLATION-STAGE-2 (1 references)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DROP       all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
 4207  194K RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination
10160   29M RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0
[root@node1 ~]#
```

范例:

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:9c:90:17:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:9cff:fe90:1799/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#docker run -itd -p 8888:80  nginx
5dee9be9afdbab8c2f6c4c5eb0f956c9579efe93110daf638f8fd15f43d961e2
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
```

```
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:9c:90:17:99 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:9cff:fe90:1799/64 scope link
       valid_lft forever preferred_lft forever
71: veth9e4fb80@if70: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether a2:7b:84:f7:8b:ff brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::a07b:84ff:fef7:8bff/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#docker exec -it 5dee9b bash
root@5dee9be9afdb:/# apt update
Get:1 http://security-cdn.debian.org/debian-security buster/updates InRelease
[65.4 kB]
Get:2 http://security-cdn.debian.org/debian-security buster/updates/main amd64
Packages [173 kB]
Get:3 http://deb.debian.org/debian buster InRelease [122 kB]
Get:4 http://deb.debian.org/debian buster-updates InRelease [49.3 kB]
Get:5 http://deb.debian.org/debian buster/main amd64 Packages [7908 kB]
Get:6 http://deb.debian.org/debian buster-updates/main amd64 Packages [5792 B]

Fetched 8323 kB in 13s (656 kB/s)

Reading package lists... Done
Building dependency tree
Reading state information... Done
All packages are up to date.
root@5dee9be9afdb:/# apt install net-tools
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  net-tools
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 248 kB of archives.
After this operation, 1002 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian buster/main amd64 net-tools amd64
1.60+git20180626.aebd88e-1 [248 kB]
Fetched 248 kB in 0s (610 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package net-tools.
(Reading database ... 7203 files and directories currently installed.)
Preparing to unpack .../net-tools_1.60+git20180626.aebd88e-1_amd64.deb ...
Unpacking net-tools (1.60+git20180626.aebd88e-1)
.....................................................................
.]
```

```
Setting up net-tools (1.60+git20180626.aebd88e-1)
...########.....................................................................
]
root@5dee9be9afdb:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 1926  bytes 8680620 (8.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1466  bytes 80919 (79.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@5dee9be9afdb:/# exit
exit
[root@ubuntu1804 ~]#iptables -vnL -t nat
Chain PREROUTING (policy ACCEPT 9 packets, 563 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0
          ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain OUTPUT (policy ACCEPT 1 packets, 76 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8
          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 1 packets, 76 bytes)
 pkts bytes target     prot opt in     out     source               destination

   71  4548 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0

    0     0 MASQUERADE  tcp  --  *      *       172.17.0.2           172.17.0.2
          tcp dpt:80

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination

    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0

    0     0 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0
           tcp dpt:8888 to:172.17.0.2:80

[root@ubuntu1804 ~]#ss -ntlp
State      Recv-Q     Send-Q          Local Address:Port          Peer
Address:Port
```

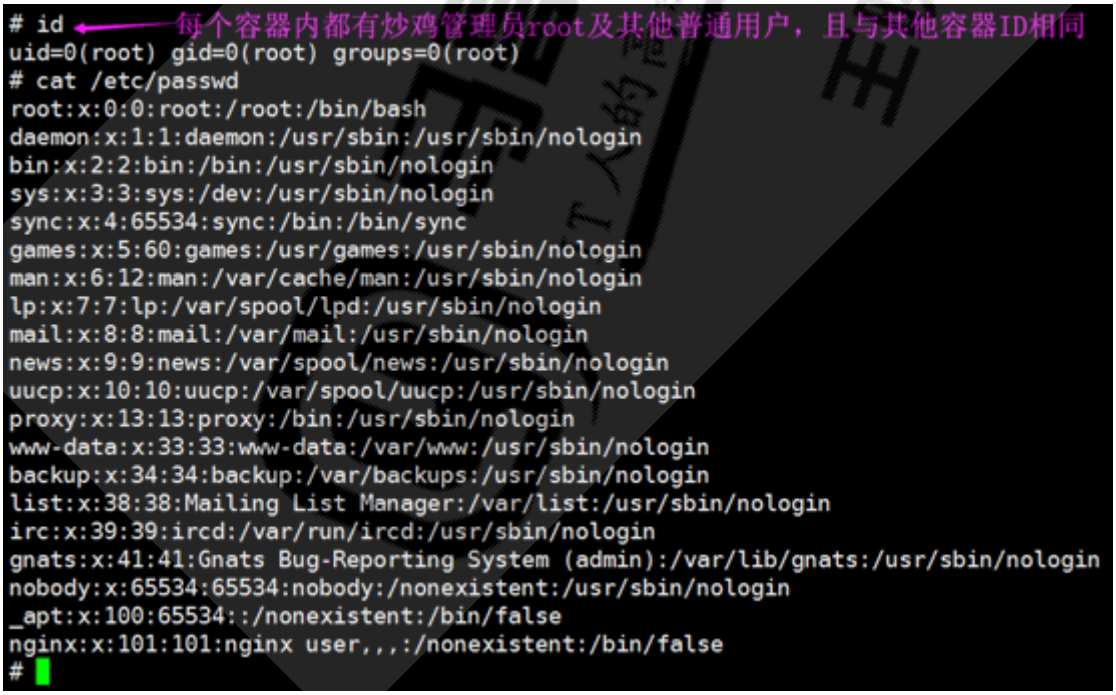
```
LISTEN    0          64                    0.0.0.0:2049               0.0.0.0:*

LISTEN    0          128                   0.0.0.0:43045              0.0.0.0:*
        users:(("rpc.mountd",pid=788,fd=17))
LISTEN    0          64                    0.0.0.0:38599              0.0.0.0:*

LISTEN    0          128                   0.0.0.0:111                0.0.0.0:*
        users:(("rpcbind",pid=725,fd=8))
LISTEN    0          128                   0.0.0.0:38805              0.0.0.0:*
        users:(("rpc.mountd",pid=788,fd=13))
LISTEN    0          128             127.0.0.53%lo:53                 0.0.0.0:*
        users:(("systemd-resolve",pid=785,fd=13))
LISTEN    0          128                   0.0.0.0:22                 0.0.0.0:*
        users:(("sshd",pid=863,fd=3))
LISTEN    0          128                 127.0.0.1:6010               0.0.0.0:*
        users:(("sshd",pid=913,fd=9))
LISTEN    0          128                 127.0.0.1:6011               0.0.0.0:*
        users:(("sshd",pid=913,fd=14))
LISTEN    0          128                   0.0.0.0:43775              0.0.0.0:*
        users:(("rpc.mountd",pid=788,fd=9))
LISTEN    0          64                       [::]:33633                 [::]:*

LISTEN    0          64                       [::]:2049                  [::]:*

LISTEN    0          128                      [::]:55659                 [::]:*
        users:(("rpc.mountd",pid=788,fd=15))
LISTEN    0          128                      [::]:111                   [::]:*
        users:(("rpcbind",pid=725,fd=11))
LISTEN    0          128                      [::]:44917                 [::]:*
        users:(("rpc.mountd",pid=788,fd=11))
LISTEN    0          128                      [::]:22                    [::]:*
        users:(("sshd",pid=863,fd=4))
LISTEN    0          128                       *:8888                     *:*
        users:(("docker-proxy",pid=15249,fd=4))
LISTEN    0          128                      [::]:41529                 [::]:*
        users:(("rpc.mountd",pid=788,fd=19))
LISTEN    0          128                     [::1]:6010                  [::]:*
        users:(("sshd",pid=913,fd=8))
LISTEN    0          128                     [::1]:6011                  [::]:*
        users:(("sshd",pid=913,fd=11))
[root@ubuntu1804 ~]#
[root@ubuntu1804 ~]#apt install bridge-utils
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  ifupdown
The following NEW packages will be installed:
  bridge-utils
0 upgraded, 1 newly installed, 0 to remove and 225 not upgraded.
Need to get 30.1 kB of archives.
After this operation, 102 kB of additional disk space will be used.
Get:1 http://mirrors.aliyun.com/ubuntu bionic/main amd64 bridge-utils amd64 1.5-
15ubuntu1 [30.1 kB]
Fetched 30.1 kB in 0s (259 kB/s)
ySelecting previously unselected package bridge-utils.
(Reading database ... 71346 files and directories currently installed.)
Preparing to unpack .../bridge-utils_1.5-15ubuntu1_amd64.deb ...
```

```
Unpacking bridge-utils (1.5-15ubuntu1)
.....................................................................................
...........]
Setting up bridge-utils (1.5-15ubuntu1)
...##############################.....................................................
.........]
Processing triggers for man-db (2.8.3-2)
...#########################################################################.........
.........]
[root@ubuntu1804 ~]#brctl  show
bridge name bridge id       STP enabled interfaces
docker0     8000.02429c901799   no      veth9e4fb80
```

1.1.5.6 User Namespace

各个容器内可能会出现重名的用户和用户组名称，或重复的用户UID或者GID，那么怎么隔离各个容器内的用户空间呢?

User Namespace允许在各个宿主机的各个容器空间内创建相同的用户名以及相同的用户UID和GID，只是会把用户的作用范围限制在每个容器内，即A容器和B容器可以有相同的用户名称和ID的账户，但是此用户的有效范围仅是当前容器内，不能访问另外一个容器内的文件系统，即相互隔离、互不影响、永不相见。



1.1.6 Control groups

Linux Cgroups的全称是Linux Control Groups,是Linux内核的一个功能.最早是由Google的工程师（主要是Paul Menage和Rohit Seth）在2006年发起，最早的名称为**进程容器**（process containers）。在2007年时，因为在Linux内核中，容器（container）这个名词有许多不同的意义，为避免混乱，被重命名为cgroup，并且被合并到2.6.24版的内核中去。自那以后，又添加了很多功能。

如果不对一个容器做任何资源限制，则宿主机会允许其占用无限大的内存空间，有时候会因为代码bug程序会一直申请内存，直到把宿主机内存占完，为了避免此类的问题出现，宿主机有必要对容器进行资源分配限制，比如CPU、内存等

Cgroups 最主要的作用，就是限制一个进程组能够使用的资源上限，包括CPU、内存、磁盘、网络带宽等等。此外，还能够对进程进行优先级设置，资源的计量以及资源的控制(比如:将进程挂起和恢复等操作)。

1.1.6.1 验证系统 cgroups

Cgroups在内核层默认已经开启，从CentOS 和 Ubuntu 不同版本对比，显然内核较新的支持的功能更多。

Centos 8.1 cgroups:

```
[root@centos8 ~]#cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)
[root@centos8 ~]#grep CGROUP /boot/config-4.18.0-147.el8.x86_64
CONFIG_CGROUPS=y
CONFIG_BLK_CGROUP=y
# CONFIG_DEBUG_BLK_CGROUP is not set
CONFIG_CGROUP_WRITEBACK=y
CONFIG_CGROUP_SCHED=y
CONFIG_CGROUP_PIDS=y
CONFIG_CGROUP_RDMA=y
CONFIG_CGROUP_FREEZER=y
CONFIG_CGROUP_HUGETLB=y
CONFIG_CGROUP_DEVICE=y
CONFIG_CGROUP_CPUACCT=y
CONFIG_CGROUP_PERF=y
CONFIG_CGROUP_BPF=y
# CONFIG_CGROUP_DEBUG is not set
CONFIG_SOCK_CGROUP_DATA=y
# CONFIG_BLK_CGROUP_IOLATENCY is not set
CONFIG_NETFILTER_XT_MATCH_CGROUP=m
CONFIG_NET_CLS_CGROUP=y
CONFIG_CGROUP_NET_PRIO=y
CONFIG_CGROUP_NET_CLASSID=y
[root@centos8 ~]#
```

Centos 7.6 cgroups:

```
[root@centos7 ~]#cat /etc/redhat-release
CentOS Linux release 7.6.1810 (Core)
[root@centos7 ~]#grep CGROUP /boot/config-3.10.0-957.el7.x86_64
CONFIG_CGROUPS=y
# CONFIG_CGROUP_DEBUG is not set
CONFIG_CGROUP_FREEZER=y
CONFIG_CGROUP_PIDS=y
CONFIG_CGROUP_DEVICE=y
CONFIG_CGROUP_CPUACCT=y
CONFIG_CGROUP_HUGETLB=y
CONFIG_CGROUP_PERF=y
CONFIG_CGROUP_SCHED=y
CONFIG_BLK_CGROUP=y
# CONFIG_DEBUG_BLK_CGROUP is not set
CONFIG_NETFILTER_XT_MATCH_CGROUP=m
CONFIG_NET_CLS_CGROUP=y
CONFIG_NETPRIO_CGROUP=y
```

ubuntu cgroups:

```
[root@ubuntu1804 ~]#grep CGROUP  /boot/config-4.15.0-29-generic
CONFIG_CGROUPS=y
CONFIG_BLK_CGROUP=y
# CONFIG_DEBUG_BLK_CGROUP is not set
CONFIG_CGROUP_WRITEBACK=y
CONFIG_CGROUP_SCHED=y
CONFIG_CGROUP_PIDS=y
CONFIG_CGROUP_RDMA=y
CONFIG_CGROUP_FREEZER=y
CONFIG_CGROUP_HUGETLB=y
CONFIG_CGROUP_DEVICE=y
CONFIG_CGROUP_CPUACCT=y
CONFIG_CGROUP_PERF=y
CONFIG_CGROUP_BPF=y
# CONFIG_CGROUP_DEBUG is not set
CONFIG_SOCK_CGROUP_DATA=y
CONFIG_NETFILTER_XT_MATCH_CGROUP=m
CONFIG_NET_CLS_CGROUP=m
CONFIG_CGROUP_NET_PRIO=y
CONFIG_CGROUP_NET_CLASSID=y
```

cgroups 中内存模块:

```
[root@ubuntu1804 ~]#grep MEMCG  /boot/config-4.15.0-29-generic
CONFIG_MEMCG=y
CONFIG_MEMCG_SWAP=y
# CONFIG_MEMCG_SWAP_ENABLED is not set
CONFIG_SLUB_MEMCG_SYSFS_ON=y
```

1.1.6.2 cgroups 具体实现

- blkio: 块设备IO限制
- cpu: 使用调度程序为 cgroup 任务提供 cpu 的访问
- cpuacct: 产生 cgroup 任务的 cpu 资源报告
- cpuset: 如果是多核心的 cpu，这个子系统会为 cgroup 任务分配单独的 cpu 和内存
- devices: 允许或拒绝 cgroup 任务对设备的访问
- freezer: 暂停和恢复 cgroup 任务
- memory: 设置每个 cgroup 的内存限制以及产生内存资源报告
- net_cls: 标记每个网络包以供 cgroup 方便使用
- ns: 命名空间子系统
- perf_event: 增加了对每 group 的监测跟踪的能力，可以监测属于某个特定的 group 的所有线程以及运行在特定CPU上的线程

1.1.6.3 查看系统 cgroups

```
[root@ubuntu1804 ~]#ll  /sys/fs/cgroup/
total 0
drwxr-xr-x 15 root root 380 Jan 22 16:20 ./
drwxr-xr-x 10 root root   0 Jan 22 16:20 ../
dr-xr-xr-x  5 root root   0 Jan 22 16:20 blkio/
lrwxrwxrwx  1 root root  11 Jan 22 16:20 cpu -> cpu,cpuacct/
lrwxrwxrwx  1 root root  11 Jan 22 16:20 cpuacct -> cpu,cpuacct/
```

```
dr-xr-xr-x  5 root root   0 Jan 22 16:20 cpu,cpuacct/
dr-xr-xr-x  3 root root   0 Jan 22 16:20 cpuset/
dr-xr-xr-x  5 root root   0 Jan 22 16:20 devices/
dr-xr-xr-x  3 root root   0 Jan 22 16:20 freezer/
dr-xr-xr-x  3 root root   0 Jan 22 16:20 hugetlb/
dr-xr-xr-x  5 root root   0 Jan 22 16:20 memory/
lrwxrwxrwx  1 root root  16 Jan 22 16:20 net_cls -> net_cls,net_prio/
dr-xr-xr-x  3 root root   0 Jan 22 16:20 net_cls,net_prio/
lrwxrwxrwx  1 root root  16 Jan 22 16:20 net_prio -> net_cls,net_prio/
dr-xr-xr-x  3 root root   0 Jan 22 16:20 perf_event/
dr-xr-xr-x  5 root root   0 Jan 22 16:20 pids/
dr-xr-xr-x  2 root root   0 Jan 22 16:20 rdma/
dr-xr-xr-x  6 root root   0 Jan 22 16:20 systemd/
dr-xr-xr-x  5 root root   0 Jan 22 16:20 unified/

[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/cpu/docker/5dee9be9afdbab8c2f6c4c5eb0f956c9579efe93110daf638f8fd1
5f43d961e2/cpuacct.usage
4751336886
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/5dee9be9afdbab8c2f6c4c5eb0f956c9579efe93110daf638f8
fd15f43d961e2/cpuacct.usage
cat:
/sys/fs/cgroup/memory/docker/5dee9be9afdbab8c2f6c4c5eb0f956c9579efe93110daf638f8
fd15f43d961e2/cpuacct.usage: No such file or directory

[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/5dee9be9afdbab8c2f6c4c5eb0f956c9579efe93110daf638f8
fd15f43d961e2/memory.limit_in_bytes
9223372036854771712
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/5dee9be9afdbab8c2f6c4c5eb0f956c9579efe93110daf638f8
fd15f43d961e2/memory.max_usage_in_bytes
79278080
```

1.1.7 容器管理工具

有了以上的chroot、namespace、cgroups就具备了基础的容器运行环境，但是还需要有相应的容器创建与删除的管理工具、以及怎么样把容器运行起来、容器数据怎么处理、怎么进行启动与关闭等问题需要解决，于是容器管理技术出现了。目前主要是使用docker，早期使用 LXC

1.1.7.1 LXC

LXC：Linux Container。可以提供轻量级的虚拟化功能,以便隔离进程和资源,包括一系列容器的管理工具软件,如，lxc-create,lxc-start,lxc-attach等,但这技术功能不完善,目前较少使用

官方网站：https://linuxcontainers.org/

案例：Ubuntu安装 和使用 LXC

```
[root@ubuntu1804 ~]#apt install lxc lxd
Reading package lists... Done
Building dependency tree
Reading state information... Done
lxd is already the newest version (3.0.3-0ubuntu1~18.04.1).
```

```
lxc is already the newest version (3.0.3-0ubuntu1~18.04.1).
......

[root@ubuntu1804 ~]#lxc-checkconfig   #检查内核对lcx的支持状况，必须全部为lcx
Kernel configuration not found at /proc/config.gz; searching...
Kernel configuration found at /boot/config-4.15.0-29-generic
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
......


[root@ubuntu1804 ~]#lxc-create -t download --name alpine1 --  --dist alpine  --
release 3.9 --arch amd64
Setting up the GPG keyring
Downloading the image index
Downloading the rootfs
Downloading the metadata
The image cache is now ready
Unpacking the rootfs

---
You just created an Alpinelinux 3.9 x86_64 (20200121_13:00) container.
[root@ubuntu1804 ~]#lxc-start alpine1    #启动lxc容器
[root@ubuntu1804 ~]#lxc-attach alpine1   #进入lxc容器
~ # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:16:3E:DF:9E:45
          inet addr:10.0.1.51  Bcast:10.0.1.255  Mask:255.255.255.0
          inet6 addr: fe80::216:3eff:fedf:9e45/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2484 (2.4 KiB)  TX bytes:1726 (1.6 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

~ # uname -r
4.15.0-29-generic
~ # uname -a
Linux alpine12 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018
x86_64 Linux
~ # cat /etc/issue
Welcome to Alpine Linux 9
Kernel \r on an \m (\l)

~ # exit
```

```
[root@ubuntu1804 ~]#
```

命令选项说明:

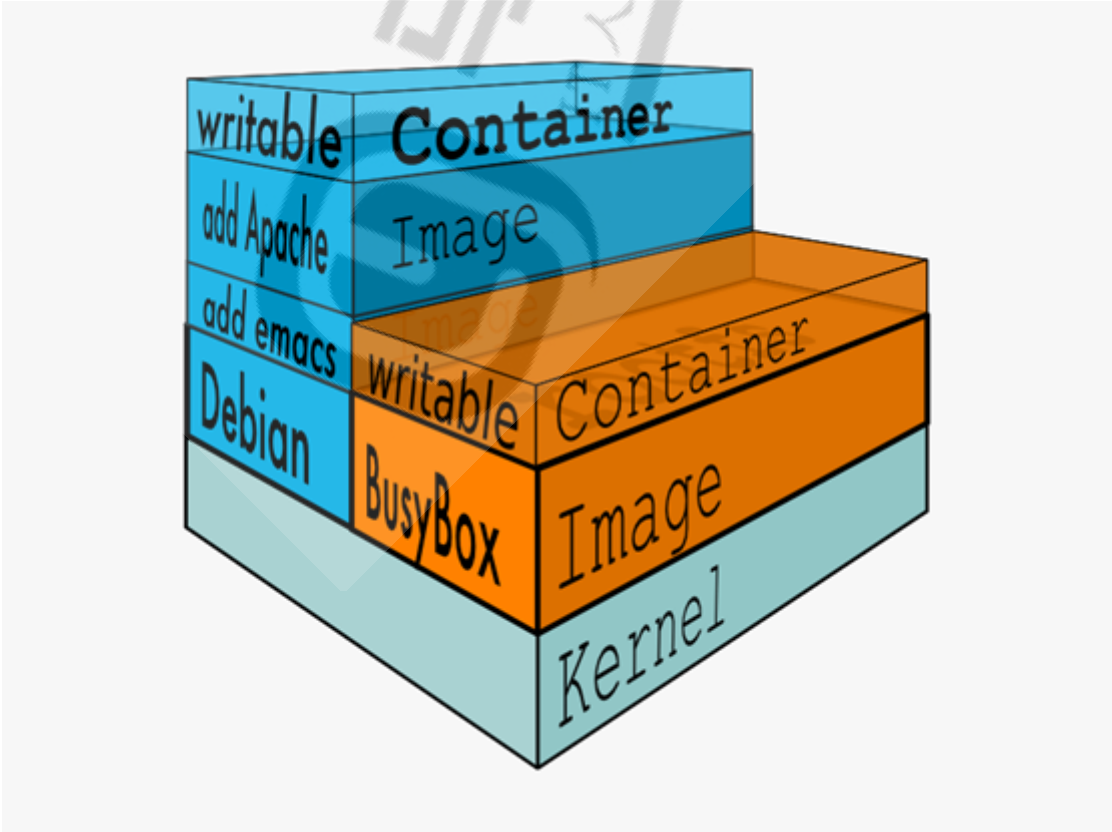
```
-t 模板: -t 选项后面跟的是模板，模式可以认为是一个原型，用来说明需要一个什么样的容器(比如容器里面需不需要有vim, apache等软件). 模板实际上就是一个脚本文件(位于/usr/share/lxc/templates目录)，我们这里指定download模板(lxc-create会调用lxc-download脚本，该脚本位于刚说的模板目录中)是说明我们目前没有自己模板，需要下载官方的模板

--name 容器名称:    为创建的容器命名
-- : --用来说明后面的参数是传递给download脚本的，告诉脚本需要下载什么样的模板
--dist 操作系统名称:  指定操作系统
--release 操作系统: 指定操作系统，可以是各种Linux的变种
--arch 架构:  指定架构，是x86还是arm，是32位还是64位
```

lxc启动容器依赖于模板，清华模板源: https://mirrors.tuna.tsinghua.edu.cn/help/lxc-images/，但是做模板相对较难，需要手动一步步创构建文件系统、准备基础目录及可执行程序等，而且在大规模使用容器的场景很难横向扩展，另外后期代码升级也需要重新从头构建模板，基于以上种种原因便有了docker

1.1.7.2 docker

Docker 相当于增强版的LXC,功能更为强大和易用,也是当前最主流的容器前端管理工具

Docker 先启动一个容器也需要一个外部模板，也称为镜像，docke的镜像可以保存在一个公共的地方共享使用，只要把镜像下载下来就可以使用，最主要的是可以在镜像基础之上做自定义配置并且可以再把其提交为一个镜像，一个镜像可以被启动为多个容器。



Docker的镜像是分层的，镜像底层为库文件且只读层即不能写入也不能删除数据，从镜像加载启动为一个容器后会生成一个可写层，其写入的数据会复制到宿主机上对应容器的目录，但是容器内的数据在删除容器后也会被随之删除。

1.1.7.3 pouch

项目网点: https://github.com/alibaba/pouch

Pouch （小袋子）起源于 2011 年，并于2017年11月19日上午，在中国开源年会现场，阿里巴巴正式开源了基于 Apache 2.0 协议的容器技术 Pouch。Pouch 是一款轻量级的容器技术，拥有快速高效、可移植性高、资源占用少等特性，主要帮助阿里更快的做到内部业务的交付，同时提高超大规模下数据中心的物理资源利用率

目前的容器方案大多基于 Linux 内核提供的 cgroup 和 namespace 来实现隔离，然后这样轻量级方案存在弊端:

- 容器间，容器与宿主间，共享同一个内核
- 内核实现的隔离资源，维度不足

面对如此的内核现状，阿里巴巴采取了三个方面的工作，来解决容器的安全问题:

- 用户态增强容器的隔离维度，比如网络带宽、磁盘使用量等
- 给内核提交 patch，修复容器的资源可见性问题，cgroup 方面的 bug
- 实现基于 Hypervisor 的容器，通过创建新内核来实现容器隔离

1.1.7.4 Podman

虽然目前 Docker 是管理 Linux 容器最好的工具，注意没有之一，但是podman的横空出现即将改变这一点

什么是Podman?

Podman即Pod Manager tool，从名称上可以看出和kubernets的pod的密切联系，不过就其功能来说，简而言之: `alias docker = podman`,是CentOS 8 新集成的功能，或许不久的未来会代替docker

Podman是一个 为 Kubernetes 而生的开源的容器管理工具，原来是 CRI-O（即容器运行时接口CRI 和开放容器计划OCI） 项目的一部分，后来被分离成一个单独的项目叫 libpod。其可在大多数Linux平台上使用，它是一种无守护程序的容器引擎，用于在Linux系统上开发，管理和运行任何符合Open Container Initiative（OCI）标准的容器和容器镜像。

Podman 提供了一个与Docker兼容的命令行前端，Podman 里面87%的指令都和Docker CLI 相同，因此可以简单地为Docker CLI别名，即“ alias docker = podman”，事实上，podman使用的一些库也是docker的一部分。

```
CRI-O is an implementation of the Kubernetes CRI (Container Runtime Interface)
to enable using OCI (Open Container Initiative) compatible runtimes
```

官网地址: https://podman.io/

项目地址: https://github.com/containers/libpod

Podman 和docker不同之处

- docker 需要在系统上运行一个守护进程(docker daemon)，这会产生一定的开销，而podman 不需要
- 启动容器的方式不同:
  `docker cli` 命令通过API跟 `Docker Engine(引擎)` 交互告诉它我想创建一个container，然后 `docker Engine` 才会调用 `OCI container runtime(runc)` 来启动一个container。这代表container的process(进程)不会是 `Docker CLI` 的 `child process(子进程)`，而是 `Docker Engine` 的 `child process`。
  `Podman` 是直接给 `OCI containner runtime(runc)` 进行交互来创建container的，所以 `container process` 直接是 `podman` 的 `child process`。
- 因为docke有docker daemon，所以docker启动的容器支持 `--restart` 策略，但是podman不支持
- docker需要使用root用户来创建容器。 这可能会产生安全风险，尤其是当用户知道docker run命令的--privileged选项时。podman既可以由root用户运行，也可以由非特权用户运行
- docker在Linux上作为守护进程运行扼杀了容器社区的创新。 如果要更改容器的工作方式，则需要更改docker守护程序并将这些更改推送到上游。 没有守护进程，容器基础结构更加模块化，更容易进行更改。 podman的无守护进程架构更加灵活和安全。

1.1.8 Docker 的优势

- 快速部署: 短时间内可以部署成百上千个应用，更快速交付到线上
- 高效虚拟化: 不需要额外hypervisor支持，基于linux内核实现应用虚拟化，相比虚拟机大幅提高性能和效率
- 节省开支: 提高服务器利用率，降低IT支出
- 简化配置: 将运行环境打包保存至容器，使用时直接启动即可
- 环境统一: 将开发，测试，生产的应用运行环境进行标准化和统一，减少环境不一样带来的各种问题
- 快速迁移和扩展: 可实现跨平台运行在物理机、虚拟机、公有云等环境，良好的兼容性可以方便将应用从A宿主机迁移到B宿主机，甚至是A平台迁移到B平台
- 更好的实现面向服务的架构,推荐一个容器只运行一个应用,实现分布的应用模型,可以方便的进行横向扩展,符合开发中高内聚,低耦合的要求,减少不同服务之间的相互影响

1.1.9 Docker 的缺点

- 多个容器共用宿主机的内核，各应用之间的隔离不如虚拟机彻底
- 由于和宿主机之间的进程也是隔离的,需要进入容器查看和调试容器内进程等资源,变得比较困难和繁琐
- 如果容器内进程需要查看和调试,需要在每个容器内都需要安装相应的工具,这也造成存储空间的重复浪费

1.1.10 容器的核心技术

1.1.10.1 容器规范

OCI 官网:https://opencontainers.org/

容器技术除了的docker之外，还有coreOS的rkt，还有阿里的Pouch，为了保证容器生态的标准性和健康可持续发展，包括Linux 基金会、Docker、微软、红帽谷歌和、IBM、等公司在2015年6月共同成立了一个叫Open Container Initiative（OCI）的组织，其目的就是制定开放的标准的容器规范，目前OCI一共发布了两个规范，**分别是runtime spec和 image format spec**，有了这两个规范，不同的容器公司开发的容器只要兼容这两个规范，就可以保证容器的可移植性和相互可操作性。

1.1.10.2 容器 runtime

runtime是真正运行容器的地方，因此为了运行不同的容器runtime需要和操作系统内核紧密合作相互在支持，以便为容器提供相应的运行环境

runtime 类型:

- Lxc: linux上早期的runtime，在 2013 年 Docker 刚发布的时候,就是采用lxc作为runtime, Docker 把 LXC 复杂的容器创建与使用方式简化为 Docker 自己的一套命令体系。随着Docker的发展，原有的LXC不能满足Docker的需求,比如跨平台功能
- Libcontainer: 随着 Docker 的不断发展，重新定义容器的实现标准，将底层实现都抽象化到 Libcontainer 的接口。这就意味着，底层容器的实现方式变成了一种可变的方案，无论是使用 namespace、cgroups 技术抑或是使用 systemd 等其他方案，只要实现了 Libcontainer 定义的一组接口，Docker 都可以运行。这也为 Docker 实现全面的跨平台带来了可能。
- **runc**: 早期libcontainer是Docker公司控制的一个开源项目，OCI的成立后,Docker把libcontainer项目移交给了OCI组织,runC就是在libcontainer的基础上进化而来,目前Docker默认的runtime，runc遵守OCI规范
- rkt: 是CoreOS开发的容器runtime，也符合OCI规范，所以使用rktruntime也可以运行Docker容器

范例: 查看docker的 runtime

```
[root@ubuntu1804 ~]#docker info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 1
 Server Version: 19.03.5
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc          #Runtimes
 Default Runtime: runc  #runtime
 Init Binary: docker-init
 containerd version: b34a5c8af56e510852c35414db4c1f4fa6172339
 runc version: 3e425f80a8c931f88e6d94a8c831b9d5aa481657
 init version: fec3683
```

```
 Security Options:
  apparmor
  seccomp
   Profile: default
 Kernel Version: 4.15.0-29-generic
 Operating System: Ubuntu 18.04.1 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 962MiB
 Name: ubuntu1804.magedu.org
 ID: G2JQ:M4DG:CW74:EETR:GU5U:OROC:ZN2F:RKSA:YQY2:XJYX:OHG7:SSVE
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false

WARNING: No swap limit support
```

1.1.10.3 容器管理工具

管理工具连接runtime与用户，对用户提供图形或命令方式操作，然后管理工具将用户操作传递给runtime执行。

- lxc 是lxd 的管理工具
- Runc的管理工具是**docker engine**，docker engine包含后台deamon和cli两部分，大家经常提到的Docker就是指的docker engine
- Rkt的管理工具是rkt cli

范例: 查看docker engine

```
[root@ubuntu1804 ~]#docker version
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:29:52 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
  Git commit:       633a0ea838
  Built:            Wed Nov 13 07:28:22 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.10
```

```
  GitCommit:        b34a5c8af56e510852c35414db4c1f4fa6172339
 runc:
  Version:          1.0.0-rc8+dev
  GitCommit:        3e425f80a8c931f88e6d94a8c831b9d5aa481657
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
[root@ubuntu1804 ~]#
```

1.1.10.4 容器定义工具

容器定义工具允许用户定义容器的属性和内容，以方便容器能够被保存、共享和重建。

Docker image：是docker 容器的模板，runtime依据docker image创建容器

Dockerfile：包含N个命令的文本文件，通过dockerfile创建出docker image

ACI(App container image)：与docker image类似，是CoreOS开发的rkt容器的镜像格式

1.1.10.5 镜像仓库 Registry

统一保存镜像而且是多个不同镜像版本的地方，叫做镜像仓库

- Docker hub：docker官方的公共仓库，已经保存了大量的常用镜像，可以方便大家直接使用
- 阿里云，网易等第三方镜像的公共仓库
- Image registry：docker 官方提供的私有仓库部署工具，无web管理界面，目前使用较少
- Harbor：vmware 提供的自带web界面自带认证功能的镜像私有仓库，目前有很多公司使用

范例：镜像地址格式

```
docker.io/library/alpine
harbor.magedu.org/project/centos:7.2.1511
registry.cn-hangzhou.aliyuncs.com/wangxiaochun/magedu:v1
172.18.200.101/project/centos: latest
172.18.200.101/project/java-7.0.59:v1
```

1.1.10.6 容器编排工具

当多个容器在多个主机运行的时候，单独管理容器是相当复杂而且很容易出错，而且也无法实现某一台主机宕机后容器自动迁移到其他主机从而实现高可用的目的，也无法实现动态伸缩的功能，因此需要有一种工具可以实现统一管理、动态伸缩、故障自愈、批量执行等功能，这就是容器编排引擎

容器编排通常包括容器管理、调度、集群定义和服务发现等功能

- Docker compose : docker 官方实现单机的容器的编排工具
- Docker swarm：docker 官方开发的容器编排引擎,支持overlay network
- Mesos+Marathon：Mesos是Apache下的开源分布式资源管理框架，它被称为是分布式系统的内核。Mesos最初是由加州大学伯克利分校的AMPLab开发的，后在Twitter得到广泛使用。通用的集群组员调度平台，mesos(资源分配)与marathon(容器编排平台)一起提供容器编排引擎功能
- Kubernetes：google领导开发的容器编排引擎，内部项目为Borg，且其同时支持 docker 和 CoreOS,当前已成为容器编排工具事实上的标准

1.1.11 docker(容器)的依赖技术

容器网络:

docker自带的网络docker network仅支持管理单机的容器网络，当多主机运行的时候需要使用第三方开源网络，例如:calico、flannel等

服务发现:

容器的动态扩容特性决定了容器IP也会随之变化，因此需要有一种机制开源自动识别并将用户请求动态转发到新创建的容器上，kubernetes自带服务发现功能，需要结合kube-dns服务解析内部域名

容器监控:

可以通过原生命令docker ps/top/stats 查看容器运行状态，另外也可以使用Prometheus 、heapster等第三方监控工具监控容器的运行状态

数据管理:

容器的动态迁移会导致其在不同的Host之间迁移，因此如何保证与容器相关的数据也能随之迁移或随时访问，可以使用逻辑卷/存储挂载等方式解决

日志收集:

docker 原生的日志查看工具docker logs，但是容器内部的日志需要通过ELK等专门的日志收集分析和展示工具进行处理

1.2 Docker安装及基础命令介绍

1.2.1 Docker 安装准备

官方网址: https://www.docker.com/

OS系统版本选择:

Docker 目前已经支持多种操作系统的安装运行，比如Ubuntu、CentOS、Redhat、Debian、Fedora，甚至是还支持了Mac和Windows，在linux系统上需要内核版本在3.10或以上

Docker版本选择:

docker版本号之前一直是0.X版本或1.X版本，但是从2017年3月1号开始改为每个季度发布一次稳定版，其版本号规则也统一变更为YY.MM，例如17.09表示是2017年9月份发布的

Docker之前没有区分版本，但是2017年推出(将docker更名为)新的项目Moby，github地址: https://github.com/moby/moby，Moby项目属于Docker项目的全新上游，Docker将是一个隶属于的Moby的子产品，而且之后的版本之后开始区分为 CE（Docker Community Edition，社区版本）和 EE（Docker Enterprise Edition，企业收费版），CE社区版本和EE企业版本都是每个季度发布一个新版本，但是EE版本提供后期安全维护1年，而CE版本是4个月，以下为官方原文:

https://blog.docker.com/2017/03/docker-enterprise-edition/

```
Docker CE and EE are released quarterly, and CE also has a monthly “Edge”
option. Each Docker EE release is supported and maintained for one year and
receives security and critical bugfixes during that period. We are also
improving Docker CE maintainability by maintaining each quarterly CE release for
4 months. That gets Docker CE users a new 1-month window to update from one
version to the next.
```

如果要布署到kubernets上，需要查看相关kubernets对docker版本要求的说明，比如:

https://github.com/kubernetes/kubernetes/blob/v1.17.2/CHANGELOG-1.17.md

1.2.2 安装和删除方法

官方文档 : https://docs.docker.com/engine/install/

阿里云文档: https://developer.aliyun.com/mirror/docker-ce?spm=a2c6h.13651102.0.0.3e221b11guHCWE

1.2.2.1 Ubuntu 安装和删除Docker

官方文档: https://docs.docker.com/install/linux/docker-ce/ubuntu/

Ubuntu 14.04/16.04/18.04 安装docker

```
# step 1: 安装必要的一些系统工具
sudo apt-get update
sudo apt-get -y install apt-transport-https ca-certificates curl software-properties-common
# step 2: 安装GPG证书
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key add -
# Step 3: 写入软件源信息
sudo add-apt-repository "deb [arch=amd64] https://mirrors.aliyun.com/docker-ce/linux/ubuntu $(lsb_release -cs) stable"
# Step 4: 更新并安装Docker-CE
sudo apt-get -y update
sudo apt-get -y install docker-ce

# 安装指定版本的Docker-CE:
# Step 1: 查找Docker-CE的版本:
apt-cache madison docker-ce
 docker-ce | 5:19.03.5~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:19.03.4~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:19.03.3~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:19.03.2~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:19.03.1~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:19.03.0~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.9~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.8~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.7~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.6~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.5~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.4~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.3~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.2~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.1~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 5:18.09.0~3-0~ubuntu-bionic | https://mirrors.aliyun.com/docker-ce/linux/ubuntu bionic/stable amd64 Packages
```

```
 docker-ce | 18.06.3~ce~3-0~ubuntu | https://mirrors.aliyun.com/docker-
ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 18.06.2~ce~3-0~ubuntu | https://mirrors.aliyun.com/docker-
ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 18.06.1~ce~3-0~ubuntu | https://mirrors.aliyun.com/docker-
ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 18.06.0~ce~3-0~ubuntu | https://mirrors.aliyun.com/docker-
ce/linux/ubuntu bionic/stable amd64 Packages
 docker-ce | 18.03.1~ce~3-0~ubuntu | https://mirrors.aliyun.com/docker-
ce/linux/ubuntu bionic/stable amd64 Packages

# Step 2: 安装指定版本的Docker-CE: (VERSION例如上面的5:17.03.1~ce-0~ubuntu-xenial)
sudo apt-get -y install docker-ce=[VERSION] docker-ce-cli=[VERSION]
#示例:指定版本安装
apt-get -y install docker-ce=5:18.09.9~3-0~ubuntu-bionic  docker-ce-
cli=5:18.09.9~3-0~ubuntu-bionic
```

删除docker

```
[root@ubuntu ~]#apt purge docker-ce
[root@ubuntu ~]#rm -rf /var/lib/docker
```

1.2.2.2 CentOS 安装和删除Docker

官方文档: https://docs.docker.com/install/linux/docker-ce/centos/

CentOS 6 因内核太旧，即使支持安装docker，但会有各种问题，不建议安装

CentOS 7 的 extras 源虽然可以安装docker，但包比较旧，建议从官方源或镜像源站点下载安装docker

CentOS 8 有新技术 podman 代替 docker

因此建议在CentOS 7 上安装 docker

```
#extras 源中包名为docker
[root@centos7 ~]#yum list docker
Loaded plugins: fastestmirror
Repository base is listed more than once in the configuration
Repository extras is listed more than once in the configuration
Loading mirror speeds from cached hostfile
 * base: mirrors.tuna.tsinghua.edu.cn
 * extras: mirrors.tuna.tsinghua.edu.cn
 * updates: mirrors.tuna.tsinghua.edu.cn
Available Packages
docker.x86_64       2:1.13.1-103.git7f2769b.el7.centos
extras
```

下载rpm包安装:

官方rpm包下载地址:

https://download.docker.com/linux/centos/7/x86_64/stable/Packages/

阿里镜像下载地址:

https://mirrors.aliyun.com/docker-ce/linux/centos/7/x86_64/stable/Packages/

通过yum源安装:

由于官网的yum源太慢，下面使用阿里云的Yum源进行安装

```
rm -rf /etc/yum.repos.d/*

#CentOS 7 安装docker依赖三个yum源:Base,Extras,docker-ce
wget -O /etc/yum.repos.d/CentOS-Base.repo http://mirrors.aliyun.com/repo/Centos-7.repo
wget -O /etc/yum.repos.d/epel.repo http://mirrors.aliyun.com/repo/epel-7.repo
wget -O /etc/yum.repos.d/docker-ce.repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo


yum clean all
yum -y install docker-ce
systemctl enable --now docker
```

删除 docker

```
[root@centos7 ~]#yum remove docker-ce

#删除docker资源存放的相关文件
[root@centos7 ~]#rm -rf /var/lib/docker
```

范例: CentOS 7 基于阿里云的安装docker方法

阿里云说明: https://developer.aliyun.com/mirror/docker-ce?spm=a2c6h.13651102.0.0.3e221b11sUMKNV

```
# step 1: 安装必要的一些系统工具
yum install -y yum-utils device-mapper-persistent-data lvm2
# Step 2: 添加软件源信息
yum-config-manager --add-repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
# Step 3: 更新并安装Docker-CE
yum makecache fast
yum -y install docker-ce
# Step 4: 开启Docker服务
service docker start

# 注意:
# 官方软件源默认启用了最新的软件，您可以通过编辑软件源的方式获取各个版本的软件包。例如官方并没有将测试版本的软件源置为可用，您可以通过以下方式开启。同理可以开启各种测试版本等。
# vim /etc/yum.repos.d/docker-ee.repo
#   将[docker-ce-test]下方的enabled=0修改为enabled=1
#
# 安装指定版本的Docker-CE:
# Step 1: 查找Docker-CE的版本:
# yum list docker-ce.x86_64 --showduplicates | sort -r
#   Loading mirror speeds from cached hostfile
#   Loaded plugins: branch, fastestmirror, langpacks
#   docker-ce.x86_64            17.03.1.ce-1.el7.centos            docker-ce-stable
```

```
#   docker-ce.x86_64            17.03.1.ce-1.el7.centos            @docker-ce-
stable
#   docker-ce.x86_64            17.03.0.ce-1.el7.centos            docker-ce-
stable
#   Available Packages
# Step2: 安装指定版本的Docker-CE: (VERSION例如上面的17.03.0.ce.1-1.el7.centos)
yum -y install docker-ce-[VERSION]

#示例
[root@centos7 ~]#yum -y install docker-ce-19.03.12-3.el7
```

范例：在CentOS 7上安装指定版本的docker

```
[root@centos7 ~]#cat /etc/redhat-release
CentOS Linux release 7.6.1810 (Core)
[root@centos7 ~]#ls /etc/yum.repos.d/
backup  base.repo
[root@centos7 ~]#wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo
Saving to: '/etc/yum.repos.d/docker-ce.repo'
100%[======================================================================>]
2,640       --.-K/s   in 0s

2020-01-23 21:56:21 (505 MB/s) - '/etc/yum.repos.d/docker-ce.repo' saved
[2640/2640]

[root@centos7 ~]#ls /etc/yum.repos.d/
backup  base.repo  docker-ce.repo
[root@centos7 ~]#yum clean all
Loaded plugins: fastestmirror
Cleaning repos: base docker-ce-stable epel extras
Cleaning up list of fastest mirrors
[root@centos7 ~]#yum repolist
repo id                    repo name
status
base                       CentOS
10,019
docker-ce-stable/x86_64    Docker CE Stable - x86_64
   63
epel/7/x86_64              EPEL
13,513
extras/7/x86_64            extras
   307
repolist: 23,902
[root@centos7 ~]#yum list docker-ce* --showduplicates | sort -r
Loading mirror speeds from cached hostfile
Loaded plugins: fastestmirror
docker-ce.x86_64                3:19.03.5-3.el7                docker-ce-stable
docker-ce.x86_64                3:19.03.4-3.el7                docker-ce-stable
docker-ce.x86_64                3:19.03.3-3.el7                docker-ce-stable
docker-ce.x86_64                3:19.03.2-3.el7                docker-ce-stable
docker-ce.x86_64                3:19.03.1-3.el7                docker-ce-stable
docker-ce.x86_64                3:19.03.0-3.el7                docker-ce-stable
docker-ce.x86_64                3:18.09.9-3.el7                docker-ce-stable
docker-ce.x86_64                3:18.09.8-3.el7                docker-ce-stable
docker-ce.x86_64                3:18.09.7-3.el7                docker-ce-stable
docker-ce.x86_64                3:18.09.6-3.el7                docker-ce-stable
```

```
docker-ce.x86_64                3:18.09.5-3.el7                 docker-ce-stable
docker-ce.x86_64                3:18.09.4-3.el7                 docker-ce-stable
docker-ce.x86_64                3:18.09.3-3.el7                 docker-ce-stable
docker-ce.x86_64                3:18.09.2-3.el7                 docker-ce-stable
docker-ce.x86_64                3:18.09.1-3.el7                 docker-ce-stable
docker-ce.x86_64                3:18.09.0-3.el7                 docker-ce-stable
docker-ce.x86_64                18.06.3.ce-3.el7                docker-ce-stable
docker-ce.x86_64                18.06.2.ce-3.el7                docker-ce-stable
docker-ce.x86_64                18.06.1.ce-3.el7                docker-ce-stable
docker-ce.x86_64                18.06.0.ce-3.el7                docker-ce-stable
docker-ce.x86_64                18.03.1.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                18.03.0.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.12.1.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.12.0.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.09.1.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.09.0.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.06.2.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.06.1.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.06.0.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.03.3.ce-1.el7                docker-ce-stable
docker-ce.x86_64                17.03.2.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.03.1.ce-1.el7.centos         docker-ce-stable
docker-ce.x86_64                17.03.0.ce-1.el7.centos         docker-ce-stable
docker-ce-selinux.noarch        17.03.3.ce-1.el7                docker-ce-stable
docker-ce-selinux.noarch        17.03.2.ce-1.el7.centos         docker-ce-stable
docker-ce-selinux.noarch        17.03.1.ce-1.el7.centos         docker-ce-stable
docker-ce-selinux.noarch        17.03.0.ce-1.el7.centos         docker-ce-stable
docker-ce-cli.x86_64            1:19.03.5-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:19.03.4-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:19.03.3-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:19.03.2-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:19.03.1-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:19.03.0-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.9-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.8-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.7-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.6-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.5-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.4-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.3-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.2-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.1-3.el7                 docker-ce-stable
docker-ce-cli.x86_64            1:18.09.0-3.el7                 docker-ce-stable
Available Packages
[root@centos7 ~]#yum -y install docker-ce-18.09.9-3.el7 docker-ce-cli-18.09.9-
3.el7
Dependencies Resolved

================================================================================
=========
 Package                        Arch            Version
Repository                  Size
================================================================================
=========
Installing:
 docker-ce                      x86_64          3:18.09.9-3.el7
docker-ce-stable            21 M
```

```
 docker-ce-cli                     x86_64          1:18.09.9-3.el7
docker-ce-stable           16 M
Installing for dependencies:
 audit-libs-python                 x86_64          2.8.4-4.el7
base                       76 k
 checkpolicy                       x86_64          2.5-8.el7
base                      295 k
 container-selinux                 noarch          2:2.107-3.el7
extras                     39 k
 containerd.io                     x86_64          1.2.10-3.2.el7
docker-ce-stable           23 M
 libcgroup                         x86_64          0.41-20.el7
base                       66 k
 libsemanage-python                x86_64          2.5-14.el7
base                      113 k
 policycoreutils-python            x86_64          2.5-29.el7
base                      456 k
 python-IPy                        noarch          0.75-6.el7
base                       32 k
 setools-libs                      x86_64          3.3.8-4.el7
base                      620 k

Transaction Summary
================================================================================
================================
Install  2 Packages (+9 Dependent packages)

Total download size: 62 M
Installed size: 258 M
Downloading packages:
(1/4): container-selinux-2.107-3.el7.noarch.rpm
       |  39 kB  00:00:00
warning: /var/cache/yum/x86_64/7/docker-ce-stable/packages/containerd.io-1.2.10-
3.2.el7.x86_64.rpm: Header V4 RSA/SHA512 Signature, key ID 621e9f35: NOKEY
Public key for containerd.io-1.2.10-3.2.el7.x86_64.rpm is not installed
(2/4): containerd.io-1.2.10-3.2.el7.x86_64.rpm
      |  23 MB  00:00:03
(3/4): docker-ce-18.09.9-3.el7.x86_64.rpm
       |  21 MB  00:00:04
(4/4): docker-ce-cli-18.09.9-3.el7.x86_64.rpm
       |  16 MB  00:00:01

Complete!
[root@centos7 ~]#docker version
Client:
 Version:           18.09.9
 API version:       1.39
 Go version:        go1.11.13
 Git commit:        039a7df9ba
 Built:             Wed Sep  4 16:51:21 2019
 OS/Arch:           linux/amd64
 Experimental:      false
Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the
docker daemon running?
[root@centos7 ~]#systemctl enable --now docker
[root@centos7 ~]#docker version
Client:
 Version:           18.09.9
```

```
 API version:       1.39
 Go version:        go1.11.13
 Git commit:        039a7df9ba
 Built:             Wed Sep  4 16:51:21 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.9
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.11.13
  Git commit:       039a7df
  Built:            Wed Sep  4 16:22:32 2019
  OS/Arch:          linux/amd64
  Experimental:     false
[root@centos7 ~]#
```

范例: 在CentOS8安装docker

```
[root@centos8 ~]#tee  /etc/yum.repos.d/docker.repo  <<EOF
[docker]
name=docker
gpgcheck=0
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/8/x86_64/stable/
EOF

[root@centos8 ~]#dnf -y install docker-ce
```

1.2.2.3 Linux 二进制安装

本方法适用于无法上网或无法通过包安装方式安装的主机上安装docker

安装文档: https://docs.docker.com/install/linux/docker-ce/binaries/

二进制安装下载路径

https://download.docker.com/linux/

https://mirrors.aliyun.com/docker-ce/linux/static/stable/x86_64/

范例: 在CentOS8上实现二进制安装docker

```
[root@centos8 ~]#wget
https://download.docker.com/linux/static/stable/x86_64/docker-19.03.5.tgz
--2020-01-24 18:02:55--
https://download.docker.com/linux/static/stable/x86_64/docker-19.03.5.tgz
Resolving download.docker.com (download.docker.com)... 143.204.83.37,
143.204.83.32, 143.204.83.95, ...
Connecting to download.docker.com (download.docker.com)|143.204.83.37|:443...
connected.
HTTP request sent, awaiting response... 200 OK
Length: 63252595 (60M) [application/x-tar]
Saving to: ‘docker-19.03.5.tgz’
```

```
docker-19.03.5.tgz          100%[=======================================>]
 60.32M  11.3MB/s    in 5.6s

2020-01-24 18:03:02 (10.7 MB/s) - ‘docker-19.03.5.tgz’ saved [63252595/63252595]

[root@centos8 ~]#tar xvf docker-19.03.5.tgz
docker/
docker/docker-init
docker/docker
docker/dockerd
docker/runc
docker/ctr
docker/docker-proxy
docker/containerd
docker/containerd-shim
[root@centos8 ~]#cp docker/* /usr/bin/


#启动dockerd服务
[root@centos8 ~]#dockerd &>/dev/null &


[root@centos8 ~]#docker version
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:22:05 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
  Git commit:       633a0ea838
  Built:            Wed Nov 13 07:28:45 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          v1.2.10
  GitCommit:        b34a5c8af56e510852c35414db4c1f4fa6172339
 runc:
  Version:          1.0.0-rc8+dev
  GitCommit:        3e425f80a8c931f88e6d94a8c831b9d5aa481657
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
[root@centos8 ~]#docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:9572f7cdcee8591948c2963463447a53466950b3fc15a247fcad1917ca215a2f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
[root@centos8 ~]#pstree -p
systemd(1)─┬─NetworkManager(660)─┬─{NetworkManager}(669)
           |                     └─{NetworkManager}(671)
           ├─VGAuthService(662)
           ├─agetty(718)
           ├─atd(712)
           ├─auditd(625)───{auditd}(627)
           ├─automount(905)─┬─{automount}(912)
           |                ├─{automount}(913)
           |                ├─{automount}(930)
           |                └─{automount}(937)
           ├─containerd(679)─┬─{containerd}(693)
           |                 ├─{containerd}(694)
           |                 ├─{containerd}(696)
           |                 ├─{containerd}(704)
           |                 ├─{containerd}(705)
           |                 ├─{containerd}(707)
           |                 └─{containerd}(708)
           ├─crond(713)
           ├─dbus-daemon(658)
           ├─dockerd(908)─┬─{dockerd}(922)
           |              ├─{dockerd}(923)
           |              ├─{dockerd}(925)
           |              ├─{dockerd}(944)
           |              ├─{dockerd}(1028)
           |              ├─{dockerd}(1100)
           |              └─{dockerd}(1114)
           ├─polkitd(659)─┬─{polkitd}(670)
           |              ├─{polkitd}(672)
           |              ├─{polkitd}(677)
           |              ├─{polkitd}(678)
           |              └─{polkitd}(701)
           ├─rngd(664)───{rngd}(666)
           ├─rsyslogd(906)─┬─{rsyslogd}(911)
           |               └─{rsyslogd}(914)
           ├─sshd(675)───sshd(1370)───sshd(1382)───bash(1383)───pstree(1441)
           ├─sssd(661)─┬─sssd_be(688)
           |           └─sssd_nss(703)
           ├─systemd(1373)───(sd-pam)(1376)
```

```
            ├─systemd-journal(551)
            ├─systemd-logind(709)
            ├─systemd-udevd(580)
            ├─tuned(674)─┬─{tuned}(915)
            |            ├─{tuned}(934)
            |            └─{tuned}(948)
            └─vmtoolsd(663)
```

范例: 创建相关的service文件,此方式新版有问题

```
#创建相关的service文件,此方式新版有问题
[root@centos8 ~]#groupadd -r docker

#将Ubuntu1804或CentOS7基于包方式安装的相关文件复制到相应目录下
[root@ubuntu1804 ~]#ll /lib/systemd/system/docker.*
-rw-r--r-- 1 root root 1683 Jun 22 23:44 /lib/systemd/system/docker.service
-rw-r--r-- 1 root root  197 Jun 22 23:44 /lib/systemd/system/docker.socket
[root@ubuntu1804 ~]#ll /lib/systemd/system/containerd.service
-rw-r--r-- 1 root root 1085 May  2  2020 /lib/systemd/system/containerd.service
[root@ubuntu1804 ~]#cat /lib/systemd/system/docker.socket
[Unit]
Description=Docker Socket for the API
PartOf=docker.service

[Socket]
ListenStream=/var/run/docker.sock
SocketMode=0660
SocketUser=root
SocketGroup=docker

[Install]
WantedBy=sockets.target

[root@ubuntu1804 ~]#cat /lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues
still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

# Note that StartLimit* options were moved from "Service" to "Unit" in systemd
229.
```

```
# Both the old, and new location are accepted by systemd 229 and up, so using
the old location
# to make them work for either version of systemd.
StartLimitBurst=3

# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd
230.
# Both the old, and new name are accepted by systemd 230 and up, so using the
old name to make
# this option work for either version of systemd.
StartLimitInterval=60s

# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity

# Comment TasksMax if your systemd version does not support it.
# Only systemd 226 and above support this option.
TasksMax=infinity

# set delegate yes so that systemd does not reset the cgroups of docker
containers
Delegate=yes

# kill only the docker process, not all processes in the cgroup
KillMode=process

[Install]
WantedBy=multi-user.target

[root@ubuntu1804 ~]#cat  /lib/systemd/system/containerd.service
#   Copyright 2018-2020 Docker Inc.

#   Licensed under the Apache License, Version 2.0 (the "License");
#   you may not use this file except in compliance with the License.
#   You may obtain a copy of the License at

#       http://www.apache.org/licenses/LICENSE-2.0

#   Unless required by applicable law or agreed to in writing, software
#   distributed under the License is distributed on an "AS IS" BASIS,
#   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
#   See the License for the specific language governing permissions and
#   limitations under the License.

[Unit]
Description=containerd container runtime
Documentation=https://containerd.io
After=network.target

[Service]
ExecStartPre=-/sbin/modprobe overlay
ExecStart=/usr/bin/containerd
KillMode=process
Delegate=yes
LimitNOFILE=1048576
```

```
# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNPROC=infinity
LimitCORE=infinity
TasksMax=infinity

[Install]
WantedBy=multi-user.target


[root@ubuntu1804 ~]#scp /lib/systemd/system/docker.*
/lib/systemd/system/containerd.service  10.0.0.8:/lib/systemd/system/
The authenticity of host '10.0.0.8 (10.0.0.8)' can't be established.
ECDSA key fingerprint is SHA256:8mUO3Wy13Ktt5pRBKaOU40avmw1xOgH5XTPK48CEWoM.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.0.8' (ECDSA) to the list of known hosts.
root@10.0.0.8's password:
docker.service                                          100% 1683    650.8KB/s
  00:00  docker.socket                                          100%  197
303.3KB/s   00:00
containerd.service                                      100%  487    516.6KB/s
  00:00

[root@centos8 ~]#systemctl daemon-reload
[root@centos8 ~]#systemctl enable --now docker
```

1.2.2.4 安装 podman

范例: 在CentOS8上安装podman

```
#在CentOS8上安装docker会自动安装podman,docker工具只是一个脚本，调用了Podman
[root@centos8 ~]#dnf install docker
[root@centos8 ~]#rpm -ql podman-docker
/usr/bin/docker
[root@centos8 ~]#cat /usr/bin/docker
#!/bin/sh
[ -f /etc/containers/nodocker ] || \
echo "Emulate Docker CLI using podman. Create /etc/containers/nodocker to quiet
msg." >&2
exec /usr/bin/podman "$@"
[root@centos8 ~]#podman version
Version:            1.4.2-stable2
RemoteAPI Version:  1
Go Version:         go1.12.8
OS/Arch:            linux/amd64

#修改拉取镜像的地址的顺序，提高速度
[root@centos8 ~]#vim /etc/containers/registries.conf
[registries.search]
registries = ['docker.io', 'quay.io', 'registry.redhat.io',
'registry.access.redhat.com']
```

1.2.2.5 在不同系统上实现一键安装 docker 脚本

1.2.2.5.1 基于 ubuntu 1804 的 一键安装 docker 脚本

```
[root@ubuntu1804 ~]#cat install_docker_ubuntu.sh
#!/bin/bash
#Description: Install docker on Ubuntu1804
#Version:1.0
#Date:2020-01-22

COLOR="echo -e \\033[1;31m"
END="\033[m"
DOCKER_VERSION="5:19.03.5~3-0~ubuntu-bionic"

install_docker(){
dpkg -s docker-ce &> /dev/null && ${COLOR}"Docker已安装，退出"${END} && exit
apt update
apt  -y install apt-transport-https ca-certificates curl software-properties-
common
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key
add -
add-apt-repository "deb [arch=amd64] https://mirrors.aliyun.com/docker-
ce/linux/ubuntu $(lsb_release -cs) stable"
apt update
${COLOR}"Docker有以下版本"${END}
apt-cache madison docker-ce
${COLOR}"5秒后即将安装: docker-"${DOCKER_VERSION}" 版本....."${END}
${COLOR}"如果想安装其它Docker版本，请按ctrl+c键退出，修改版本再执行"${END}
sleep 5

apt -y  install docker-ce=${DOCKER_VERSION} docker-ce-cli=${DOCKER_VERSION}

mkdir -p /etc/docker
tee /etc/docker/daemon.json <<-'EOF'
{
      "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}
EOF
systemctl daemon-reload
systemctl enable --now docker
docker version && ${COLOR}"Docker 安装成功"${END} ||  ${COLOR}"Docker 安装失
败"${END}
}
install_docker
```

1.2.2.5.2 基于 CentOS 8 实现一键安装 docker 脚本

1.2.2.5.2.1 脚本1

利用阿里云的基于CentOS8的docker yum源实现

```
#!/bin/bash
#
#**************************************************************************
#Author:        wangxiaochun
#QQ:            29308620
#Date:          2020-11-13
#FileName:       install_docker_for_centos8.sh
#URL:           http://www.wangxiaochun.com
```

```
#Description:       The test script
#Copyright (C):     2020 All rights reserved
#*********************************************************************


. /etc/init.d/functions
COLOR="echo -e \\E[1;32m"
END="\\E[0m"
DOCKER_VERSION="-19.03.13-3.el8"


install_docker() {

    ${COLOR}"开始安装 Docker....."${END}
    sleep 1
    cat >  /etc/yum.repos.d/docker.repo  <<EOF
[docker]
name=docker
gpgcheck=0
baseurl=https://mirrors.aliyun.com/docker-ce/linux/centos/8/x86_64/stable/
EOF

    yum clean all
    yum -y install docker-ce$DOCKER_VERSION docker-ce-cli$DOCKER_VERSION  \
        || { ${COLOR}"Base,Extras的yum源失败,请检查yum源配置"${END};exit; }

    mkdir -p /etc/docker
    cat > /etc/docker/daemon.json <<EOF
{
 "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
 }
EOF

    systemctl enable --now docker
    docker version && ${COLOR}"Docker安装成功"${END} || ${COLOR}"Docker安装失
败"${END}

}

rpm -q docker-ce &> /dev/null && action "Docker已安装" || install_docker
```

1.2.2.5.2.2 脚本2

早期CentOS8无yum仓库,可以利用下面脚本安装docker

```
#!/bin/bash
#
#*********************************************************************
#Author:        wangxiaochun
#QQ:            29308620
#Date:          2020-02-27
#FileName:          install_docker_for_centos8.sh
#URL:           http://www.magedu.com
#Description:       The test script
```

```
#Copyright (C):     2020 All rights reserved
#******************************************************************
. /etc/init.d/functions
COLOR="echo -e \\E[1;32m"
END="\\E[0m"
DOCKER_VERSION="-19.03.8-3.el7"


install_docker() {

    ${COLOR}"开始安装 Docker....."${END}
    sleep 1

    wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo  || { ${COLOR}"
互联网连接失败，请检查网络配置!"${END};exit; }
    yum clean all
    dnf -y install https://mirrors.aliyun.com/docker-
ce/linux/centos/7/x86_64/stable/Packages/containerd.io-1.2.13-3.1.el7.x86_64.rpm
    yum -y install docker-ce$DOCKER_VERSION docker-ce-cli$DOCKER_VERSION  \
        || { ${COLOR}"Base,Extras的yum源失败,请检查yum源配置"${END};exit; }

    mkdir -p /etc/docker
    cat > /etc/docker/daemon.json <<EOF
{
 "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
 }
EOF

    systemctl enable --now docker
    docker version && ${COLOR}"Docker安装成功"${END} || ${COLOR}"Docker安装失
败"${END}

}

rpm -q docker &> /dev/null && action "Docker已安装" || install_docker
```

1.2.2.5.3 基于 CentOS 7 实现一键安装docker 脚本

```
[root@centos7 ~]#cat install_docker_for_centos7.sh
#!/bin/bash
#
#******************************************************************
#Author:        wangxiaochun
#QQ:            29308620
#Date:          2020-01-26
#FileName:          install_docker_for_centos7.sh
#URL:           http://www.magedu.com
#Description:       The test script
#Copyright (C):     2020 All rights reserved
#******************************************************************
COLOR="echo -e \\033[1;31m"
END="\033[m"
VERSION="19.03.5-3.el7"
```

```
wget -P /etc/yum.repos.d/ https://mirrors.aliyun.com/docker-
ce/linux/centos/docker-ce.repo  || { ${COLOR}"互联网连接失败，请检查网络配
置!"${END};exit; }
yum clean all
yum -y install docker-ce-$VERSION docker-ce-cli-$VERSION || {
${COLOR}"Base,Extras的yum源失败,请检查yum源配置"${END};exit; }

#使用阿里做镜像加速
mkdir -p /etc/docker
cat > /etc/docker/daemon.json <<EOF
{
 "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
 }
EOF

systemctl enable --now docker
docker version && ${COLOR}"Docker安装成功"${END} || ${COLOR}"Docker安装失败"${END}
```

1.2.3 docker 程序环境

环境配置文件:

```
/etc/sysconfig/docker-network
/etc/sysconfig/docker-storage
/etc/sysconfig/docker
```

Unit File:

```
/usr/lib/systemd/system/docker.service
```

docker-ce 配置文件:

```
/etc/docker/daemon.json
```

Docker Registry配置文件:

```
/etc/containers/registries.conf
```

范例: ubuntu 查看docker相关文件

```
#服务器端相关文件
[root@ubuntu1804 ~]#dpkg -L docker-ce
/.
/etc
/etc/default
/etc/default/docker
/etc/init
/etc/init/docker.conf
/etc/init.d
/etc/init.d/docker
/lib
/lib/systemd
```

```
/lib/systemd/system
/lib/systemd/system/docker.service
/lib/systemd/system/docker.socket
/usr
/usr/bin
/usr/bin/docker-init
/usr/bin/docker-proxy
/usr/bin/dockerd
/usr/share
/usr/share/doc
/usr/share/doc/docker-ce
/usr/share/doc/docker-ce/README.md
/usr/share/doc/docker-ce/changelog.Debian.gz
/var
/var/lib
/var/lib/docker-engine
/var/lib/docker-engine/distribution_based_engine.json


#客户端相关文件
[root@ubuntu1804 ~]#dpkg -L docker-ce-cli
/.
/usr
/usr/bin
/usr/bin/docker
/usr/libexec
/usr/libexec/docker
/usr/libexec/docker/cli-plugins
/usr/libexec/docker/cli-plugins/docker-app
/usr/libexec/docker/cli-plugins/docker-buildx
/usr/share
/usr/share/bash-completion
/usr/share/bash-completion/completions
/usr/share/bash-completion/completions/docker
/usr/share/doc
/usr/share/doc/docker-ce-cli
/usr/share/doc/docker-ce-cli/changelog.Debian.gz
/usr/share/fish
/usr/share/fish/vendor_completions.d
/usr/share/fish/vendor_completions.d/docker.fish
/usr/share/man
/usr/share/man/man1
/usr/share/man/man1/docker-attach.1.gz
/usr/share/man/man1/docker-build.1.gz
/usr/share/man/man1/docker-builder-build.1.gz
/usr/share/man/man1/docker-builder-prune.1.gz
/usr/share/man/man1/docker-builder.1.gz
/usr/share/man/man1/docker-checkpoint-create.1.gz
/usr/share/man/man1/docker-checkpoint-ls.1.gz
/usr/share/man/man1/docker-checkpoint-rm.1.gz
/usr/share/man/man1/docker-checkpoint.1.gz
/usr/share/man/man1/docker-commit.1.gz
/usr/share/man/man1/docker-config-create.1.gz
/usr/share/man/man1/docker-config-inspect.1.gz
/usr/share/man/man1/docker-config-ls.1.gz
/usr/share/man/man1/docker-config-rm.1.gz
/usr/share/man/man1/docker-config.1.gz
/usr/share/man/man1/docker-container-attach.1.gz
```

```
/usr/share/man/man1/docker-container-commit.1.gz
/usr/share/man/man1/docker-container-cp.1.gz
/usr/share/man/man1/docker-container-create.1.gz
/usr/share/man/man1/docker-container-diff.1.gz
/usr/share/man/man1/docker-container-exec.1.gz
/usr/share/man/man1/docker-container-export.1.gz
/usr/share/man/man1/docker-container-inspect.1.gz
/usr/share/man/man1/docker-container-kill.1.gz
/usr/share/man/man1/docker-container-logs.1.gz
/usr/share/man/man1/docker-container-ls.1.gz
/usr/share/man/man1/docker-container-pause.1.gz
/usr/share/man/man1/docker-container-port.1.gz
/usr/share/man/man1/docker-container-prune.1.gz
/usr/share/man/man1/docker-container-rename.1.gz
/usr/share/man/man1/docker-container-restart.1.gz
/usr/share/man/man1/docker-container-rm.1.gz
/usr/share/man/man1/docker-container-run.1.gz
/usr/share/man/man1/docker-container-start.1.gz
/usr/share/man/man1/docker-container-stats.1.gz
/usr/share/man/man1/docker-container-stop.1.gz
/usr/share/man/man1/docker-container-top.1.gz
/usr/share/man/man1/docker-container-unpause.1.gz
/usr/share/man/man1/docker-container-update.1.gz
/usr/share/man/man1/docker-container-wait.1.gz
/usr/share/man/man1/docker-container.1.gz
/usr/share/man/man1/docker-context-create.1.gz
/usr/share/man/man1/docker-context-export.1.gz
/usr/share/man/man1/docker-context-import.1.gz
/usr/share/man/man1/docker-context-inspect.1.gz
/usr/share/man/man1/docker-context-ls.1.gz
/usr/share/man/man1/docker-context-rm.1.gz
/usr/share/man/man1/docker-context-update.1.gz
/usr/share/man/man1/docker-context-use.1.gz
/usr/share/man/man1/docker-context.1.gz
/usr/share/man/man1/docker-cp.1.gz
/usr/share/man/man1/docker-create.1.gz
/usr/share/man/man1/docker-deploy.1.gz
/usr/share/man/man1/docker-diff.1.gz
/usr/share/man/man1/docker-engine-activate.1.gz
/usr/share/man/man1/docker-engine-check.1.gz
/usr/share/man/man1/docker-engine-update.1.gz
/usr/share/man/man1/docker-engine.1.gz
/usr/share/man/man1/docker-events.1.gz
/usr/share/man/man1/docker-exec.1.gz
/usr/share/man/man1/docker-export.1.gz
/usr/share/man/man1/docker-history.1.gz
/usr/share/man/man1/docker-image-build.1.gz
/usr/share/man/man1/docker-image-history.1.gz
/usr/share/man/man1/docker-image-import.1.gz
/usr/share/man/man1/docker-image-inspect.1.gz
/usr/share/man/man1/docker-image-load.1.gz
/usr/share/man/man1/docker-image-ls.1.gz
/usr/share/man/man1/docker-image-prune.1.gz
/usr/share/man/man1/docker-image-pull.1.gz
/usr/share/man/man1/docker-image-push.1.gz
/usr/share/man/man1/docker-image-rm.1.gz
/usr/share/man/man1/docker-image-save.1.gz
/usr/share/man/man1/docker-image-tag.1.gz
```

```
/usr/share/man/man1/docker-image.1.gz
/usr/share/man/man1/docker-images.1.gz
/usr/share/man/man1/docker-import.1.gz
/usr/share/man/man1/docker-info.1.gz
/usr/share/man/man1/docker-inspect.1.gz
/usr/share/man/man1/docker-kill.1.gz
/usr/share/man/man1/docker-load.1.gz
/usr/share/man/man1/docker-login.1.gz
/usr/share/man/man1/docker-logout.1.gz
/usr/share/man/man1/docker-logs.1.gz
/usr/share/man/man1/docker-manifest-annotate.1.gz
/usr/share/man/man1/docker-manifest-create.1.gz
/usr/share/man/man1/docker-manifest-inspect.1.gz
/usr/share/man/man1/docker-manifest-push.1.gz
/usr/share/man/man1/docker-manifest.1.gz
/usr/share/man/man1/docker-network-connect.1.gz
/usr/share/man/man1/docker-network-create.1.gz
/usr/share/man/man1/docker-network-disconnect.1.gz
/usr/share/man/man1/docker-network-inspect.1.gz
/usr/share/man/man1/docker-network-ls.1.gz
/usr/share/man/man1/docker-network-prune.1.gz
/usr/share/man/man1/docker-network-rm.1.gz
/usr/share/man/man1/docker-network.1.gz
/usr/share/man/man1/docker-node-demote.1.gz
/usr/share/man/man1/docker-node-inspect.1.gz
/usr/share/man/man1/docker-node-ls.1.gz
/usr/share/man/man1/docker-node-promote.1.gz
/usr/share/man/man1/docker-node-ps.1.gz
/usr/share/man/man1/docker-node-rm.1.gz
/usr/share/man/man1/docker-node-update.1.gz
/usr/share/man/man1/docker-node.1.gz
/usr/share/man/man1/docker-pause.1.gz
/usr/share/man/man1/docker-plugin-create.1.gz
/usr/share/man/man1/docker-plugin-disable.1.gz
/usr/share/man/man1/docker-plugin-enable.1.gz
/usr/share/man/man1/docker-plugin-inspect.1.gz
/usr/share/man/man1/docker-plugin-install.1.gz
/usr/share/man/man1/docker-plugin-ls.1.gz
/usr/share/man/man1/docker-plugin-push.1.gz
/usr/share/man/man1/docker-plugin-rm.1.gz
/usr/share/man/man1/docker-plugin-set.1.gz
/usr/share/man/man1/docker-plugin-upgrade.1.gz
/usr/share/man/man1/docker-plugin.1.gz
/usr/share/man/man1/docker-port.1.gz
/usr/share/man/man1/docker-ps.1.gz
/usr/share/man/man1/docker-pull.1.gz
/usr/share/man/man1/docker-push.1.gz
/usr/share/man/man1/docker-rename.1.gz
/usr/share/man/man1/docker-restart.1.gz
/usr/share/man/man1/docker-rm.1.gz
/usr/share/man/man1/docker-rmi.1.gz
/usr/share/man/man1/docker-run.1.gz
/usr/share/man/man1/docker-save.1.gz
/usr/share/man/man1/docker-search.1.gz
/usr/share/man/man1/docker-secret-create.1.gz
/usr/share/man/man1/docker-secret-inspect.1.gz
/usr/share/man/man1/docker-secret-ls.1.gz
/usr/share/man/man1/docker-secret-rm.1.gz
```

```
/usr/share/man/man1/docker-secret.1.gz
/usr/share/man/man1/docker-service-create.1.gz
/usr/share/man/man1/docker-service-inspect.1.gz
/usr/share/man/man1/docker-service-logs.1.gz
/usr/share/man/man1/docker-service-ls.1.gz
/usr/share/man/man1/docker-service-ps.1.gz
/usr/share/man/man1/docker-service-rm.1.gz
/usr/share/man/man1/docker-service-rollback.1.gz
/usr/share/man/man1/docker-service-scale.1.gz
/usr/share/man/man1/docker-service-update.1.gz
/usr/share/man/man1/docker-service.1.gz
/usr/share/man/man1/docker-stack-deploy.1.gz
/usr/share/man/man1/docker-stack-ls.1.gz
/usr/share/man/man1/docker-stack-ps.1.gz
/usr/share/man/man1/docker-stack-rm.1.gz
/usr/share/man/man1/docker-stack-services.1.gz
/usr/share/man/man1/docker-stack.1.gz
/usr/share/man/man1/docker-start.1.gz
/usr/share/man/man1/docker-stats.1.gz
/usr/share/man/man1/docker-stop.1.gz
/usr/share/man/man1/docker-swarm-ca.1.gz
/usr/share/man/man1/docker-swarm-init.1.gz
/usr/share/man/man1/docker-swarm-join-token.1.gz
/usr/share/man/man1/docker-swarm-join.1.gz
/usr/share/man/man1/docker-swarm-leave.1.gz
/usr/share/man/man1/docker-swarm-unlock-key.1.gz
/usr/share/man/man1/docker-swarm-unlock.1.gz
/usr/share/man/man1/docker-swarm-update.1.gz
/usr/share/man/man1/docker-swarm.1.gz
/usr/share/man/man1/docker-system-df.1.gz
/usr/share/man/man1/docker-system-events.1.gz
/usr/share/man/man1/docker-system-info.1.gz
/usr/share/man/man1/docker-system-prune.1.gz
/usr/share/man/man1/docker-system.1.gz
/usr/share/man/man1/docker-tag.1.gz
/usr/share/man/man1/docker-top.1.gz
/usr/share/man/man1/docker-trust-inspect.1.gz
/usr/share/man/man1/docker-trust-key-generate.1.gz
/usr/share/man/man1/docker-trust-key-load.1.gz
/usr/share/man/man1/docker-trust-key.1.gz
/usr/share/man/man1/docker-trust-revoke.1.gz
/usr/share/man/man1/docker-trust-sign.1.gz
/usr/share/man/man1/docker-trust-signer-add.1.gz
/usr/share/man/man1/docker-trust-signer-remove.1.gz
/usr/share/man/man1/docker-trust-signer.1.gz
/usr/share/man/man1/docker-trust.1.gz
/usr/share/man/man1/docker-unpause.1.gz
/usr/share/man/man1/docker-update.1.gz
/usr/share/man/man1/docker-version.1.gz
/usr/share/man/man1/docker-volume-create.1.gz
/usr/share/man/man1/docker-volume-inspect.1.gz
/usr/share/man/man1/docker-volume-ls.1.gz
/usr/share/man/man1/docker-volume-prune.1.gz
/usr/share/man/man1/docker-volume-rm.1.gz
/usr/share/man/man1/docker-volume.1.gz
/usr/share/man/man1/docker-wait.1.gz
/usr/share/man/man1/docker.1.gz
/usr/share/man/man5
```

```
/usr/share/man/man5/Dockerfile.5.gz
/usr/share/man/man5/docker-config-json.5.gz
/usr/share/man/man8
/usr/share/man/man8/dockerd.8.gz
/usr/share/zsh
/usr/share/zsh/vendor-completions
/usr/share/zsh/vendor-completions/_docker
```

范例: CentOS7 查看docker相关文件

```
[root@centos7 ~]#rpm -ql docker-ce
/usr/bin/docker-init
/usr/bin/docker-proxy
/usr/bin/dockerd
/usr/lib/systemd/system/docker.service
/usr/lib/systemd/system/docker.socket
[root@centos7 ~]#rpm -ql docker-ce-cli
/usr/bin/docker
/usr/libexec/docker/cli-plugins/docker-app
/usr/libexec/docker/cli-plugins/docker-buildx
/usr/share/bash-completion/completions/docker
/usr/share/doc/docker-ce-cli-19.03.12
/usr/share/doc/docker-ce-cli-19.03.12/LICENSE
/usr/share/doc/docker-ce-cli-19.03.12/MAINTAINERS
/usr/share/doc/docker-ce-cli-19.03.12/NOTICE
/usr/share/doc/docker-ce-cli-19.03.12/README.md
/usr/share/fish/vendor_completions.d/docker.fish
/usr/share/man/man1/docker-attach.1.gz
/usr/share/man/man1/docker-build.1.gz
/usr/share/man/man1/docker-builder-build.1.gz
/usr/share/man/man1/docker-builder-prune.1.gz
/usr/share/man/man1/docker-builder.1.gz
/usr/share/man/man1/docker-checkpoint-create.1.gz
/usr/share/man/man1/docker-checkpoint-ls.1.gz
/usr/share/man/man1/docker-checkpoint-rm.1.gz
/usr/share/man/man1/docker-checkpoint.1.gz
/usr/share/man/man1/docker-commit.1.gz
/usr/share/man/man1/docker-config-create.1.gz
/usr/share/man/man1/docker-config-inspect.1.gz
/usr/share/man/man1/docker-config-ls.1.gz
/usr/share/man/man1/docker-config-rm.1.gz
/usr/share/man/man1/docker-config.1.gz
/usr/share/man/man1/docker-container-attach.1.gz
/usr/share/man/man1/docker-container-commit.1.gz
/usr/share/man/man1/docker-container-cp.1.gz
/usr/share/man/man1/docker-container-create.1.gz
/usr/share/man/man1/docker-container-diff.1.gz
/usr/share/man/man1/docker-container-exec.1.gz
/usr/share/man/man1/docker-container-export.1.gz
/usr/share/man/man1/docker-container-inspect.1.gz
/usr/share/man/man1/docker-container-kill.1.gz
/usr/share/man/man1/docker-container-logs.1.gz
/usr/share/man/man1/docker-container-ls.1.gz
/usr/share/man/man1/docker-container-pause.1.gz
/usr/share/man/man1/docker-container-port.1.gz
/usr/share/man/man1/docker-container-prune.1.gz
```

```
/usr/share/man/man1/docker-container-rename.1.gz
/usr/share/man/man1/docker-container-restart.1.gz
/usr/share/man/man1/docker-container-rm.1.gz
/usr/share/man/man1/docker-container-run.1.gz
/usr/share/man/man1/docker-container-start.1.gz
/usr/share/man/man1/docker-container-stats.1.gz
/usr/share/man/man1/docker-container-stop.1.gz
/usr/share/man/man1/docker-container-top.1.gz
/usr/share/man/man1/docker-container-unpause.1.gz
/usr/share/man/man1/docker-container-update.1.gz
/usr/share/man/man1/docker-container-wait.1.gz
/usr/share/man/man1/docker-container.1.gz
/usr/share/man/man1/docker-context-create.1.gz
/usr/share/man/man1/docker-context-export.1.gz
/usr/share/man/man1/docker-context-import.1.gz
/usr/share/man/man1/docker-context-inspect.1.gz
/usr/share/man/man1/docker-context-ls.1.gz
/usr/share/man/man1/docker-context-rm.1.gz
/usr/share/man/man1/docker-context-update.1.gz
/usr/share/man/man1/docker-context-use.1.gz
/usr/share/man/man1/docker-context.1.gz
/usr/share/man/man1/docker-cp.1.gz
/usr/share/man/man1/docker-create.1.gz
/usr/share/man/man1/docker-deploy.1.gz
/usr/share/man/man1/docker-diff.1.gz
/usr/share/man/man1/docker-engine-activate.1.gz
/usr/share/man/man1/docker-engine-check.1.gz
/usr/share/man/man1/docker-engine-update.1.gz
/usr/share/man/man1/docker-engine.1.gz
/usr/share/man/man1/docker-events.1.gz
/usr/share/man/man1/docker-exec.1.gz
/usr/share/man/man1/docker-export.1.gz
/usr/share/man/man1/docker-history.1.gz
/usr/share/man/man1/docker-image-build.1.gz
/usr/share/man/man1/docker-image-history.1.gz
/usr/share/man/man1/docker-image-import.1.gz
/usr/share/man/man1/docker-image-inspect.1.gz
/usr/share/man/man1/docker-image-load.1.gz
/usr/share/man/man1/docker-image-ls.1.gz
/usr/share/man/man1/docker-image-prune.1.gz
/usr/share/man/man1/docker-image-pull.1.gz
/usr/share/man/man1/docker-image-push.1.gz
/usr/share/man/man1/docker-image-rm.1.gz
/usr/share/man/man1/docker-image-save.1.gz
/usr/share/man/man1/docker-image-tag.1.gz
/usr/share/man/man1/docker-image.1.gz
/usr/share/man/man1/docker-images.1.gz
/usr/share/man/man1/docker-import.1.gz
/usr/share/man/man1/docker-info.1.gz
/usr/share/man/man1/docker-inspect.1.gz
/usr/share/man/man1/docker-kill.1.gz
/usr/share/man/man1/docker-load.1.gz
/usr/share/man/man1/docker-login.1.gz
/usr/share/man/man1/docker-logout.1.gz
/usr/share/man/man1/docker-logs.1.gz
/usr/share/man/man1/docker-manifest-annotate.1.gz
/usr/share/man/man1/docker-manifest-create.1.gz
/usr/share/man/man1/docker-manifest-inspect.1.gz
```

```
/usr/share/man/man1/docker-manifest-push.1.gz
/usr/share/man/man1/docker-manifest.1.gz
/usr/share/man/man1/docker-network-connect.1.gz
/usr/share/man/man1/docker-network-create.1.gz
/usr/share/man/man1/docker-network-disconnect.1.gz
/usr/share/man/man1/docker-network-inspect.1.gz
/usr/share/man/man1/docker-network-ls.1.gz
/usr/share/man/man1/docker-network-prune.1.gz
/usr/share/man/man1/docker-network-rm.1.gz
/usr/share/man/man1/docker-network.1.gz
/usr/share/man/man1/docker-node-demote.1.gz
/usr/share/man/man1/docker-node-inspect.1.gz
/usr/share/man/man1/docker-node-ls.1.gz
/usr/share/man/man1/docker-node-promote.1.gz
/usr/share/man/man1/docker-node-ps.1.gz
/usr/share/man/man1/docker-node-rm.1.gz
/usr/share/man/man1/docker-node-update.1.gz
/usr/share/man/man1/docker-node.1.gz
/usr/share/man/man1/docker-pause.1.gz
/usr/share/man/man1/docker-plugin-create.1.gz
/usr/share/man/man1/docker-plugin-disable.1.gz
/usr/share/man/man1/docker-plugin-enable.1.gz
/usr/share/man/man1/docker-plugin-inspect.1.gz
/usr/share/man/man1/docker-plugin-install.1.gz
/usr/share/man/man1/docker-plugin-ls.1.gz
/usr/share/man/man1/docker-plugin-push.1.gz
/usr/share/man/man1/docker-plugin-rm.1.gz
/usr/share/man/man1/docker-plugin-set.1.gz
/usr/share/man/man1/docker-plugin-upgrade.1.gz
/usr/share/man/man1/docker-plugin.1.gz
/usr/share/man/man1/docker-port.1.gz
/usr/share/man/man1/docker-ps.1.gz
/usr/share/man/man1/docker-pull.1.gz
/usr/share/man/man1/docker-push.1.gz
/usr/share/man/man1/docker-rename.1.gz
/usr/share/man/man1/docker-restart.1.gz
/usr/share/man/man1/docker-rm.1.gz
/usr/share/man/man1/docker-rmi.1.gz
/usr/share/man/man1/docker-run.1.gz
/usr/share/man/man1/docker-save.1.gz
/usr/share/man/man1/docker-search.1.gz
/usr/share/man/man1/docker-secret-create.1.gz
/usr/share/man/man1/docker-secret-inspect.1.gz
/usr/share/man/man1/docker-secret-ls.1.gz
/usr/share/man/man1/docker-secret-rm.1.gz
/usr/share/man/man1/docker-secret.1.gz
/usr/share/man/man1/docker-service-create.1.gz
/usr/share/man/man1/docker-service-inspect.1.gz
/usr/share/man/man1/docker-service-logs.1.gz
/usr/share/man/man1/docker-service-ls.1.gz
/usr/share/man/man1/docker-service-ps.1.gz
/usr/share/man/man1/docker-service-rm.1.gz
/usr/share/man/man1/docker-service-rollback.1.gz
/usr/share/man/man1/docker-service-scale.1.gz
/usr/share/man/man1/docker-service-update.1.gz
/usr/share/man/man1/docker-service.1.gz
/usr/share/man/man1/docker-stack-deploy.1.gz
/usr/share/man/man1/docker-stack-ls.1.gz
```

```
/usr/share/man/man1/docker-stack-ps.1.gz
/usr/share/man/man1/docker-stack-rm.1.gz
/usr/share/man/man1/docker-stack-services.1.gz
/usr/share/man/man1/docker-stack.1.gz
/usr/share/man/man1/docker-start.1.gz
/usr/share/man/man1/docker-stats.1.gz
/usr/share/man/man1/docker-stop.1.gz
/usr/share/man/man1/docker-swarm-ca.1.gz
/usr/share/man/man1/docker-swarm-init.1.gz
/usr/share/man/man1/docker-swarm-join-token.1.gz
/usr/share/man/man1/docker-swarm-join.1.gz
/usr/share/man/man1/docker-swarm-leave.1.gz
/usr/share/man/man1/docker-swarm-unlock-key.1.gz
/usr/share/man/man1/docker-swarm-unlock.1.gz
/usr/share/man/man1/docker-swarm-update.1.gz
/usr/share/man/man1/docker-swarm.1.gz
/usr/share/man/man1/docker-system-df.1.gz
/usr/share/man/man1/docker-system-events.1.gz
/usr/share/man/man1/docker-system-info.1.gz
/usr/share/man/man1/docker-system-prune.1.gz
/usr/share/man/man1/docker-system.1.gz
/usr/share/man/man1/docker-tag.1.gz
/usr/share/man/man1/docker-top.1.gz
/usr/share/man/man1/docker-trust-inspect.1.gz
/usr/share/man/man1/docker-trust-key-generate.1.gz
/usr/share/man/man1/docker-trust-key-load.1.gz
/usr/share/man/man1/docker-trust-key.1.gz
/usr/share/man/man1/docker-trust-revoke.1.gz
/usr/share/man/man1/docker-trust-sign.1.gz
/usr/share/man/man1/docker-trust-signer-add.1.gz
/usr/share/man/man1/docker-trust-signer-remove.1.gz
/usr/share/man/man1/docker-trust-signer.1.gz
/usr/share/man/man1/docker-trust.1.gz
/usr/share/man/man1/docker-unpause.1.gz
/usr/share/man/man1/docker-update.1.gz
/usr/share/man/man1/docker-version.1.gz
/usr/share/man/man1/docker-volume-create.1.gz
/usr/share/man/man1/docker-volume-inspect.1.gz
/usr/share/man/man1/docker-volume-ls.1.gz
/usr/share/man/man1/docker-volume-prune.1.gz
/usr/share/man/man1/docker-volume-rm.1.gz
/usr/share/man/man1/docker-volume.1.gz
/usr/share/man/man1/docker-wait.1.gz
/usr/share/man/man1/docker.1.gz
/usr/share/man/man5/Dockerfile.5.gz
/usr/share/man/man5/docker-config-json.5.gz
/usr/share/man/man8/dockerd.8.gz
/usr/share/zsh/vendor-completions/_docker
```

1.2.4 docker 命令帮助

docker 命令是最常使用的docker 客户端命令，其后面可以加不同的参数以实现不同的功能

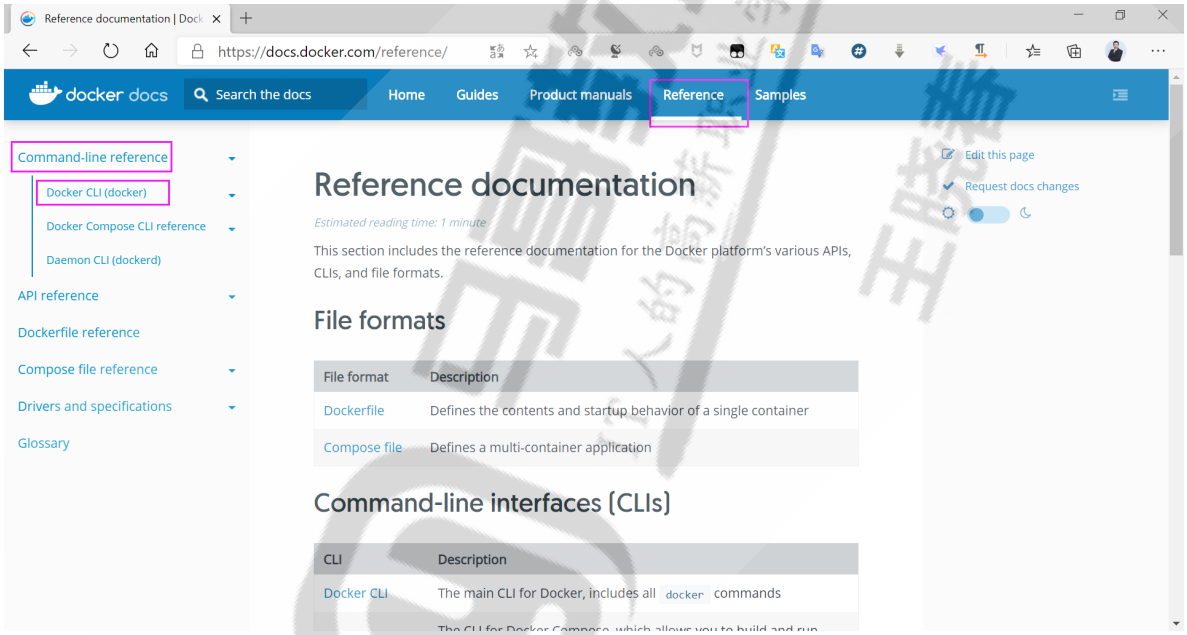
docker 命令格式

```
docker [OPTIONS] COMMAND

COMMAND分为
Management Commands   #指定管理的资源对象类型,较新的命令用法,将命令按资源类型进行分类,方便使用
Commands              #对不同资源操作的命令不分类,使用容易产生混乱
```

docker 命令有很多子命令，可以用下面方法查看帮助

```
#docker 命令帮助
man docker
docker
docker  --help

#docker  子命令帮助
man docker-COMMAND
docker COMMAND --help
```

官方文档: https://docs.docker.com/reference/



范例: 查看docker命令帮助

```
[root@ubuntu1804 ~]#docker --help

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default
"/root/.docker")
  -c, --context string     Name of the context to use to connect to the daemon
(overrides DOCKER_HOST env var and default
                           context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level
("debug"|"info"|"warn"|"error"|"fatal") (default "info")
```

```
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
"/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default
"/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default
"/root/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  engine      Manage the docker engine
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams 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 to files or directories on a container's
filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
```

```
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by
default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit
codes

Run 'docker COMMAND --help' for more information on a command.
```

1.2.5 查看 Docker 相关信息

1.2.5.1 查看 docker 版本

```
[root@ubuntu1804 ~]#docker version
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:29:52 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
  Git commit:       633a0ea838
  Built:            Wed Nov 13 07:28:22 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.10
  GitCommit:        b34a5c8af56e510852c35414db4c1f4fa6172339
 runc:
  Version:          1.0.0-rc8+dev
  GitCommit:        3e425f80a8c931f88e6d94a8c831b9d5aa481657
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

1.2.5.2 查看 docker 详解信息

```
[root@ubuntu1804 ~]#docker info
Client:
 Debug Mode: false      #client 端是否开启 debug

Server:
 Containers: 2    #当前主机运行的容器总数
  Running: 0      #有几个容器是正在运行的
  Paused: 0       #有几个容器是暂停的
  Stopped: 2      #有几个容器是停止的
 Images: 4        #当前服务器的镜像数
 Server Version: 19.03.5    #服务端版本
 Storage Driver: overlay2   #正在使用的存储引擎
  Backing Filesystem: extfs    #后端文件系统，即服务器的磁盘文件系统
  Supports d_type: true   #是否支持 d_type
  Native Overlay Diff: true   #是否支持差异数据存储
 Logging Driver: json-file    #日志类型
 Cgroup Driver: cgroupfs   #Cgroups 类型
 Plugins:                    #插件
  Volume: local              #卷
  Network: bridge host ipvlan macvlan null overlay # overlay 跨主机通信
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog  # 日志类型
 Swarm: inactive     #是否支持 swarm
 Runtimes: runc      #已安装的容器运行时
 Default Runtime: runc    #默认使用的容器运行时
 Init Binary: docker-init    #初始化容器的守护进程，即 pid 为 1 的进程
 containerd version: b34a5c8af56e510852c35414db4c1f4fa6172339 #版本
 runc version: 3e425f80a8c931f88e6d94a8c831b9d5aa481657   #runc 版本
 init version: fec3683   #init 版本
 Security Options:    #安全选项
  apparmor      #安全模块，https://docs.docker.com/engine/security/apparmor/
  seccomp   #安全计算模块，即制容器操作，
https://docs.docker.com/engine/security/seccomp/
   Profile: default   #默认的配置文件
 Kernel Version: 4.15.0-29-generic   #宿主机内核版本
 Operating System: Ubuntu 18.04.1 LTS   #宿主机操作系统
 OSType: linux     #宿主机操作系统类型
 Architecture: x86_64    #宿主机架构
 CPUs: 1     #宿主机 CPU 数量
 Total Memory: 962MiB    #宿主机总内存
 Name: ubuntu1804.magedu.org #宿主机 hostname
 ID: IZHJ:WPIN:BRMC:XQUI:VVVR:UVGK:NZBM:YQXT:JDWB:33RS:45V7:SQWJ #宿主机 ID
 Docker Root Dir: /var/lib/docker   #宿主机关于docker数据的保存目录
 Debug Mode: false    #server 端是否开启 debug
 Registry: https://index.docker.io/v1/   #仓库路径
 Labels:
 Experimental: false   #是否测试版
 Insecure Registries:
  127.0.0.0/8   : #非安全的镜像仓库
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/    #镜像仓库
 Live Restore Enabled: false   #是否开启活动重启 (重启docker-daemon 不关闭容器 )

WARNING: No swap limit support   #系统警告信息 (没有开启 swap 资源限制 )
```

范例: 解决上述SWAP报警提示

官方文档: https://docs.docker.com/install/linux/linux-postinstall/#your-kernel-does-not-support-cgroup-swap-limit-capabilities

```
[root@ubuntu1804 ~]#docker info
......
WARNING: No swap limit support

[root@ubuntu1804 ~]# vim /etc/default/grub
GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=hidden
GRUB_TIMEOUT=2
GRUB_DISTRIBUTOR=`lsb_ release -i -s 2> /dev/null || echo Debian`
GRUB_CMDLINE_LINUX_DEFAULT=""
GRUB_CMDLINE_LINUX="net.ifnames=0 biosdevname=0 swapaccount=1"  #修改此行

[root@ubuntu1804 ~]# update-grub
[root@ubuntu1804 ~]# reboot
```

1.2.5.3 查看 docker0 网卡

在docker安装启动之后，默认会生成一个名称为docker0的网卡并且默认IP地址为172.17.0.1的网卡

```
#ubuntu18.04安装docker后网卡配置
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
4: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:d3:26:ed:4e brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:d3ff:fe26:ed4e/64 scope link
       valid_lft forever preferred_lft forever

#CentOS 7.6 安装docker后网卡配置
[root@centos7 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 00:0c:29:ca:00:e4 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.7/24 brd 10.0.0.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:feca:e4/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:d2:81:c2:e0 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever

#CentOS 8.1 安装docker后网卡配置
[root@centos8 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:4d:ef:3e brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.18/24 brd 10.0.0.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe4d:ef3e/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:f5:3e:65:b6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
[root@centos8 ~]#route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    100    0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     100    0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
```

1.2.5.4 docker 存储引擎

官方文档关于存储引擎的相关文档:

https://docs.docker.com/storage/storagedriver/

https://docs.docker.com/storage/storagedriver/select-storage-driver/

- AUFS: （Advanced Mult-Layered Unification Filesystem，版本2之前旧称AnotherUnionFS）是一种 Union FS ，是文件级的存储驱动。Aufs是之前的UnionFS的重新实现，2006年由Junjiro Okajima开发

所谓 UnionFS就是把不同物理位置的目录合并 mount 到同一个目录中。简单来说就是支持将不同目录挂载到一个虚拟文件系统下的。这种可以层层地叠加修改文件。无论底下有多少都是只读的，最上系统可写的。当需要修改一个文件时， AUFS 创建该文件的一个副本，使用 CoW 将文件从只读层复制到可写进行修改，结果也保存在Docker 中，底下的只读层就是 image，可写层就是 Container

aufs 被拒绝合并到主线 Linux 。其代码被批评为"dense, unreadable, uncommented 密集、不可读、未注释"。 相反，OverlayFS被合并到 Linux 内核中。在多次尝试将 aufs 合并到主线内核失败后，作者放弃了

AUFS 是 Docker 18.06 及更早版本的首选存储驱动程序，在内核 3.13 上运行 Ubuntu 14.04 时不支持 overlay2

- Overlay：一种 Union FS 文件系统， Linux 内核 3.18 后支持
- Overlay2: Overlay 的升级版，到目前为止，所有 Linux 发行版推荐使用的存储类 型，也是docker默认使用的存储引擎为overlay2，需要磁盘分区支持d-type功能，因此需要系统磁盘的额外支持，相对AUFS来说Overlay2 有以下优势：更简单地设计； 从3.18开始就进入了Linux内核主线；资源消耗更少
- devicemapper：因为CentOS 7.2和RHEL 7.2 的之前版本内核版本不支持 overlay2，默认使用的存储驱动程序，最大数据容量只支持100GB且性能不佳，当前较新版本的CentOS 已经支持 overlay2， 因此推荐使用 overlay2,另外此存储引擎已在Docker Engine 18.09中弃用
- ZFS(Sun -2005)/btrfs(Oracle-2007)：目前没有广泛使用
- vfs：用于测试环境，适用于无法使用 copy-on -writewrite 时的情况。 此存储驱动程序的性能很差，通常不建议用于生产

修改存储引擎参考文档:

https://docs.docker.com/storage/storagedriver/overlayfs-driver/

范例: 在CentOS7.2修改存储引擎

```
[root@centos7 ~]#vim /lib/systemd/system/docker.service
.....
ExecStart=/usr/bin/dockerd -s overlay2 -H fd:// --
containerd=/run/containerd/containerd.sock
......

#创建新的xfs分区,添加ftype特性,否则默认无法启动docker服务
[root@centos7 ~]#mkfs.xfs  -n ftype=1 /dev/sdb
[root@centos7 ~]#mount /dev/sdb /var/lib/docker

[root@centos7 ~]#systemctl daemon-reload
[root@centos7 ~]#systemctl restart docker
```

注意:修改存储引擎会导致所有容器丢失,所以先备份再修改

```
#查看Ubuntu1804的默认存储引擎
[root@ubuntu1804 ~]#docker info |grep Storage
WARNING: No swap limit support
 Storage Driver: overlay2

#查看CentOS7.6的默认存储引擎
[root@centos7 ~]#docker info |grep Storage
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
Storage Driver: overlay2
```

Docker官方推荐首选存储引擎为overlay2，其次为devicemapper，但是devicemapper存在使用空间方面的一些限制，虽然可以通过后期配置解决，但是官方依然推荐使用overlay2，以下是生产故障事例:

https://www.cnblogs.com/youruncloud/p/5736718.html

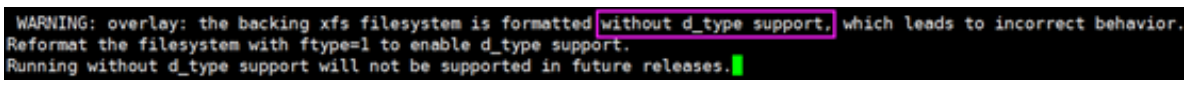
```
[root@centos7 ~]#xfs_info  /data
meta-data=/dev/mapper/centos-root isize=512    agcount=4, agsize=3276800 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=0 spinodes=0
data     =                       bsize=4096   blocks=13107200, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal               bsize=4096   blocks=6400, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
[root@centos7 ~]#
```

如果docker数据目录是一块单独的磁盘分区而且是xfs格式的，那么需要在格式化的时候加上参数-n ftype=1(启用此功能表示节点文件类型存入在目录结构中），示例: mkfs.xfs -n ftype=1 devname ，否则后期在无法启动容器,并会报错不支持 d_type

注意: ext4文件系统无需此d_type特性

```
[root@centos7 ~]#xfs_info  /data
meta-data=/dev/mapper/centos-root isize=512    agcount=4, agsize=3276800 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=0 spinodes=0
data     =                       bsize=4096   blocks=13107200, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0   #CentOS7.2之前版本此特性默认ftype=0
log      =internal               bsize=4096   blocks=6400, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
[root@centos7 ~]#
```

报错界面:



范例：aufs 实现联合文件系统挂载

```
[root@ubuntu1804 ~]#cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   ramfs
nodev   bdev
nodev   proc
nodev   cpuset
nodev   cgroup
nodev   cgroup2
nodev   tmpfs
nodev   devtmpfs
nodev   configfs
nodev   debugfs
nodev   tracefs
nodev   securityfs
nodev   sockfs
nodev   dax
nodev   bpf
nodev   pipefs
nodev   hugetlbfs
nodev   devpts
    ext3
    ext2
    ext4
    squashfs
    vfat
nodev   ecryptfs
    fuseblk
nodev   fuse
nodev   fusectl
nodev   pstore
nodev   mqueue
    btrfs
nodev   autofs
nodev   rpc_pipefs
nodev   nfsd
nodev   overlay
nodev   aufs
[root@ubuntu1804 ~]#grep -i aufs /boot/config-4.15.0-29-generic
CONFIG_AUFS_FS=m
CONFIG_AUFS_BRANCH_MAX_127=y
# CONFIG_AUFS_BRANCH_MAX_511 is not set
# CONFIG_AUFS_BRANCH_MAX_1023 is not set
# CONFIG_AUFS_BRANCH_MAX_32767 is not set
CONFIG_AUFS_SBILIST=y
# CONFIG_AUFS_HNOTIFY is not set
CONFIG_AUFS_EXPORT=y
CONFIG_AUFS_INO_T_64=y
CONFIG_AUFS_XATTR=y
# CONFIG_AUFS_FHSM is not set
# CONFIG_AUFS_RDU is not set
CONFIG_AUFS_DIRREN=y
# CONFIG_AUFS_SHWH is not set
# CONFIG_AUFS_BR_RAMFS is not set
# CONFIG_AUFS_BR_FUSE is not set
```

```
CONFIG_AUFS_BR_HFSPLUS=y
CONFIG_AUFS_BDEV_LOOP=y
# CONFIG_AUFS_DEBUG is not set
[root@ubuntu1804 ~]#mkdir dir{1,2}
[root@ubuntu1804 ~]#echo here is dir1 > dir1/file1
[root@ubuntu1804 ~]#echo here is dir2 > dir2/file2

[root@ubuntu1804 ~]#mkdir /data/aufs
[root@ubuntu1804 ~]#mount -t aufs -o br=/root/dir1=ro:/root/dir2=rw none
/data/aufs
[root@ubuntu1804 ~]#ll /data/aufs/
total 16
drwxr-xr-x 4 root root 4096 Jan 25 16:22 ./
drwxr-xr-x 4 root root 4096 Jan 25 16:22 ../
-rw-r--r-- 1 root root   13 Jan 25 16:22 file1
-rw-r--r-- 1 root root   13 Jan 25 16:22 file2
[root@ubuntu1804 ~]#cat /data/aufs/file1
here is dir1
[root@ubuntu1804 ~]#cat /data/aufs/file2
here is dir2
[root@ubuntu1804 ~]#df -T
Filesystem     Type     1K-blocks    Used Available Use% Mounted on
udev           devtmpfs    462560       0    462560   0% /dev
tmpfs          tmpfs        98512   10296     88216  11% /run
/dev/sda2      ext4      47799020 2770244  42570972   7% /
tmpfs          tmpfs       492552       0    492552   0% /dev/shm
tmpfs          tmpfs         5120       0      5120   0% /run/lock
tmpfs          tmpfs       492552       0    492552   0% /sys/fs/cgroup
/dev/sda3      ext4      19091540   45084  18053588   1% /data
/dev/sda1      ext4        944120   77112    801832   9% /boot
tmpfs          tmpfs        98508       0     98508   0% /run/user/0
none           aufs      47799020 2770244  42570972   7% /data/aufs
[root@ubuntu1804 ~]#echo write to file1 >> /data/aufs/file1
-bash: /data/aufs/file1: Read-only file system
[root@ubuntu1804 ~]#echo write to file2 >> /data/aufs/file2
[root@ubuntu1804 ~]#cat /data/aufs/file1
here is dir1
[root@ubuntu1804 ~]#cat /data/aufs/file2
here is dir2
write to file2
[root@ubuntu1804 ~]#umount /data/aufs
[root@ubuntu1804 ~]#mv dir1/file1 dir1/file2
[root@ubuntu1804 ~]#cat dir1/file2
here is dir1
[root@ubuntu1804 ~]#cat dir2/file2
here is dir2
write to file2
[root@ubuntu1804 ~]#mount -t aufs -o br=/root/dir1=ro:/root/dir2=rw none
/data/aufs
[root@ubuntu1804 ~]#ls /data/aufs -l
total 4
-rw-r--r-- 1 root root 13 Jan 25 16:22 file2
[root@ubuntu1804 ~]#cat /data/aufs/file2
here is dir1
[root@ubuntu1804 ~]#
```

范例：修改存储引擎

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx               latest              5ad3bd0e67a9        3 days ago
127MB
alpine              latest              e7d92cdc71fe        7 days ago
 5.59MB
centos              centos8.1.1911      470671670cac        7 days ago
237MB
centos              latest              470671670cac        7 days ago
237MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
hello-world         latest              fce289e99eb9        12 months ago
1.84kB
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
d4741f815199        busybox             "sh"                     41 hours ago
     Exited (137) 23 hours ago                      flamboyant_moser
5dee9be9afdb        nginx               "nginx -g 'daemon of…"   2 days ago
     Exited (0) 23 hours ago                        lucid_lichterman

[root@ubuntu1804 ~]#docker info |grep "Storage Driver"
 Storage Driver: overlay2
[root@ubuntu1804 ~]#systemctl stop docker
Warning: Stopping docker.service, but it can still be activated by:
  docker.socket
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
[root@ubuntu1804 ~]#cat /etc/docker/daemon.json
{
   "storage-driver": "aufs"
}

[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#docker info |grep aufs
WARNING: the aufs storage-driver is deprecated, and will be removed in a future
release.
 Storage Driver: aufs
  Root Dir: /var/lib/docker/aufs
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
{
   "storage-driver": "aufs"
}
[root@ubuntu1804 ~]#ls /var/lib/docker
aufs  builder  buildkit  containers  image  network  overlay2  plugins  runtimes
swarm  tmp  trust  volumes
[root@ubuntu1804 ~]#ls /var/lib/docker/aufs/
diff  layers  mnt
[root@ubuntu1804 ~]#ll /var/lib/docker/aufs/
total 20
```

```
drwx------  5 root root 4096 Jan 25 16:46 ./
drwx--x--x 15 root root 4096 Jan 25 16:46 ../
drwx------  2 root root 4096 Jan 25 16:46 diff/
drwx------  2 root root 4096 Jan 25 16:46 layers/
drwx------  2 root root 4096 Jan 25 16:46 mnt/
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
[root@ubuntu1804 ~]#cat /etc/docker/daemon.json
{
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}
[root@ubuntu1804 ~]#
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#ll /var/lib/docker/aufs/
total 20
drwx------  5 root root 4096 Jan 25 16:46 ./
drwx--x--x 15 root root 4096 Jan 25 16:48 ../
drwx------  2 root root 4096 Jan 25 16:46 diff/
drwx------  2 root root 4096 Jan 25 16:46 layers/
drwx------  2 root root 4096 Jan 25 16:46 mnt/
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
d4741f815199        busybox             "sh"                     41 hours ago
     Exited (137) 23 hours ago                       flamboyant_moser
5dee9be9afdb        nginx               "nginx -g 'daemon of…"   2 days ago
     Exited (0) 23 hours ago                         lucid_lichterman
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx               latest              5ad3bd0e67a9        3 days ago
127MB
alpine              latest              e7d92cdc71fe        7 days ago
 5.59MB
centos              centos8.1.1911      470671670cac        7 days ago
237MB
centos              latest              470671670cac        7 days ago
237MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
hello-world         latest              fce289e99eb9        12 months ago
1.84kB
[root@ubuntu1804 ~]#
```

1.2.5.5 docker 服务进程

通过查看docker进程，了解docker的运行及工作方式

1.2.5.5.1 查看宿主机进程树

```
[root@ubuntu1804 ~]#docker version
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
```

```
 Built:             Wed Nov 13 07:29:52 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
  Git commit:       633a0ea838
  Built:            Wed Nov 13 07:28:22 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.10
  GitCommit:        b34a5c8af56e510852c35414db4c1f4fa6172339
 runc:
  Version:          1.0.0-rc8+dev
  GitCommit:        3e425f80a8c931f88e6d94a8c831b9d5aa481657
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
[root@ubuntu1804 ~]#pstree -p
systemd(1)─┬─VGAuthService(796)
           ├─accounts-daemon(800)─┬─{accounts-daemon}(805)
           │                      └─{accounts-daemon}(807)
           ├─agetty(866)
           ├─atd(799)
           ├─blkmapd(502)
           ├─containerd(801)─┬─containerd-shim(2497)─┬─sh(2520)
           │                 │                       ├─{containerd-shim}(2499)
           │                 │                       ├─{containerd-shim}(2500)
           │                 │                       ├─{containerd-shim}(2501)
           │                 │                       ├─{containerd-shim}(2502)
           │                 │                       ├─{containerd-shim}(2503)
           │                 │                       ├─{containerd-shim}(2504)
           │                 │                       ├─{containerd-shim}(2505)
           │                 │                       ├─{containerd-shim}(2550)
           │                 │                       └─{containerd-shim}(2551)
           │                 ├─containerd-shim(2627)─┬─nginx(2649)───nginx(2699)
           │                 │                       ├─{containerd-shim}(2628)
           │                 │                       ├─{containerd-shim}(2629)
           │                 │                       ├─{containerd-shim}(2630)
           │                 │                       ├─{containerd-shim}(2631)
           │                 │                       ├─{containerd-shim}(2632)
           │                 │                       ├─{containerd-shim}(2633)
           │                 │                       ├─{containerd-shim}(2634)
           │                 │                       ├─{containerd-shim}(2636)
           │                 │                       └─{containerd-shim}(2678)
           │                 ├─containerd-shim(2756)─┬─nginx(2776)───nginx(2823)
           │                 │                       ├─{containerd-shim}(2757)
           │                 │                       ├─{containerd-shim}(2758)
           │                 │                       ├─{containerd-shim}(2759)
           │                 │                       ├─{containerd-shim}(2760)
           │                 │                       ├─{containerd-shim}(2761)
           │                 │                       ├─{containerd-shim}(2762)
           │                 │                       ├─{containerd-shim}(2763)
           │                 │                       ├─{containerd-shim}(2765)
```

```
|                 |                       └─{containerd-shim}(2805)
|                 ├─{containerd}(906)
|                 ├─{containerd}(907)
|                 ├─{containerd}(909)
|                 ├─{containerd}(930)
|                 ├─{containerd}(931)
|                 ├─{containerd}(933)
|                 ├─{containerd}(934)
|                 ├─{containerd}(948)
|                 ├─{containerd}(2498)
|                 └─{containerd}(2827)
├─cron(794)
├─dbus-daemon(781)
├─dockerd(2207)─┬─docker-proxy(2622)─┬─{docker-proxy}(2623)
|               |                    ├─{docker-proxy}(2624)
|               |                    ├─{docker-proxy}(2625)
|               |                    └─{docker-proxy}(2626)
|               ├─docker-proxy(2750)─┬─{docker-proxy}(2751)
|               |                    ├─{docker-proxy}(2752)
|               |                    ├─{docker-proxy}(2753)
|               |                    ├─{docker-proxy}(2754)
|               |                    └─{docker-proxy}(2755)
|               ├─{dockerd}(2209)
|               ├─{dockerd}(2210)
|               ├─{dockerd}(2211)
|               ├─{dockerd}(2215)
|               ├─{dockerd}(2220)
|               ├─{dockerd}(2221)
|               ├─{dockerd}(2222)
|               ├─{dockerd}(2514)
|               └─{dockerd}(2540)
├─iscsid(837)
├─iscsid(839)
├─lvmetad(513)
├─lxcfs(780)─┬─{lxcfs}(783)
|            ├─{lxcfs}(784)
|            └─{lxcfs}(1512)
├─networkd-dispat(795)───{networkd-dispat}(925)
├─polkitd(808)─┬─{polkitd}(809)
|              └─{polkitd}(816)
├─rpc.idmapd(690)
├─rpc.mountd(751)
├─rpcbind(693)
├─rsyslogd(788)─┬─{rsyslogd}(791)
|               ├─{rsyslogd}(792)
|               └─{rsyslogd}(793)
├─snapd(798)─┬─{snapd}(823)
|            ├─{snapd}(826)
|            ├─{snapd}(827)
|            ├─{snapd}(828)
|            ├─{snapd}(873)
|            ├─{snapd}(882)
|            └─{snapd}(883)
├─sshd(881)───sshd(912)───bash(1240)───pstree(2900)
├─systemd(947)───(sd-pam)(958)
├─systemd-journal(481)
├─systemd-logind(797)
├─systemd-network(700)
```

```
            ├─systemd-resolve(730)
            ├─systemd-timesyn(691)───{systemd-timesyn}(746)
            ├─systemd-udevd(508)
            └─vmtoolsd(489)

[root@ubuntu1804 ~]#ps aux|grep  -E "containerd|docker"
root        801  0.0  4.4 776680 43972 ?        Ssl  12:30   0:03
/usr/bin/containerd
root       2207  0.0  8.8 839016 86712 ?        Ssl  16:48   0:02
/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
root       2497  0.0  0.5 107696  5564 ?        Sl   17:33   0:00 containerd-
shim -namespace moby -workdir
/var/lib/containerd/io.containerd.runtime.v1.linux/moby/d4741f815199a35c7e802662
206160342d56125b47ec46d48a5f580759d86a6e -address
/run/containerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-
root /var/run/docker/runtime-runc
root       2622  0.0  0.4 405532  4128 ?        Sl   17:34   0:00
/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8888 -container-ip
172.17.0.3 -container-port 80
root       2627  0.0  0.6 109104  6196 ?        Sl   17:34   0:00 containerd-
shim -namespace moby -workdir
/var/lib/containerd/io.containerd.runtime.v1.linux/moby/5dee9be9afdbab8c2f6c4c5e
b0f956c9579efe93110daf638f8fd15f43d961e2 -address
/run/containerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-
root /var/run/docker/runtime-runc
root       2750  0.0  0.4 479264  4148 ?        Sl   17:38   0:00
/usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 6666 -container-ip
172.17.0.4 -container-port 80
root       2756  0.0  0.6 109104  6204 ?        Sl   17:38   0:00 containerd-
shim -namespace moby -workdir
/var/lib/containerd/io.containerd.runtime.v1.linux/moby/d9e7f75cdb9d7d30f8febae3
0d72cdc4b6e96b0408fa998af6deb3937d5271ed -address
/run/containerd/containerd.sock -containerd-binary /usr/bin/containerd -runtime-
root /var/run/docker/runtime-runc
root       2899  0.0  0.1  14428  1100 pts/0    S+   17:51   0:00 grep --
color=auto -E containerd|docker
```

18.06及之前的docker版本，进程关系:

1.2.5.5.2 docker的进程关系

docker 相关的四个进程:

- dockerd：服务器程序,被client直接访问，其父进程为宿主机的systemd守护进程。
- docker-proxy：每个进程docker-proxy实现对应一个需要网络通信的容器，管理宿主机和容器的之间端口映射，其父进程为dockerd，如果容器不需要网络则无需启动
- containerd：被dockerd进程调用以实现与runc交互

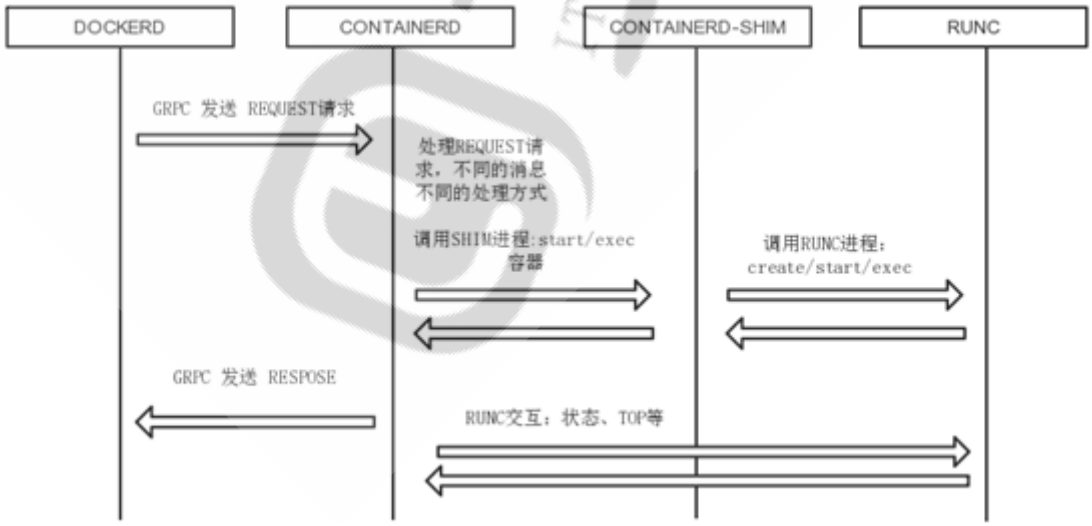
- containerd-shim：真正运行容器的载体，每个容器对应一个containerd-shim进程，其父进程为containerd

1.2.5.5.3 containerd-shim命令使用

```
[root@ubuntu1804 ~]#containerd-shim -h
Usage of containerd-shim:
  -address string
        grpc address back to main containerd
  -containerd-binary containerd publish
        path to containerd binary (used for containerd publish) (default
"containerd")
  -criu string
        path to criu binary
  -debug
        enable debug output in logs
  -namespace string
        namespace that owns the shim
  -runtime-root string
        root directory for the runtime (default "/run/containerd/runc")
  -socket string
        abstract socket path to serve
  -systemd-cgroup
        set runtime to use systemd-cgroup
  -workdir string
        path used to storge large temporary data
```

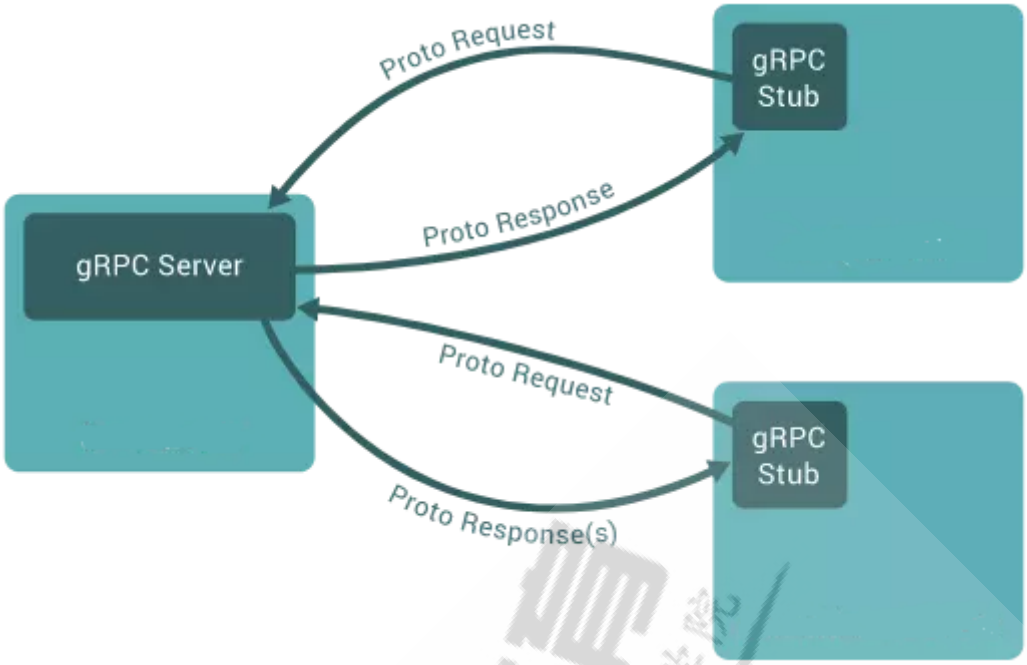
1.2.5.5.4 容器的创建与管理过程

通信流程:



1. dockerd通过grpc和 containerd模块通信，dockerd由libcontainerd负责和containerd进行交换，dockerd和containerd 通信socket文件：/run/containerd/containerd.sock
2. containerd在dockerd启动时被启动，然后containerd启动grpc请求监听，containerd处理grpc请求，根据请求做相应动作
3. 若是run, start或是exec 容器，containerd 拉起一个container-shim , 并进行相应的操作
4. container-shim别拉起后，start/exec/create拉起runC进程，通过exit、control文件和containerd通信，通过父子进程关系和SIGCHLD监控容器中进程状态
5. 在整个容器生命周期中，containerd通过 epoll 监控容器文件，监控容器事件

1.2.5.5.5 gRPC简介



gRPC是Google开发的一款高性能、开源和通用的 RPC 框架，支持众多语言客户端

官网: https://www.grpc.io/

1.2.5.5.6 podman 的进程结构

podman没有dockerd服务进程，所以当无容器启动时，无需启动任何进程，而容器启动时，会做为conmon的子进程

```
[root@centos8 ~]#podman version
Version:            1.4.2-stable2
RemoteAPI Version:  1
Go Version:         go1.12.8
OS/Arch:            linux/amd64

[root@centos8 ~]#podman run -d -p 80:80 docker.io/library/nginx
d8877293635c599a82ab5cb82c942cd86baf7c5810dd824154b15b0a88e76be8
[root@centos8 ~]#ss -tlnp
State    Recv-Q   Send-Q      Local Address:Port       Peer Address:Port
           LISTEN  0        128              0.0.0.0:80              0.0.0.0:*
      users:(("conmon",pid=5173,fd=5))
LISTEN  0        128               0.0.0.0:22              0.0.0.0:*
users:(("sshd",pid=687,fd=4))
LISTEN  0        128                  [::]:22                 [::]:*
users:(("sshd",pid=687,fd=6))

[root@centos8 ~]#pstree -p
systemd(1)─┬─NetworkManager(660)─┬─{NetworkManager}(680)
           |                     └─{NetworkManager}(682)
           ├─VGAuthService(663)
           ├─agetty(805)
           ├─anacron(2793)
           ├─atd(799)
           ├─auditd(616)───{auditd}(617)
           ├─automount(816)─┬─{automount}(821)
           |                ├─{automount}(822)
```

```
            |                ├─{automount}(829)
            |                └─{automount}(837)
            ├─conmon(5173)─┬─nginx(5183)───nginx(5194)
            |              └─{conmon}(5175)
            ├─crond(797)
            ├─dbus-daemon(658)
            ├─polkitd(665)─┬─{polkitd}(679)
            |              ├─{polkitd}(683)
            |              ├─{polkitd}(694)
            |              ├─{polkitd}(695)
            |              └─{polkitd}(750)
            ├─rngd(661)───{rngd}(673)
            ├─rsyslogd(814)─┬─{rsyslogd}(818)
            |               └─{rsyslogd}(820)
            ├─sshd(687)─┬─sshd(1166)───sshd(1243)───bash(1244)
            |           └─sshd(1306)───sshd(1308)───bash(1309)───pstree(5198)
            ├─sssd(659)─┬─sssd_be(722)
            |           └─sssd_nss(749)
            ├─systemd(1234)───(sd-pam)(1237)
            ├─systemd-journal(543)
            ├─systemd-logind(794)
            ├─systemd-udevd(575)
            ├─tuned(692)─┬─{tuned}(1080)
            |            ├─{tuned}(1089)
            |            └─{tuned}(1097)
            └─vmtoolsd(664)───{vmtoolsd}(762)
[root@centos8 ~]#
```

1.2.6 docker 服务管理

docker 服务基于C/S 结构,可以实现基于本地和远程方式进行管理

```
#Dockerd守护进程启动选项
-H  tcp://host:port
    unix:///path/to/socket,
    fd://* or fd://socketfd
#守护进程默认配置:
-H unix:///var/run/docker.sock

#使用Docker客户端命令选项
-H tcp://host:port
   unix:///path/to/socket,
   fd://* or fd://socketfd
客户端默认配置:
-H unix:///var/run/docker.sock

#docker客户端也可以使用环境变量DOCKER_ HOST,代替-H选项
export DOCKER_HOST="tcp://docker-server:2375"
```

范例: 通过UDS访问docker

```
[root@ubuntu1804 ~]#cat /lib/systemd/system/docker.service
[Unit]
Description=Docker Application Container Engine
```

```
Documentation=https://docs.docker.com
BindsTo=containerd.service
After=network-online.target firewalld.service containerd.service
Wants=network-online.target
Requires=docker.socket

[Service]
Type=notify
# the default is not to use systemd for cgroups because the delegate issues
still
# exists and systemd currently does not support the cgroup feature set required
# for containers run by docker
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
ExecReload=/bin/kill -s HUP $MAINPID
TimeoutSec=0
RestartSec=2
Restart=always

# Note that StartLimit* options were moved from "Service" to "Unit" in systemd
229.
# Both the old, and new location are accepted by systemd 229 and up, so using
the old location
# to make them work for either version of systemd.
StartLimitBurst=3

# Note that StartLimitInterval was renamed to StartLimitIntervalSec in systemd
230.
# Both the old, and new name are accepted by systemd 230 and up, so using the
old name to make
# this option work for either version of systemd.
StartLimitInterval=60s

# Having non-zero Limit*s causes performance problems due to accounting overhead
# in the kernel. We recommend using cgroups to do container-local accounting.
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity

# Comment TasksMax if your systemd version does not support it.
# Only systemd 226 and above support this option.
TasksMax=infinity

# set delegate yes so that systemd does not reset the cgroups of docker
containers
Delegate=yes

# kill only the docker process, not all processes in the cgroup
KillMode=process

[Install]
WantedBy=multi-user.target

[root@ubuntu1804 ~]#systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset:
enabled)
   Active: active (running) since Wed 2020-07-22 14:06:46 CST; 5h 50min ago
     Docs: https://docs.docker.com
```

```
 Main PID: 1138 (dockerd)
    Tasks: 17
   CGroup: /system.slice/docker.service
           └─1138 /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock

Jul 22 15:43:42 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T15:43:42.056506408+08:00" level=info msg="
Jul 22 15:43:42 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T15:43:42.064414577+08:00" level=info msg="
Jul 22 15:53:58 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T15:53:58.938037439+08:00" level=info msg="
Jul 22 15:53:58 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T15:53:58.951842078+08:00" level=info msg="
Jul 22 15:55:33 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T15:55:33.837166628+08:00" level=info msg="
Jul 22 19:47:55 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T19:47:55.212507176+08:00" level=info msg="
Jul 22 19:47:55 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T19:47:55.220542970+08:00" level=info msg="
Jul 22 19:47:55 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T19:47:55.234106123+08:00" level=info msg="
Jul 22 19:47:55 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T19:47:55.237476234+08:00" level=info msg="
Jul 22 19:47:55 ubuntu1804.magedu.org dockerd[1138]: time="2020-07-
22T19:47:55.238167375+08:00" level=info msg=

[root@ubuntu1804 ~]#ll /var/run/docker.sock
srw-rw---- 1 root docker 0 Jul 22 20:33 /var/run/docker.sock=

[root@ubuntu1804 ~]#nc -U /var/run/docker.sock
GET /info HTTP/1.1
host: www.magedu.org

HTTP/1.1 200 OK
Api-Version: 1.40
Content-Type: application/json
Docker-Experimental: false
Ostype: linux
Server: Docker/19.03.12 (linux)
Date: Wed, 22 Jul 2020 11:54:12 GMT
Transfer-Encoding: chunked

947
```

{"ID":"LVU6:OXD3:TAPB:KDNQ:YRSN:XTAS:3V32:IERB:2DM6:4CDK:CRO6:ZKAW","Containers":0,"ContainersRunning":0,"ContainersPaused":0,"ContainersStopped":0,"Images":5,"Driver":"overlay2","DriverStatus":[["Backing Filesystem","extfs"],["Supports d_type","true"],["Native Overlay Diff","true"]],"SystemStatus":null,"Plugins":{"Volume":["local"],"Network":["bridge","host","ipvlan","macvlan","null","overlay"],"Authorization":null,"Log":["awslogs","fluentd","gcplogs","gelf","journald","json-file","local","logentries","splunk","syslog"]},"MemoryLimit":true,"SwapLimit":false,"KernelMemory":true,"KernelMemoryTCP":true,"CpuCfsPeriod":true,"CpuCfsQuota":true,"CPUShares":true,"CPUSet":true,"PidsLimit":true,"IPv4Forwarding":true,"BridgeNfIptables":true,"BridgeNfIp6tables":true,"Debug":false,"NFd":21,"OomKillDisable":true,"NGoroutines":35,"SystemTime":"2020-07-22T19:54:12.335572334+08:00","LoggingDriver":"json-file","CgroupDriver":"cgroupfs","NEventsListener":0,"KernelVersion":"4.15.0-111-generic","OperatingSystem":"Ubuntu 18.04.4 LTS","OSType":"linux","Architecture":"x86_64","IndexServerAddress":"https://index.docker.io/v1/","RegistryConfig":{"AllowNondistributableArtifactsCIDRs":[],"AllowNondistributableArtifactsHostnames":[],"InsecureRegistryCIDRs":["127.0.0.0/8"],"IndexConfigs":{"docker.io":{"Name":"docker.io","Mirrors":["https://si7y70hh.mirror.aliyuncs.com/"],"Secure":true,"Official":true}},"Mirrors":["https://si7y70hh.mirror.aliyuncs.com/"]},"NCPU":4,"MemTotal":3122880512,"GenericResources":null,"DockerRootDir":"/var/lib/docker","HttpProxy":"","HttpsProxy":"","NoProxy":"","Name":"ubuntu1804.magedu.org","Labels":[],"ExperimentalBuild":false,"ServerVersion":"19.03.12","ClusterStore":"","ClusterAdvertise":"","Runtimes":{"runc":{"path":"runc"}},"DefaultRuntime":"runc","Swarm":{"NodeID":"","NodeAddr":"","LocalNodeState":"inactive","ControlAvailable":false,"Error":"","RemoteManagers":null},"LiveRestoreEnabled":false,"Isolation":"","InitBinary":"docker-init","ContainerdCommit":{"ID":"7ad184331fa3e55e52b890ea95e65ba581ae3429","Expected":"7ad184331fa3e55e52b890ea95e65ba581ae3429"},"RuncCommit":{"ID":"dc9208a3303feef5b3839f4323d9beb36df0a9dd","Expected":"dc9208a3303feef5b3839f4323d9beb36df0a9dd"},"InitCommit":{"ID":"fec3683","Expected":"fec3683"},"SecurityOptions":["name=apparmor","name=seccomp,profile=default"],"Warnings":["WARNING: No swap limit support"]}

范例: docker服务添加标签

```
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
#修改下面行
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --label="name=docker1"

[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#docker info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 5
```

```
 Server Version: 19.03.12
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  apparmor
  seccomp
   Profile: default
 Kernel Version: 4.15.0-111-generic
 Operating System: Ubuntu 18.04.4 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 4
 Total Memory: 2.908GiB
 Name: ubuntu1804.magedu.org
 ID: LVU6:OXD3:TAPB:KDNQ:YRSN:XTAS:3V32:IERB:2DM6:4CDK:CRO6:ZKAW
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
  name=docker1   #此处显示添加的标签
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/
 Live Restore Enabled: false

WARNING: No swap limit support
```

范例: 开启docker的远程访问

```
#方法1
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
#修改下面行
ExecStart=/usr/bin/dockerd -H fd:// -H tcp://0.0.0.0:2375 --
containerd=/run/containerd/containerd.sock --label="name=docker1"

#方法2
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd  --containerd=/run/containerd/containerd.sock
```

```
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
{
    "hosts": ["tcp://0.0.0.0:2375", "fd://"]
}

[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#ss -tnlp|grep 2375
LISTEN   0         128                       *:2375                   *:*
users:(("dockerd",pid=9964,fd=3))
[root@ubuntu1804 ~]#ps -ef | grep docker
root       9964      1  0 20:33 ?        00:00:00 /usr/bin/dockerd -H fd:// -H
tcp://0.0.0.0:2375 --containerd=/run/containerd/containerd.sock --
label=name=docker1
root      10187   2854  0 20:37 pts/1    00:00:00 grep --color=auto docker
[root@ubuntu1804 ~]#ll /var/run/docker.sock
srw-rw---- 1 root docker 0 Jul 22 20:33 /var/run/docker.sock=

#实现远程访问方式1
[root@centos7 ~]#curl http://10.0.0.100:2375/info
```

{"ID":"LVU6:OXD3:TAPB:KDNQ:YRSN:XTAS:3V32:IERB:2DM6:4CDK:CRO6:ZKAW","Containers":0,"ContainersRunning":0,"ContainersPaused":0,"ContainersStopped":0,"Images":5,"Driver":"overlay2","DriverStatus":[["Backing Filesystem","extfs"],["Supports d_type","true"],["Native Overlay Diff","true"]],"SystemStatus":null,"Plugins":{"Volume":["local"],"Network":["bridge","host","ipvlan","macvlan","null","overlay"],"Authorization":null,"Log":["awslogs","fluentd","gcplogs","gelf","journald","json-file","local","logentries","splunk","syslog"]},"MemoryLimit":true,"SwapLimit":false,"KernelMemory":true,"KernelMemoryTCP":true,"CpuCfsPeriod":true,"CpuCfsQuota":true,"CPUShares":true,"CPUSet":true,"PidsLimit":true,"IPv4Forwarding":true,"BridgeNfIptables":true,"BridgeNfIp6tables":true,"Debug":false,"NFd":22,"OomKillDisable":true,"NGoroutines":35,"SystemTime":"2020-07-22T20:54:42.419355793+08:00","LoggingDriver":"json-file","CgroupDriver":"cgroupfs","NEventsListener":0,"KernelVersion":"4.15.0-111-generic","OperatingSystem":"Ubuntu 18.04.4 LTS","OSType":"linux","Architecture":"x86_64","IndexServerAddress":"https://index.docker.io/v1/","RegistryConfig":{"AllowNondistributableArtifactsCIDRs":[],"AllowNondistributableArtifactsHostnames":[],"InsecureRegistryCIDRs":["127.0.0.0/8"],"IndexConfigs":{"docker.io":{"Name":"docker.io","Mirrors":["https://si7y70hh.mirror.aliyuncs.com/"],"Secure":true,"Official":true}},"Mirrors":["https://si7y70hh.mirror.aliyuncs.com/"]},"NCPU":4,"MemTotal":3122880512,"GenericResources":null,"DockerRootDir":"/var/lib/docker","HttpProxy":"","HttpsProxy":"","NoProxy":"","Name":"ubuntu1804.magedu.org","Labels":["name=docker1"],"ExperimentalBuild":false,"ServerVersion":"19.03.12","ClusterStore":"","ClusterAdvertise":"","Runtimes":{"runc":{"path":"runc"}},"DefaultRuntime":"runc","Swarm":{"NodeID":"","NodeAddr":"","LocalNodeState":"inactive","ControlAvailable":false,"Error":"","RemoteManagers":null},"LiveRestoreEnabled":false,"Isolation":"","InitBinary":"docker-init","ContainerdCommit":{"ID":"7ad184331fa3e55e52b890ea95e65ba581ae3429","Expected":"7ad184331fa3e55e52b890ea95e65ba581ae3429"},"RuncCommit":{"ID":"dc9208a3303feef5b3839f4323d9beb36df0a9dd","Expected":"dc9208a3303feef5b3839f4323d9beb36df0a9dd"},"InitCommit":{"ID":"fec3683","Expected":"fec3683"},"SecurityOptions":["name=apparmor","name=seccomp,profile=default"],"Warnings":["WARNING: API is accessible on http://0.0.0.0:2375 without encryption.\n   Access to the remote API is equivalent to root access on the host. Refer\n   to the 'Docker daemon attack surface' section in the documentation for\n   more information: https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface","WARNING: No swap limit support"]}

#实现远程访问方式2

```
[root@centos7 ~]#docker -H tcp://10.0.0.100:2375 info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 5
 Server Version: 19.03.12
 Storage Driver: overlay2
```

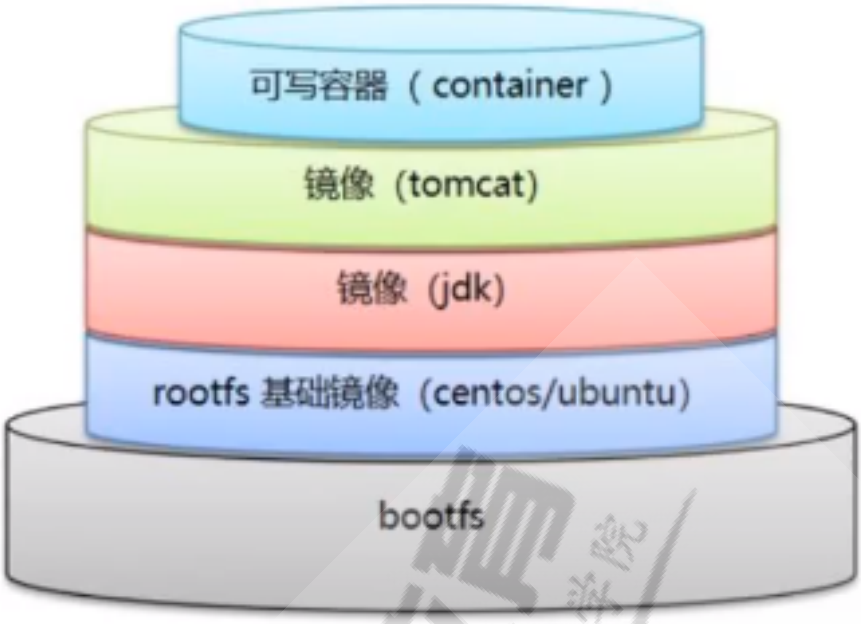
```
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  apparmor
  seccomp
   Profile: default
 Kernel Version: 4.15.0-111-generic
 Operating System: Ubuntu 18.04.4 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 4
 Total Memory: 2.908GiB
 Name: ubuntu1804.magedu.org
 ID: LVU6:OXD3:TAPB:KDNQ:YRSN:XTAS:3V32:IERB:2DM6:4CDK:CRO6:ZKAW
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
  name=docker1
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/
 Live Restore Enabled: false

WARNING: API is accessible on http://0.0.0.0:2375 without encryption.
         Access to the remote API is equivalent to root access on the host.
Refer
         to the 'Docker daemon attack surface' section in the documentation for
         more information:
https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface
WARNING: No swap limit support
[root@centos7 ~]#


#实现远程访问方式3
[root@centos7 ~]#export DOCKER_HOST="tcp://10.0.0.100:2375"
[root@centos7 ~]#docker info
Client:
 Debug Mode: false

Server:
```

```
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 5
 Server Version: 19.03.12
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  apparmor
  seccomp
   Profile: default
 Kernel Version: 4.15.0-111-generic
 Operating System: Ubuntu 18.04.4 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 4
 Total Memory: 2.908GiB
 Name: ubuntu1804.magedu.org
 ID: LVU6:OXD3:TAPB:KDNQ:YRSN:XTAS:3V32:IERB:2DM6:4CDK:CRO6:ZKAW
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
  name=docker1
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/
 Live Restore Enabled: false

WARNING: API is accessible on http://0.0.0.0:2375 without encryption.
         Access to the remote API is equivalent to root access on the host.
Refer
         to the 'Docker daemon attack surface' section in the documentation for
         more information:
https://docs.docker.com/engine/security/security/#docker-daemon-attack-surface
WARNING: No swap limit support

#恢复连接本机
[root@centos7 ~]#unset DOCKER_HOST
```

```
[root@centos7 ~]#docker info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 0
 Server Version: 19.03.5
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: 7ad184331fa3e55e52b890ea95e65ba581ae3429
 runc version: dc9208a3303feef5b3839f4323d9beb36df0a9dd
 init version: fec3683
 Security Options:
  seccomp
   Profile: default
 Kernel Version: 3.10.0-1127.el7.x86_64
 Operating System: CentOS Linux 7 (Core)
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 972.3MiB
 Name: centos7.wangxiaochun.com
 ID: USO2:CGRA:LIV3:SWOQ:5AWX:EN6W:4AUZ:XYZ7:LL6K:SUQ5:HANV:TX5L
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/
 Live Restore Enabled: false
```

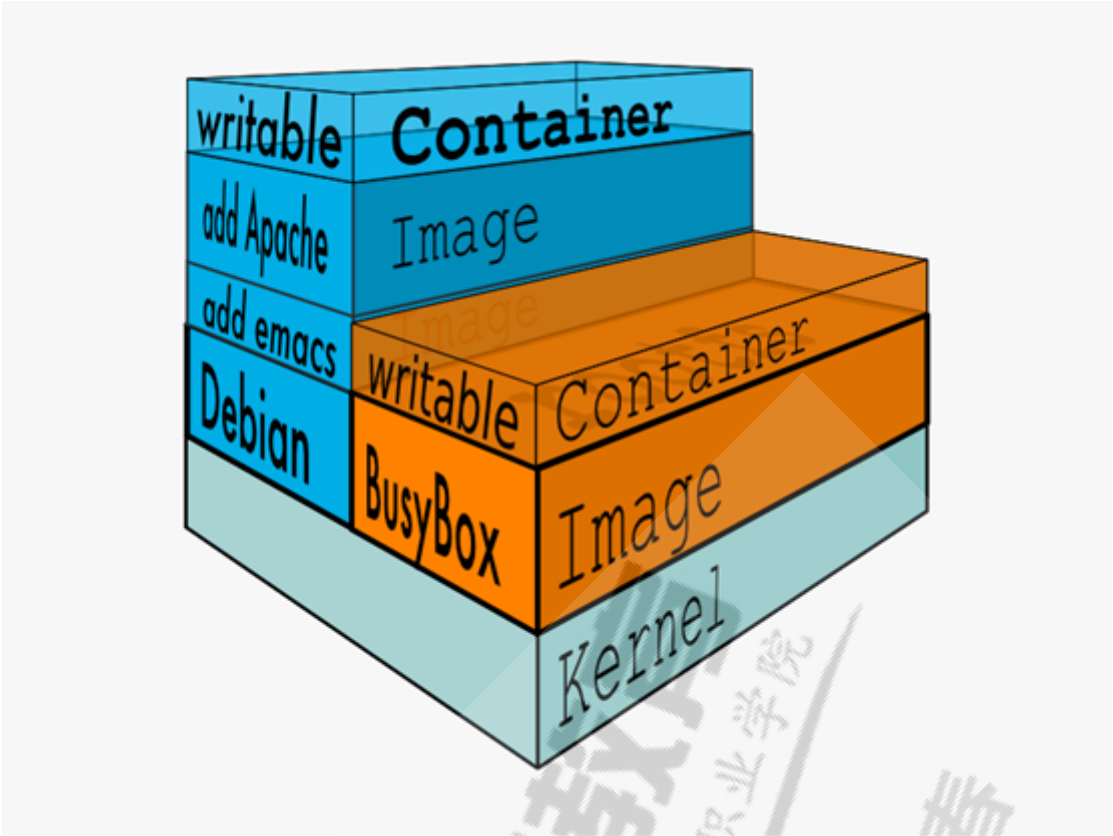
1.3 镜像管理

1.3.1 镜像结构和原理



镜像即创建容器的模版，含有启动容器所需要的文件系统及所需要的内容，因此镜像主要用于方便和快速的创建并启动容器

镜像含里面是一层层的文件系统,叫做 Union FS（联合文件系统）,联合文件系统，可以将几层目录挂载到一起（就像千层饼，洋葱头，俄罗斯套娃一样），形成一个虚拟文件系统,虚拟文件系统的目录结构就像普通 linux 的目录结构一样，镜像通过这些文件再加上宿主机的内核共同提供了一个 linux 的虚拟环境，每一层文件系统叫做一层 layer，联合文件系统可以对每一层文件系统设置三种权限，只读（readonly）、读写（readwrite）和写出（whiteout-able），但是镜像中每一层文件系统都是只读的,构建镜像的时候，从一个最基本的操作系统开始，每个构建提交的操作都相当于做一层的修改，增加了一层文件系统，一层层往上叠加，上层的修改会覆盖底层该位置的可见性，这也很容易理解，就像上层把底层遮住了一样，当使用镜像的时候，我们只会看到一个完全的整体，不知道里面有几层,实际上也不需要知道里面有几层，结构如下:



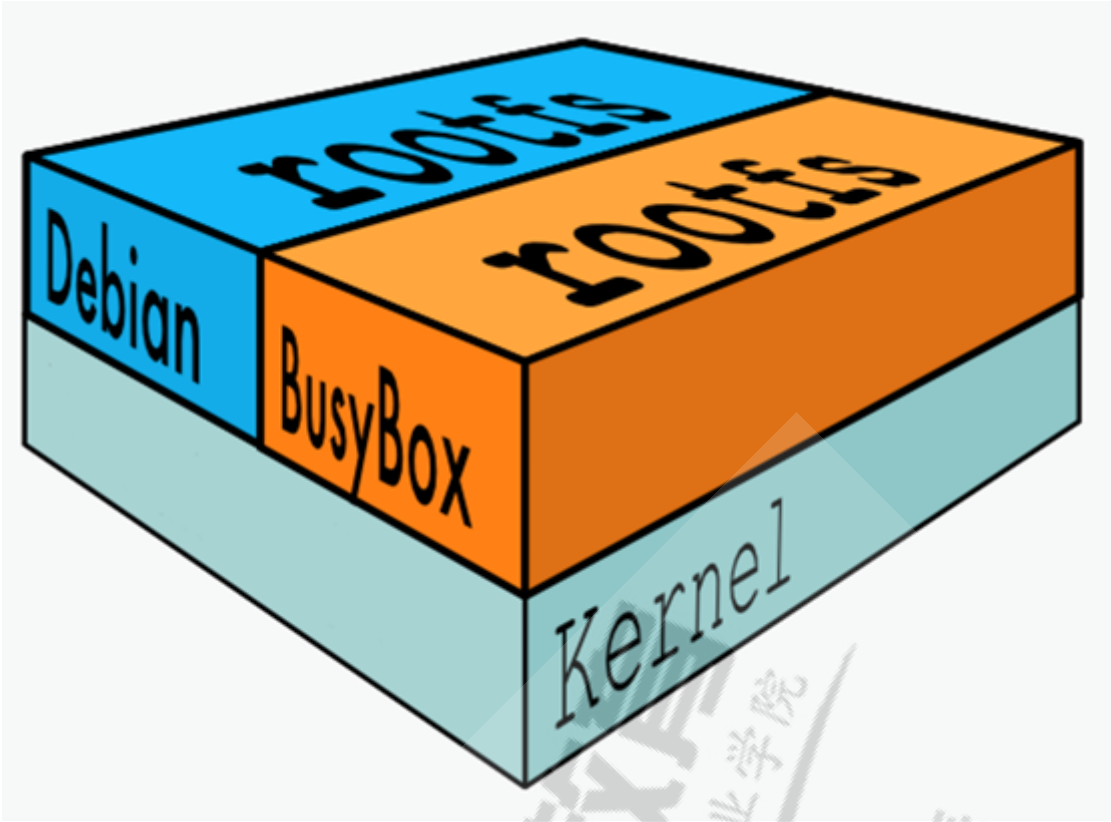
一个典型的 Linux文件系统由 bootfs 和 rootfs 两部分组成

bootfs(boot file system) 主要包含bootloader和kernel，bootloader主要用于引导加载 kernel，Linux 刚启动时会加载bootfs文件系统,当boot加载完成后,kernel 被加载到内存中后接管系统的控制权,bootfs 会被 umount 掉

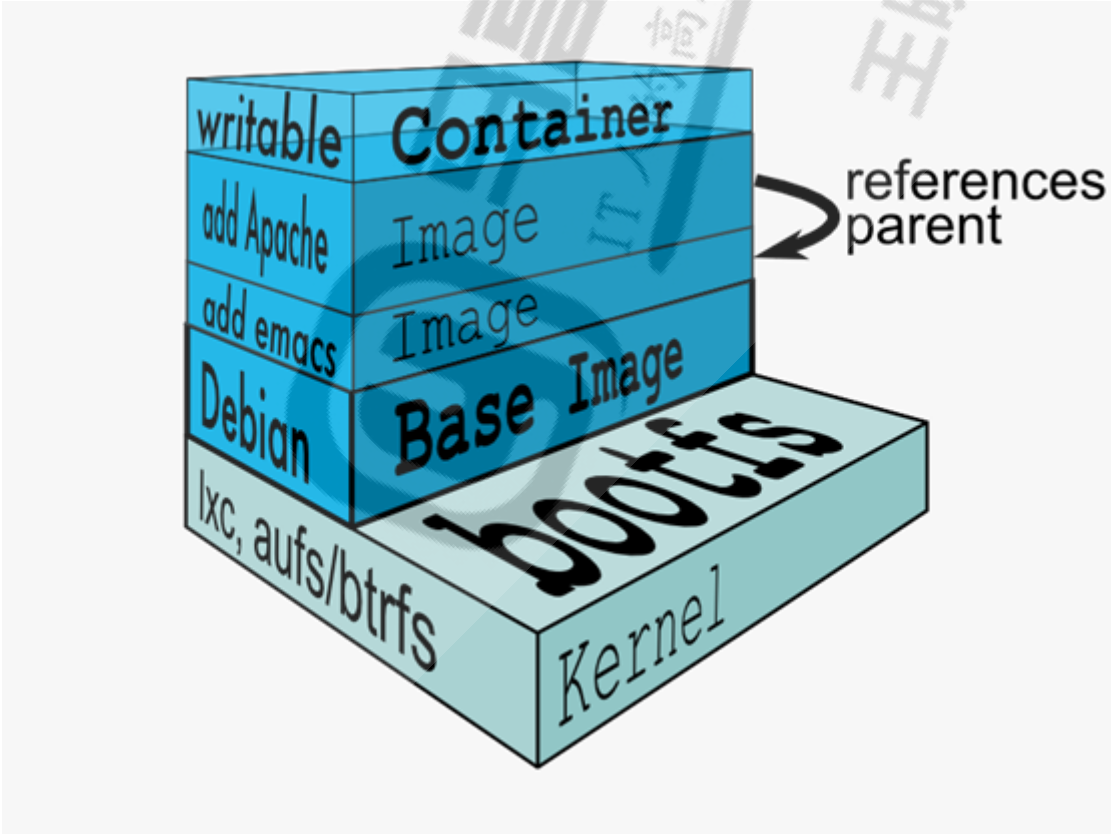
rootfs (root file system) 包含的就是典型 Linux 系统中的/dev，/proc，/bin，/etc 等标准目录和文件，不同的 linux 发行版（如 ubuntu 和 CentOS) 主要在 rootfs 这一层会有所区别。

一般的镜像通常都比较小，官方提供的Ubuntu镜像只有60MB多点，而 CentOS 基础镜像也只有200MB左右，一些其他版本的镜像甚至只有几MB，比如：busybox 才1.22MB，alpine镜像也只有5M左右。镜像直接调用宿主机的内核，镜像中只提供 rootfs，也就是只需要包括最基本的命令,配置文件和程序库等相关文件就可以了。

下图就是有两个不同的镜像在一个宿主机内核上实现不同的rootfs。



容器、镜像和父镜像关系:



范例: 查看镜像的分层结构

```
[root@ubuntu1804 ~]#docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
8ec398bc0356: Pull complete
a53c868fbde7: Pull complete
```

```
79daf9dd140d: Pull complete
Digest: sha256:70821e443be75ea38bdf52a974fd2271babd5875b2b1964f05025981c75a6717
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest

#查看镜像分层历史
[root@ubuntu1804 ~]#docker image  history nginx
IMAGE               CREATED             CREATED BY
       SIZE                COMMENT
0901fa9da894        9 days ago          /bin/sh -c #(nop)  CMD ["nginx" "-g"
"daemon…   0B
<missing>           9 days ago          /bin/sh -c #(nop)  STOPSIGNAL SIGTERM
        0B
<missing>           9 days ago          /bin/sh -c #(nop)  EXPOSE 80
       0B
<missing>           9 days ago          /bin/sh -c #(nop)  ENTRYPOINT ["/docker-
entr…   0B
<missing>           9 days ago          /bin/sh -c #(nop) COPY
file:0fd5fca330dcd6a7…   1.04kB
<missing>           9 days ago          /bin/sh -c #(nop) COPY
file:1d0a4127e78a26c1…   1.96kB
<missing>           9 days ago          /bin/sh -c #(nop) COPY
file:e7e183879c35719c…   1.2kB
<missing>           9 days ago          /bin/sh -c set -x     && addgroup --
system -…   63.3MB
<missing>           9 days ago          /bin/sh -c #(nop)  ENV
PKG_RELEASE=1~buster     0B
<missing>           9 days ago          /bin/sh -c #(nop)  ENV NJS_VERSION=0.4.2
       0B
<missing>           9 days ago          /bin/sh -c #(nop)  ENV
NGINX_VERSION=1.19.1     0B
<missing>           5 weeks ago         /bin/sh -c #(nop)  LABEL
maintainer=NGINX Do…   0B
<missing>           5 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]
        0B
<missing>           5 weeks ago         /bin/sh -c #(nop) ADD
file:4d35f6c8bbbe6801c…   69.2MB

[root@ubuntu1804 ~]#docker inspect nginx
[
    {
        "Id":
"sha256:0901fa9da894a8e9de5cb26d6749eaffb67b373dc1ff8a26c46b23b1175c913a",
        "RepoTags": [
            "nginx:latest"
        ],
        "RepoDigests": [

 "nginx@sha256:a93c8a0b0974c967aebe868a186e5c205f4d3bcb5423a56559f2f9599074bbcd"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-07-10T20:26:44.624785651Z",
        "Container":
"348c3ade7f4bdc0366f3f390ea4cfaebfb355ad7d621547eaf73728136d3bd2d",
        "ContainerConfig": {
            "Hostname": "348c3ade7f4b",
            "Domainname": "",
```

```
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.19.1",
                "NJS_VERSION=0.4.2",
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "#(nop) ",
                "CMD [\"nginx\" \"-g\" \"daemon off;\"]"
            ],
            "ArgsEscaped": true,
            "Image":
"sha256:8a6dfc8c21a1b3f3679b7755fc7869a22b5f8583778cf7835b5ee5387a73ae5e",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": [
                "/docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers <docker-
maint@nginx.com>"
            },
            "StopSignal": "SIGTERM"
        },
        "DockerVersion": "18.09.7",
        "Author": "",
        "Config": {
            "Hostname": "",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "ExposedPorts": {
                "80/tcp": {}
            },
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
                "NGINX_VERSION=1.19.1",
                "NJS_VERSION=0.4.2",
```

```
                "PKG_RELEASE=1~buster"
            ],
            "Cmd": [
                "nginx",
                "-g",
                "daemon off;"
            ],
            "ArgsEscaped": true,
            "Image":
"sha256:8a6dfc8c21a1b3f3679b7755fc7869a22b5f8583778cf7835b5ee5387a73ae5e",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": [
                "/docker-entrypoint.sh"
            ],
            "OnBuild": null,
            "Labels": {
                "maintainer": "NGINX Docker Maintainers <docker-
maint@nginx.com>"
            },
            "StopSignal": "SIGTERM"
        },
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 132484492,
        "VirtualSize": 132484492,
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/87dfa2bbff4f392e64e8a2fce11e2d9b8c3fffcfd51c6721ef0103
f7d6b525aa/diff:/var/lib/docker/overlay2/925c1a9c01939d111b3f0576608ad02a09bceea
fb6cad8dd616c24a59151bb25/diff:/var/lib/docker/overlay2/b9fd33e18d7bb7bb29b51ee3
99dfe3f21654fa4a9a086e02dcbc23f34f140c09/diff:/var/lib/docker/overlay2/35c6b8c39
68cdc21b78d4bcd6c192683b47db788db612dee04c5110037eed7af/diff",
                "MergedDir":
"/var/lib/docker/overlay2/23d3c6ecee136afe7137528b0e6997a488b1f277e86c794fa9a70b
5638a5d3f9/merged",
                "UpperDir":
"/var/lib/docker/overlay2/23d3c6ecee136afe7137528b0e6997a488b1f277e86c794fa9a70b
5638a5d3f9/diff",
                "WorkDir":
"/var/lib/docker/overlay2/23d3c6ecee136afe7137528b0e6997a488b1f277e86c794fa9a70b
5638a5d3f9/work"
            },
            "Name": "overlay2"
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [

 "sha256:13cb14c2acd34e45446a50af25cb05095a17624678dbafbcc9e26086547c1d74",

 "sha256:0e32546a8af0cd04ad451d6a9d22e650e500e5da3636a32648c9f5aca96a0ff7",

 "sha256:7ef35766ef7d5d3d958022405b308d5c105b41190e1b63b2037c4055c6950c9e",

 "sha256:4856db5e4f59384c413c20c46cd5403a860e1b07c8fdbad24df1ffd9209d44e7",
```

```
 "sha256:2808ec4a8ea71c2660284d06cf7e25354b70b58504edb46ac3e705fb7e6ea519"
            ]
        },
        "Metadata": {
            "LastTagTime": "0001-01-01T00:00:00Z"
        }
    }
]


[root@ubuntu1804 ~]#docker save nginx -o nginx.tar
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx               latest              0901fa9da894        3 days ago
127MB
alpine              3.11.3              e7d92cdc71fe        7 days ago
 5.59MB
centos              centos8.1.1911      470671670cac        7 days ago
237MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
hello-world         latest              fce289e99eb9        12 months ago
1.84kB

[root@ubuntu1804 ~]#ll -h nginx.tar
-rw------- 1 root root 131M Jul 20 22:33 nginx.tar

[root@ubuntu1804 ~]#tar xf nginx.tar  -C /data
[root@ubuntu1804 ~]#ll /data
total 60
drwxr-xr-x  8 root root  4096 Jul 20 22:34 ./
drwxr-xr-x 24 root root  4096 Jul 20 16:23 ../
-rw-r--r--  1 root root  7510 Jul 11 04:26
0901fa9da894a8e9de5cb26d6749eaffb67b373dc1ff8a26c46b23b1175c913a.json
drwxr-xr-x  2 root root  4096 Jul 11 04:26
0bb74fcd4b686412f7993916e58c26abd155fa10b10a4dc09a778e7c324c39a2/
drwxr-xr-x  2 root root  4096 Jul 11 04:26
517e3239147277447b60191907bc66168963e0ce8707a6a33532f7c63a0d2f12/
drwxr-xr-x  2 root root  4096 Jul 11 04:26
68c9e9da52d5a57ee196829ce4a461cc9425b0b920689da9ad547f1da13dbc9d/
drwxr-xr-x  2 root root  4096 Jul 11 04:26
d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd/
drwxr-xr-x  2 root root  4096 Jul 11 04:26
f4bf863ecdbb8bddb4b3bb271bdd97b067dcb6c95c56f720018abec6af190c6e/
drwx------  2 root root 16384 Mar 18 09:49 lost+found/
-rw-r--r--  1 root root   509 Jan  1  1970 manifest.json
-rw-r--r--  1 root root    88 Jan  1  1970 repositories


[root@ubuntu1804 ~]#cat /data/manifest.json
```

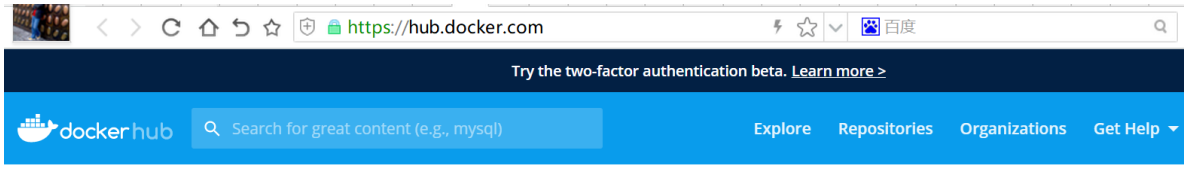
```
[{"Config":"0901fa9da894a8e9de5cb26d6749eaffb67b373dc1ff8a26c46b23b1175c913a.jso
n","RepoTags":["nginx:latest"],"Layers":
["d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd/layer.tar","f
4bf863ecdbb8bddb4b3bb271bdd97b067dcb6c95c56f720018abec6af190c6e/layer.tar","517e
3239147277447b60191907bc66168963e0ce8707a6a33532f7c63a0d2f12/layer.tar","0bb74fc
d4b686412f7993916e58c26abd155fa10b10a4dc09a778e7c324c39a2/layer.tar","68c9e9da52
d5a57ee196829ce4a461cc9425b0b920689da9ad547f1da13dbc9d/layer.tar"]}]

[root@ubuntu1804 ~]#du -sh /data/*
8.0K
/data/0901fa9da894a8e9de5cb26d6749eaffb67b373dc1ff8a26c46b23b1175c913a.json
16K /data/0bb74fcd4b686412f7993916e58c26abd155fa10b10a4dc09a778e7c324c39a2
16K /data/517e3239147277447b60191907bc66168963e0ce8707a6a33532f7c63a0d2f12
16K /data/68c9e9da52d5a57ee196829ce4a461cc9425b0b920689da9ad547f1da13dbc9d
70M /data/d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd
62M /data/f4bf863ecdbb8bddb4b3bb271bdd97b067dcb6c95c56f720018abec6af190c6e
16K /data/lost+found
4.0K    /data/manifest.json
4.0K    /data/repositories

[root@ubuntu1804 ~]#cd
/data/d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd/
[root@ubuntu1804
d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd]#ls
json  layer.tar  VERSION
[root@ubuntu1804
d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd]#tar xf
layer.tar
[root@ubuntu1804
d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd]#ls
bin   dev  home  layer.tar  lib64  mnt  proc  run   srv  tmp  var
boot  etc  json  lib        media  opt  root  sbin  sys  usr  VERSION
[root@ubuntu1804
d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd]#cat etc/i
init.d/    issue      issue.net
[root@ubuntu1804
d2cf0fc540bb3be33ee7340498c41fd4fc82c6bb02b9955fca2109e599301dbd]#cat etc/issue
Debian GNU/Linux 10 \n \l
```

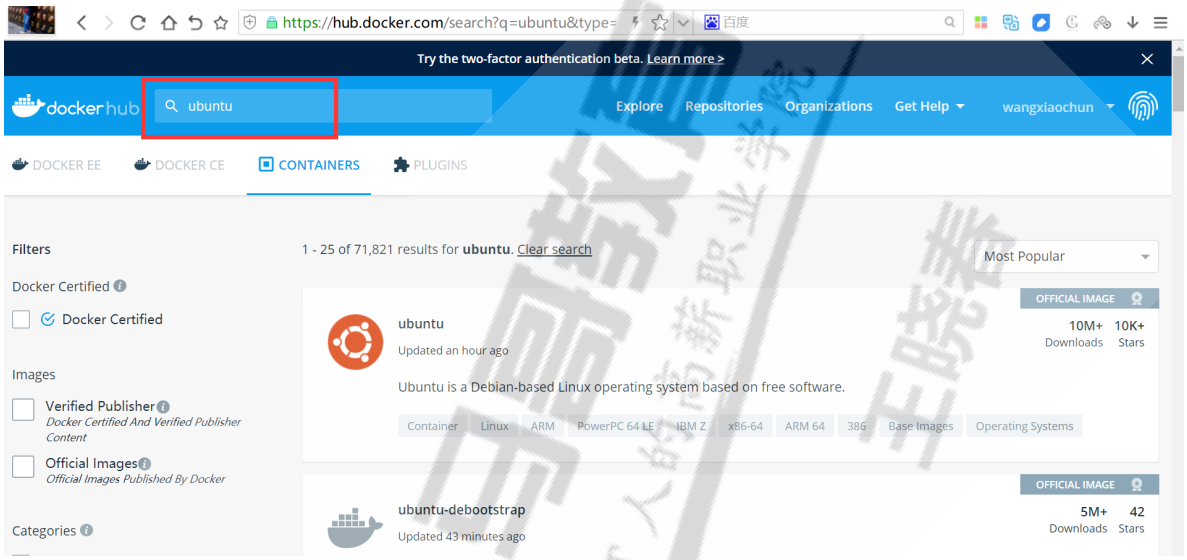
1.3.2 搜索镜像

1.3.2.1 搜索镜像

1.3.2.1.1 官方网站进行镜像的搜索

官网: http://hub.docker.com





在官方的docker 仓库中搜索指定名称的docker镜像，也会有很多三方镜像。

1.3.2.2.2 执行docker search命令进行搜索

格式如下:

```
Usage:  docker search [OPTIONS] TERM

Options:
  -f, --filter filter   Filter output based on conditions provided
      --format string   Pretty-print search using a Go template
      --limit int       Max number of search results (default 25)
      --no-trunc        Don't truncate output
```

说明:
OFFICIAL: 官方
AUTOMATED: 使用第三方docker服务来帮助编译镜像，可以在互联网上面直接拉取到镜像，减少了繁琐的编译过程

范例:

```
[root@ubuntu1804 ~]#docker search centos
```

```
NAME                                DESCRIPTION
  STARS               OFFICIAL            AUTOMATED
centos                              The official build of CentOS.
  5786                [OK]
ansible/centos7-ansible             Ansible on Centos7
   126                                 [OK]
jdeathe/centos-ssh                  OpenSSH / Supervisor / EPEL/IUS/SCL Repos - …
  114                                 [OK]
consol/centos-xfce-vnc              Centos container with "headless" VNC session…
  108                                 [OK]
centos/mysql-57-centos7             MySQL 5.7 SQL database server
  67
imagine10255/centos6-lnmp-php56     centos6-lnmp-php56
   57                                 [OK]
tutum/centos                        Simple CentOS docker image with SSH access
   44
centos/postgresql-96-centos7        PostgreSQL is an advanced Object-Relational …
  40
centos/httpd-24-centos7             Platform for running Apache httpd 2.4 or bui…
  29
kinogmt/centos-ssh                  CentOS with SSH
  29                                  [OK]
pivotaldata/centos-gpdb-dev         CentOS image for GPDB development. Tag names…
  10
drecom/centos-ruby                  centos ruby
  6                                   [OK]
centos/tools                        Docker image that has systems administration…
  5                                   [OK]
mamohr/centos-java                  Oracle Java 8 Docker image based on Centos 7
   3                                   [OK]
darksheer/centos                    Base Centos Image -- Updated hourly
  3                                   [OK]
pivotaldata/centos                  Base centos, freshened up a little with a Do…
  3
pivotaldata/centos-mingw            Using the mingw toolchain to cross-compile t…
  2
miko2u/centos6                      CentOS6 日本??境
    2                                  [OK]
pivotaldata/centos-gcc-toolchain    CentOS with a toolchain, but unaffiliated wi…
  2
indigo/centos-maven                 Vanilla CentOS 7 with Oracle Java Developmen…
  1                                   [OK]
mcnaughton/centos-base              centos base image
  1                                   [OK]
blacklabelops/centos                CentOS Base Image! Built and Updates Daily!
  1                                   [OK]
pivotaldata/centos6.8-dev           CentosOS 6.8 image for GPDB development
  0
pivotaldata/centos7-dev             CentosOS 7 image for GPDB development
  0
smartentry/centos                   centos with smartentry
   0                                  [OK]
[root@ubuntu1804 ~]#
```

范例：选择性的查找镜像

```
#搜索点赞100个以上的镜像
#旧语法
[root@ubuntu1804 ~]#docker search -s 100  centos
Flag --stars has been deprecated, use --filter=stars=3 instead
......

#新语法
[root@ubuntu1804 ~]#docker  search --filter=stars=100 centos
NAME                      DESCRIPTION                                     STARS
              OFFICIAL            AUTOMATED
centos                    The official build of CentOS.                   6096
             [OK]
ansible/centos7-ansible   Ansible on Centos7                              132
                                  [OK]
consol/centos-xfce-vnc    Centos container with "headless" VNC session…   117
                                  [OK]
jdeathe/centos-ssh        OpenSSH / Supervisor / EPEL/IUS/SCL Repos - …   115
                                  [OK]
```

1.3.2.2 alpine 介绍

Alpine 操作系统是一个面向安全的轻型 Linux 发行版。它不同于通常 Linux 发行版，Alpine 采用了 musl libc 和 busybox 以减小系统的体积和运行时资源消耗，但功能上比 busybox 又完善的多，因此得到开源社区越来越多的青睐。在保持瘦身的同时，Alpine 还提供了自己的包管理工具 apk，可以通过 https://pkgs.alpinelinux.org/packages 网站上查询包信息，也可以直接通过 apk 命令直接查询和安装各种软件。

Alpine 由非商业组织维护的，支持广泛场景的 Linux发行版，它特别为资深/重度Linux用户而优化，关注安全，性能和资源效能。Alpine 镜像可以适用于更多常用场景，并且是一个优秀的可以适用于生产的基础系统/环境。

Alpine Docker 镜像也继承了 Alpine Linux 发行版的这些优势。相比于其他 Docker 镜像，它的容量非常小，仅仅只有 5 MB 左右（对比 Ubuntu 系列镜像接近 200 MB），且拥有非常友好的包管理机制。官方镜像来自 docker-alpine 项目。

目前 Docker 官方已开始推荐使用 Alpine 替代之前的 Ubuntu 做为基础镜像环境。这样会带来多个好处。包括镜像下载速度加快，镜像安全性提高，主机之间的切换更方便，占用更少磁盘空间等。

下表是官方镜像的大小比较:

```
REPOSITORY          TAG                 IMAGE ID            VIRTUAL SIZE
alpine              latest              4e38e38c8ce0        4.799 MB
debian              latest              4d6ce913b130        84.98 MB
ubuntu              latest              b39b81afc8ca        188.3 MB
centos              latest              8efe422e6104        210 MB
```

- `Alpine` 官网: https://www.alpinelinux.org/
- `Alpine` 官方仓库: https://github.com/alpinelinux
- `Alpine` 官方镜像: https://hub.docker.com/_/alpine/
- `Alpine` 官方镜像仓库: https://github.com/gliderlabs/docker-alpine
- `Alpine` 阿里云的镜像仓库: https://mirrors.aliyun.com/alpine/

范例: alpine管理软件

```
#修改源替换成阿里源，将里面 dl-cdn.alpinelinux.org 的 改成 mirrors.aliyun.com
vi /etc/apk/repositories
http://mirrors.aliyun.com/alpine/v3.8/main/
http://mirrors.aliyun.com/alpine/v3.8/community/

#更新源
apk update

#安装软件
apk add vim

#删除软件
apk del openssh openntp vim
```

范例:

```
/ # apk --help
apk-tools 2.10.4, compiled for x86_64.

Installing and removing packages:
  add       Add PACKAGEs to 'world' and install (or upgrade) them, while
ensuring that all dependencies are met
  del       Remove PACKAGEs from 'world' and uninstall them

System maintenance:
  fix       Repair package or upgrade it without modifying main dependencies
  update    Update repository indexes from all remote repositories
  upgrade   Upgrade currently installed packages to match repositories
  cache     Download missing PACKAGEs to cache and/or delete unneeded files from
cache

Querying information about packages:
  info      Give detailed information about PACKAGEs or repositories
  list      List packages by PATTERN and other criteria
  dot       Generate graphviz graphs
  policy    Show repository policy for packages

Repository maintenance:
  index     Create repository index file from FILEs
  fetch     Download PACKAGEs from global repositories to a local directory
  verify    Verify package integrity and signature
  manifest  Show checksums of package contents

Use apk <command> --help for command-specific help.
Use apk --help --verbose for a full command listing.

This apk has coffee making abilities.
```

```
/ # apk add  nginx
/ # apk info nginx
nginx-1.16.1-r6 description:
HTTP and reverse proxy server (stable version)

nginx-1.16.1-r6 webpage:
https://www.nginx.org/

nginx-1.16.1-r6 installed size:
1126400

~ # apk manifest nginx
sha1:d21a96358a10b731f8847e6d32799efdc2a7f421  etc/logrotate.d/nginx
sha1:50bd6d3b4f3e6b577d852f12cd6939719d2c2db5  etc/init.d/nginx
sha1:379c1e2a2a5ffb8c91a07328d4c9be2bc58799fd  etc/nginx/scgi_params
sha1:da38e2a0dded838afbe0eade6cb837ac30fd8046  etc/nginx/fastcgi_params
sha1:cc2fcdb4605dcac23d59f667889ccbdfdc6e3668  etc/nginx/uwsgi_params
sha1:cbf596ddb3433a8e0d325f3c188bec9c1bb746b3  etc/nginx/fastcgi.conf
sha1:e39dbc36680b717ec902fadc805a302f1cf62245  etc/nginx/mime.types
sha1:e9dddf20f1196bb67eef28107438b60c4060f4d3  etc/nginx/nginx.conf
sha1:7b2a4da1a14166442c10cbf9e357fa9fb53542ca  etc/nginx/conf.d/default.conf
sha1:cd7f5dc8ccdc838a2d0107511c90adfe318a81e7  etc/conf.d/nginx
sha1:05f050f6ed86c5e6b48c2d2328e81583315431be  usr/sbin/nginx
sha1:c3f02ca81f7f2c6bde3f878b3176f225c7781c7d  var/lib/nginx/modules
sha1:0510312d465b86769136983657df98c1854f0b60  var/lib/nginx/run
sha1:35db17c18ce0b9f84a3cc113c8a9e94e19f632b1  var/lib/nginx/logs
sha1:7dd71afcfb14e105e80b0c0d7fce370a28a41f0a  var/lib/nginx/html/index.html
sha1:95de71d58b37f9f74bede0e91bc381d6059fc2d7  var/lib/nginx/html/50x.html
~ # ls -l /bin
total 824
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 arch -> /bin/busybox
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 ash -> /bin/busybox
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 base64 -> /bin/busybox
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 bbconfig ->
/bin/busybox
-rwxr-xr-x    1 root     root        841288 Jan 15 10:36 busybox
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 cat -> /bin/busybox
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 chgrp -> /bin/busybox
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 chmod -> /bin/busybox
lrwxrwxrwx    1 root     root            12 Jan 16 21:52 chown -> /bin/busybox
```

1.3.2.3 Debian(ubuntu)系统建议安装的基础包

在很多软件官方提供的镜像都使用的是Debian(ubuntu)的系统,比如:nginx,tomcat,mysql,httpd 等,但镜像内缺少很多常用的调试工具.当需要进入容器内进行调试管理时,可以安装以下常用工具包

```
# apt update                #安装软件前需要先更新索引
# apt install procps        #提供top,ps,free等命令
# apt install psmisc        #提供pstree,killall等命令
# apt install iputils-ping  #提供ping命令
# apt install net-tools     #提供netstat网络工具等
```

1.3.3 下载镜像

从 docker 仓库将镜像下载到本地，命令格式如下:

```
docker pull [OPTIONS] NAME[:TAG|@DIGEST]
Options:
  -a, --all-tags                Download all tagged images in the repository
      --disable-content-trust   Skip image verification (default true)
      --platform string         Set platform if server is multi-platform capable
  -q, --quiet                   Suppress verbose output


NAME: 是镜像名,一般的形式   仓库服务器:端口/项目名称/镜像名称
:TAG: 即版本号,如果不指定:TAG,则下载最新版镜像
```

镜像下载说明

```
[root@ubuntu1804 ~]#docker pull hello-world
Using default tag: latest   #默认下载最新版本
latest: Pulling from library/hello-world
1b930d010525: Pull complete  #分层下载
Digest: sha256:9572f7cdcee8591948c2963463447a53466950b3fc15a247fcad1917ca215a2f
#摘要
Status: Downloaded newer image for hello-world:latest
docker.io/library/hello-world:latest  #下载的完整地址
```

镜像下载保存的路径:

```
/var/lib/docker/overlay2/镜像ID
```

注意: 镜像下载完成后，会自动解压缩，比官网显示的可能会大很多，如: centos8.1.1911下载时只有70MB，下载完后显示237MB

范例: 从docker官网下载镜像

```
docker pull hello-world
docker pull alpine
docker pull busybox
docker pull nginx
docker pull centos
docker pull centos:centos7.7.1908
docker pull docker.io/library/mysql:5.7.29
docker pull mysql:5.6.47
```

范例: 下载镜像 alpine,busybox等镜像,查看下载的存放目录

```
[root@ubuntu1804 ~]#ls /var/lib/docker/overlay2/
l
[root@ubuntu1804 ~]#du -sh /var/lib/docker/overlay2
8.0K    /var/lib/docker/overlay2
[root@ubuntu1804 ~]#ls /var/lib/docker/overlay2/l
[root@ubuntu1804 ~]#docker pull hello-world
[root@ubuntu1804 ~]#docker pull alpine:3.11.3
3.11.3: Pulling from library/alpine
c9b1b535fdd9: Pull complete
Digest: sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Downloaded newer image for alpine:3.11.3
docker.io/library/alpine:3.11.3
[root@ubuntu1804 ~]#docker pull busybox
```

```
Using default tag: latest
latest: Pulling from library/busybox
bdbbaa22dec6: Pull complete
Digest: sha256:6915be4043561d64e0ab0f8f098dc2ac48e077fe23f488ac24b665166898115a
Status: Downloaded newer image for busybox:latest
docker.io/library/busybox:latest
[root@ubuntu1804 ~]#docker pull centos:centos8.1.1911
centos8.1.1911: Pulling from library/centos
8a29a15cefae: Pull complete
Digest: sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
Status: Downloaded newer image for centos:centos8.1.1911
docker.io/library/centos:centos8.1.1911
[root@ubuntu1804 ~]#du -sh /var/lib/docker/overlay2/*
5.9M
/var/lib/docker/overlay2/1802616f4c8e0a0b52c839431b6faa8ac21f4bd831548dcbd46943d
3f60061fa
16K
/var/lib/docker/overlay2/5773b92e1351da5e589d0573d9f22d1ec3be1e0e98edbfcddba4b83
0f12c7be2
1.3M
/var/lib/docker/overlay2/de31641b8d2207de7f08eabb5240474a1aaccfef08b6034dcee02b9
623f8d9dc
252M
/var/lib/docker/overlay2/f41df336075611f9e358e5eaf2ebd5089920a90ba68760cdec8da03
edff362f7
20K /var/lib/docker/overlay2/l
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              3.11.3              e7d92cdc71fe        7 days ago
 5.59MB
centos              centos8.1.1911      470671670cac        7 days ago
237MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
hello-world         latest              fce289e99eb9        12 months ago
1.84kB

[root@ubuntu1804 ~]#ls -l  /var/lib/docker/overlay2/l
total 16
lrwxrwxrwx 1 root root 72 Jan 25 19:51 C5ZTDYHYDTO7BQG6HX36MU6X5K ->
../de31641b8d2207de7f08eabb5240474a1aaccfef08b6034dcee02b9623f8d9dc/diff
lrwxrwxrwx 1 root root 72 Jan 25 19:57 DEXHVNUGFLFJCSJAKISOHQG7JY ->
../f41df336075611f9e358e5eaf2ebd5089920a90ba68760cdec8da03edff362f7/diff
lrwxrwxrwx 1 root root 72 Jan 25 19:51 KJ5IA5AUHFUEQXFKJA7UDUIA7A ->
../1802616f4c8e0a0b52c839431b6faa8ac21f4bd831548dcbd46943d3f60061fa/diff
lrwxrwxrwx 1 root root 72 Jan 25 19:51 ZM3U4WDNHGJJX5DXHA5M4ZWAIW ->
../5773b92e1351da5e589d0573d9f22d1ec3be1e0e98edbfcddba4b830f12c7be2/diff
```

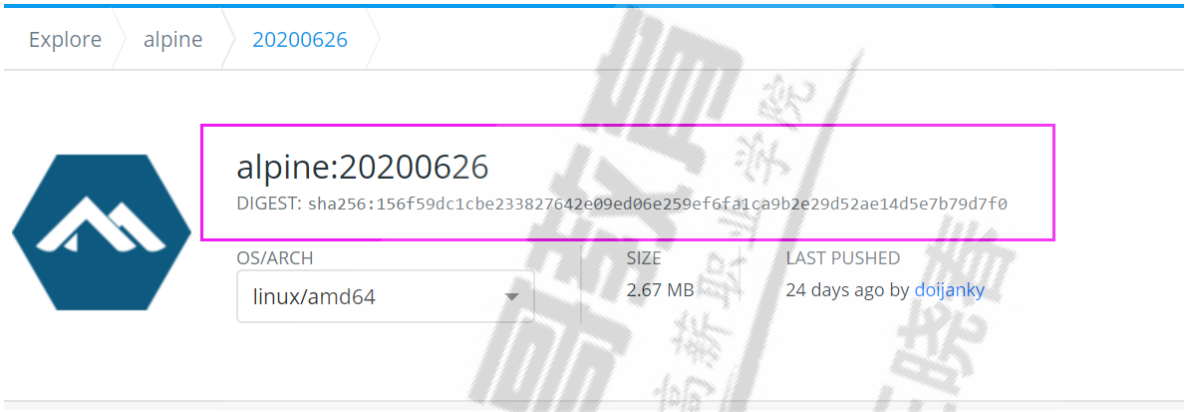
范例：指定 TAG下载特定版本的镜像

```
[root@ubuntu1804 ~]#docker pull docker.io/library/mysql:5.7.29
5.7.29: Pulling from library/mysql
804555ee0376: Pull complete
c53bab458734: Pull complete
ca9d72777f90: Pull complete
```

```
2d7aad6cb96e: Pull complete
8d6ca35c7908: Pull complete
6ddae009e760: Pull complete
327ae67bbe7b: Pull complete
9e05241b7707: Pull complete
e822978df8f0: Pull complete
14ca71ed53be: Pull complete
026afe6fd35e: Pull complete
Digest: sha256:2ca675966612f34b4036bbcfa68cb049c03e34b561fba0f88954b03931823d29
Status: Downloaded newer image for mysql:5.7.29
docker.io/library/mysql:5.7.29
[root@ubuntu1804 ~]#docker pull mysql:5.6.47
```

范例: 指定DIGEST下载特定版本的镜像

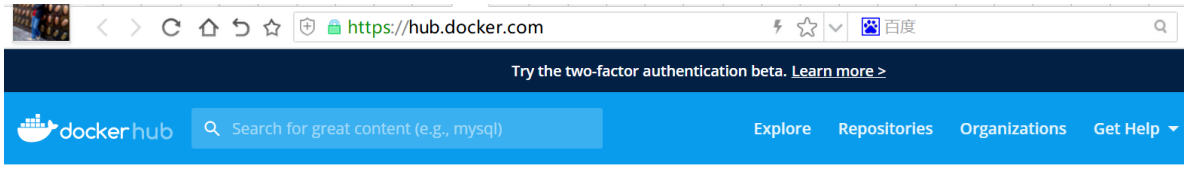
先到 hub.docker.com查到指定版本的DIGEST




```
[root@ubuntu1804 ~]#docker pull
alpine@sha256:156f59dc1cbe233827642e09ed06e259ef6fa1ca9b2e29d52ae14d5e7b79d7f0
sha256:156f59dc1cbe233827642e09ed06e259ef6fa1ca9b2e29d52ae14d5e7b79d7f0: Pulling
from library/alpine
5d2415897100: Pull complete
Digest: sha256:156f59dc1cbe233827642e09ed06e259ef6fa1ca9b2e29d52ae14d5e7b79d7f0
Status: Downloaded newer image for
alpine@sha256:156f59dc1cbe233827642e09ed06e259ef6fa1ca9b2e29d52ae14d5e7b79d7f0
docker.io/library/alpine@sha256:156f59dc1cbe233827642e09ed06e259ef6fa1ca9b2e29d5
2ae14d5e7b79d7f0

[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              <none>              3c791e92a856        3 weeks ago
5.57MB
```

1.3.4 docker 镜像加速配置

docker 镜像官方的下载站点是: https://hub.docker.com/



从国内下载官方的镜像站点有时候会很慢，因此可以更改docker配置文件添加一个加速器，可以通过加速器达到加速下载镜像的目的

国内有许多公司都提供了docker 加速镜像，比如：阿里云，腾讯云，网易云，以下以阿里云为例

1.3.4.1 阿里云获取加速地址

浏览器打开http://cr.console.aliyun.com，注册或登录阿里云账号，点击左侧的镜像加速器，将会得到一个专属的加速地址，而且下面有使用配置说明:

1.3.4.2 docker 镜像加速配置

```
1. 安装 / 升级Docker客户端
推荐安装1.10.0以上版本的Docker客户端，参考文档 docker-ce

2. 配置镜像加速器
修改daemon配置文件/etc/docker/daemon.json来使用加速器

mkdir -p /etc/docker
tee /etc/docker/daemon.json <<-'EOF'
{
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}
```

```
EOF
#网易云:  http://hub-mirror.c.163.com/
#腾讯云:  https://mirror.ccs.tencentyun.com

systemctl daemon-reload
systemctl restart docker
```

范例:

```
[root@ubuntu1804 ~]#docker info |tail
WARNING: the overlay storage-driver is deprecated, and will be removed in a
future release.
 ID: IZHJ:WPIN:BRMC:XQUI:VVVR:UVGK:NZBM:YQXT:JDWB:33RS:45V7:SQWJ
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Live Restore Enabled: false
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
[root@ubuntu1804 ~]#cat /etc/docker/daemon.json
{
  "storage-driver": "overlay",
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}

[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#docker info |tail
WARNING: the overlay storage-driver is deprecated, and will be removed in a
future release.
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/
 Live Restore Enabled: false

[root@ubuntu1804 ~]
```

1.3.5 查看本地镜像

docker images 可以查看下载至本地的镜像

格式:

```
docker images [OPTIONS] [REPOSITORY[:TAG]]
docker image ls [OPTIONS] [REPOSITORY[:TAG]]

#常用选项:
-q, --quiet            Only show numeric IDs
-a, --all              Show all images (default hides intermediate images)
    --digests          Show digests
    --no-trunc         Don't truncate output
-f, --filter filter    Filter output based on conditions provided
    --format string    Pretty-print images using a Go template
```

执行结果的显示信息说明:

```
REPOSITORY      #镜像所属的仓库名称
TAG             #镜像版本号（标识符），默认为latest
IMAGE ID        #镜像唯一ID标识,如果ID相同,说明是同一个镜像有多个名称
CREATED         #镜像在仓库中被创建时间
VIRTUAL SIZE    #镜像的大小
```

Repository仓库

- 由某特定的docker镜像的所有迭代版本组成的镜像仓库
- 一个Registry中可以存在多个Repository
- Repository可分为“顶层仓库”和“用户仓库”
- Repository用户仓库名称一般格式为“用户名/仓库名”
- 每个Repository仓库可以包含多个Tag(标签),每个标签对应一个镜像

范例:

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              3.11.3              e7d92cdc71fe        7 days ago
 5.59MB
centos              centos8.1.1911      470671670cac        7 days ago
237MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
hello-world         latest              fce289e99eb9        12 months ago
1.84kB
[root@ubuntu1804 ~]#docker images -q
e7d92cdc71fe
470671670cac
6d5fcfe5ff17
fce289e99eb9

#显示完整的ImageID
[root@ubuntu1804 ~]#docker images --no-trunc
REPOSITORY          TAG                 IMAGE ID
                                  CREATED             SIZE
tomcat              9.0.37-v1
sha256:b8d669ebf99e65d5ed69378d0d53f054e7de4865d335ab7aa0a7a5508e1369f7   47
hours ago        652MB
tomcat              latest
sha256:df72227b40e1985fa5ad529b9ca6582612a41d8f1ddf3a1bea1aa2cfcfa8fb07   5 days
ago          647MB
```

```
nginx               latest
sha256:0901fa9da894a8e9de5cb26d6749eaffb67b373dc1ff8a26c46b23b1175c913a   11
days ago         132MB
ubuntu              latest
sha256:adafef2e596ef06ec2112bc5a9663c6a4f59a3dfd4243c9cabe06c8748e7f288   2
weeks ago         73.9MB
busybox             latest
sha256:c7c37e472d31c1685b48f7004fd6a64361c95965587a951692c5f298c6685998   3
weeks ago         1.22MB
alpine              latest
sha256:a24bb4013296f61e89ba57005a7b3e52274d8edd3ae2077d04395f806b63d83e   7
weeks ago         5.57MB
redis               5.0.9-alpine3.11
sha256:3661c84ee9d0f6312a076b69eb2bd112674cadb70ef7e1594c4f00193f8df08e   2
months ago        29.8MB

#只查看指定REPOSITORY的镜像
[root@ubuntu1804 ~]#docker images tomcat
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
tomcat              9.0.37-v1           b8d669ebf99e        47 hours ago
652MB
tomcat              latest              df72227b40e1        5 days ago
647MB
```

范例: 查看指定镜像的详细信息

```
[root@centos8 ~]#podman image inspect alpine
[
    {
        "Id":
"e7d92cdc71feacf90708cb59182d0df1b911f8ae022d29e8e95d75ca6a99776a",
        "Digest":
"sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45",
        "RepoTags": [
            "docker.io/library/alpine:latest"
        ],
        "RepoDigests": [

 "docker.io/library/alpine@sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c9
0e6e1297bfa1bc0c45"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2020-01-18T01:19:37.187497623Z",
        "Config": {
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh"
            ]
        },
        "Version": "18.06.1-ce",
        "Author": "",
```

```
        "Architecture": "amd64",
        "Os": "linux",
        "Size": 5859847,
        "VirtualSize": 5859847,
        "GraphDriver": {
            "Name": "overlay",
            "Data": {
                "MergedDir":
"/var/lib/containers/storage/overlay/5216338b40a7b96416b8b9858974bbe4acc3096ee60
acbc4dfb1ee02aecceb10/merged",
                "UpperDir":
"/var/lib/containers/storage/overlay/5216338b40a7b96416b8b9858974bbe4acc3096ee60
acbc4dfb1ee02aecceb10/diff",
                "WorkDir":
"/var/lib/containers/storage/overlay/5216338b40a7b96416b8b9858974bbe4acc3096ee60
acbc4dfb1ee02aecceb10/work"
            }
        },
        "RootFS": {
            "Type": "layers",
            "Layers": [

 "sha256:5216338b40a7b96416b8b9858974bbe4acc3096ee60acbc4dfb1ee02aecceb10"
            ]
        },
        "Labels": null,
        "Annotations": {},
        "ManifestType": "application/vnd.docker.distribution.manifest.v2+json",
        "User": "",
        "History": [
            {
                "created": "2020-01-18T01:19:37.02673981Z",
                "created_by": "/bin/sh -c #(nop) ADD
file:e69d441d729412d24675dcd33e04580885df99981cec43de8c9b24015313ff8e in / "
            },
            {
                "created": "2020-01-18T01:19:37.187497623Z",
                "created_by": "/bin/sh -c #(nop)  CMD [\"/bin/sh\"]",
                "empty_layer": true
            }
        ]
    }
]
```


1.3.6 镜像导出

利用docker save命令可以将从本地镜像导出为一个打包 tar文件，然后复制到其他服务器进行导入使用

格式:

```
docker save [OPTIONS] IMAGE [IMAGE...]
选项:
-o, --output string   Write to a file, instead of STDOUT
```

常见用法:

```
docker save -o /path/file.tar IMAGE1 IMAGE2 ...
docker save IMAGE1 IMAGE2 ... > /path/file.tar
```

范例: 镜像导出

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx               latest              5ad3bd0e67a9        3 days ago
127MB
alpine              3.11.3              e7d92cdc71fe        7 days ago
 5.59MB
centos              centos8.1.1911      470671670cac        7 days ago
237MB
centos              latest              470671670cac        7 days ago
237MB
mysql               5.6.47              742f7d5a4104        10 days ago
302MB
mysql               5.7.29              b598110d0fff        10 days ago
435MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
hello-world         latest              fce289e99eb9        12 months ago
1.84kB
[root@ubuntu1804 ~]#docker save mysql:5.7.29 alpine:3.11.3 -o /data/myimages.tar
#或者
[root@ubuntu1804 ~]#docker save mysql:5.7.29 alpine:3.11.3 > /data/myimages.tar

[root@ubuntu1804 ~]#scp /data/myimages.tar   10.0.0.7:/data
```

1.3.7 镜像导入

利用docker load命令可以将镜像导出的压缩文件再导入

格式:

```
docker load [OPTIONS]

#选项
-i, --input string   Read from tar archive file, instead of STDIN
-q, --quiet          Suppress the load output
```

范例: 镜像导入

```
[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@centos7 ~]#docker load -i /data/myimages.tar

#或者
[root@centos7 ~]#docker load < /data/myimages.tar

[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              3.11.3              e7d92cdc71fe        7 days ago
 5.59MB
mysql               5.7.29              b598110d0fff        10 days ago
435MB
```

范例: 一次导出多个镜像

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              latest              e7d92cdc71fe        7 days ago
 5.59MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
[root@ubuntu1804 ~]#docker save busybox alpine > /all.tar
[root@ubuntu1804 ~]#ll -h /opt/all.tar
-rw-r--r-- 1 root root 7.0M Jan 25 22:12 /opt/all.tar
[root@ubuntu1804 ~]#docker rmi -f `docker images -q`
Untagged: alpine:latest
Deleted: sha256:e7d92cdc71feacf90708cb59182d0df1b911f8ae022d29e8e95d75ca6a99776a
Deleted: sha256:5216338b40a7b96416b8b9858974bbe4acc3096ee60acbc4dfb1ee02aecceb10
Untagged: busybox:latest
Deleted: sha256:6d5fcfe5ff170471fcc3c8b47631d6d71202a1fd44cf3c147e50c8de21cf0648
Deleted: sha256:195be5f8be1df6709dafbba7ce48f2eee785ab7775b88e0c115d8205407265c5
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@ubuntu1804 ~]#docker load -i /opt/all.tar
5216338b40a7: Loading layer
[==================================================>]  5.857MB/5.857MB
Loaded image: alpine:latest
195be5f8be1d: Loading layer
[==================================================>]  1.437MB/1.437MB
Loaded image: busybox:latest
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              latest              e7d92cdc71fe        7 days ago
 5.59MB
busybox             latest              6d5fcfe5ff17        4 weeks ago
1.22MB
```

1.3.8 删除镜像

docker rmi 命令可以删除本地镜像

格式

```
docker rmi [OPTIONS] IMAGE [IMAGE...]
docker image rm [OPTIONS] IMAGE [IMAGE...]

#选项:
-f, --force      Force removal of the image
    --no-prune   Do not delete untagged parents
```

范例:

```
[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              3.11.3              e7d92cdc71fe        7 days ago
 5.59MB
mysql               5.7.29              b598110d0fff        10 days ago
435MB
[root@centos7 ~]#docker rmi b59811
Untagged: mysql:5.7.29
Deleted: sha256:b598110d0fffc42862269a25d8767dd95764d0740f318fd6c0a097f8a22de5bf
Deleted: sha256:908aaab4c4b8d0cbbc68c96a0e3820aa74fd1dee5499e2ca326bc8fd7312f689
Deleted: sha256:e2f2e83f295186b00e3c0d119ef3204b509d552972694e34cee7c1675d157b8a
Deleted: sha256:99b4e48b1be76f50741db02a38c783bf698b1b76808cc6bb5e3fdd65ee2897c6
Deleted: sha256:79c1efa7bde3ac754af64779452ca913fa1f281b44c9dbad25cc322a51ac69b1
Deleted: sha256:5f7c68324b959d2c806db18d02f153bc810f9842722415e077351bc834cc8578
Deleted: sha256:338fc0cd3fb4b87a2b83d274e8fbf475fbde19947c4ac5c5eb6e981a6fb0e8f0
Deleted: sha256:f7a4ccab931f1d1e861961eb951a7806d91ccb375e737fe1f84282f6bbafd2be
Deleted: sha256:f388e1092f8fb931a3cd07a7381bd9707d19526ff81f8b624e932f4919c27a3e
Deleted: sha256:e209b7a884b4d2e9d56bbac40ced48f2caa6a19e7ad6eb6dd20ff754f3af2c5d
Deleted: sha256:2401cf11c5455d505ef49657afcc709197ffcdfc9bd732508e9b62578a30b3a5
Deleted: sha256:814c70fdae62bc26c603bfae861f00fb1c77fc0b1ee8d565717846f4df24ae5d
[root@centos7 ~]#docker rmi alpine:3.11.3
Untagged: alpine:3.11.3
Deleted: sha256:e7d92cdc71feacf90708cb59182d0df1b911f8ae022d29e8e95d75ca6a99776a
Deleted: sha256:5216338b40a7b96416b8b9858974bbe4acc3096ee60acbc4dfb1ee02aecceb10
[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
```

范例: 删除多个镜像

```
[root@ubuntu1804 ~]#docker rmi nginx tomcat
```

范例: 强制删除正在使用的镜像，也会删除对应的容器

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                     COMMAND                  CREATED
        STATUS                          PORTS                         NAMES
b5a0d2e1e1d0        centos:centos8.1.1911   "bash"                        41 minutes
ago      Up 41 minutes                                                 jolly_burnell
[root@ubuntu1804 ~]#docker rmi centos:centos8.1.1911
Error response from daemon: conflict: unable to remove repository reference
"centos:centos8.1.1911" (must force) - container b5a0d2e1e1d0 is using its
referenced image 470671670cac
[root@ubuntu1804 ~]#docker rmi -f centos:centos8.1.1911
Untagged: centos:centos8.1.1911
Untagged:
centos@sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                          PORTS                     NAMES
[root@ubuntu1804 ~]#
```

范例: 删除所有镜像

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
<none>              <none>              470671670cac        7 days ago
237MB
mysql               5.6.47              742f7d5a4104        10 days ago
302MB
[root@ubuntu1804 ~]#docker rmi  -f `docker images -q`
Deleted: sha256:470671670cac686c7cf0081e0b37da2e9f4f768ddc5f6a26102ccd1c6954c1ee
Deleted: sha256:0683de2821778aa9546bf3d3e6944df779daba1582631b7ea3517bb36f9e4007
Untagged: mysql:5.6.47
Untagged:
mysql@sha256:9527bae58991a173ad7d41c8309887a69cb8bd178234386acb28b51169d0b30e
Deleted: sha256:742f7d5a4104969fcac8054cf9201f5656096f0a58d10947a4a41a8e1d7d9f91
Deleted: sha256:62530ddc9fe5f85609da4397d9e0a88b422d15dbc42664d7477d1deccb51a0d9
Deleted: sha256:41309d62590858b6375bd3c4e9d07bd73e4ea5062343a81641453033424e7aba
Deleted: sha256:43cb5cbfcce71f7fdd80b121aa4d0c5eea9ed0bfcf4a2a0e55a8c5ddb78d4368
Deleted: sha256:56806995b55c0e80ee0daec20f32f8293a3d8ccf151895cd1767a1d27eab1977
Deleted: sha256:7dab204c1f667de5090d809019fab7d6e2323e371407b38e8d311ba458009234
Deleted: sha256:1a7c2d06fdee90c4b06a55793f9e45120a82ff476e0fea5618efd8a268f7d303
Deleted: sha256:f7a4ccab931f1d1e861961eb951a7806d91ccb375e737fe1f84282f6bbafd2be
Deleted: sha256:f388e1092f8fb931a3cd07a7381bd9707d19526ff81f8b624e932f4919c27a3e
Deleted: sha256:e209b7a884b4d2e9d56bbac40ced48f2caa6a19e7ad6eb6dd20ff754f3af2c5d
Deleted: sha256:2401cf11c5455d505ef49657afcc709197ffcdfc9bd732508e9b62578a30b3a5
Deleted: sha256:814c70fdae62bc26c603bfae861f00fb1c77fc0b1ee8d565717846f4df24ae5d
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
```

1.3.9 镜像打标签

docker tag 可以给镜像打标签，类似于起别名,但通常要遵守一定的命名规范,才可以上传到指定的仓库格式

```
docker tag SOURCE_IMAGE[:TAG] TARGET_IMAGE[:TAG]

#TARGET_IMAGE[:TAG]格式一般形式
仓库主机FQDN或IP[:端口]/项目名(或用户名)/image名字:版本
```

TAG默认为latest

范例:

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              latest              e7d92cdc71fe        11 days ago
5.59MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
[root@ubuntu1804 ~]#docker tag  alpine alpine:3.11
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              3.11                e7d92cdc71fe        11 days ago
5.59MB
alpine              latest              e7d92cdc71fe        11 days ago
5.59MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

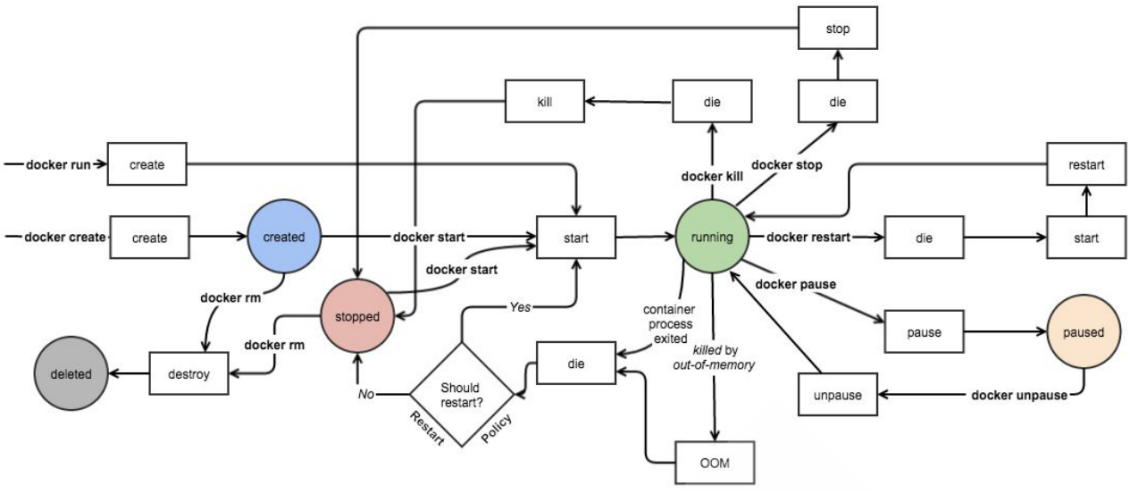
总结: 企业使用镜像及常见操作: 搜索、下载、导出、导入、删除

命令总结:

```
docker search centos
docker pull alpine
docker images
docker save > /opt/centos.tar #centos #导出镜像
docker load -i centos-latest.tar.xz  #导入本地镜像
docker rmi 镜像ID/镜像名称  #删除指定ID的镜像，此镜像对应容器正启动镜像不能被删除，除非将容器全部关闭
```

1.4 容器操作基础命令

容器生命周期



容器相关命令

```
[root@ubuntu1804 ~]#docker container

Usage:  docker container COMMAND

Manage containers

Commands:
  attach      Attach local standard input, output, and error streams to a
running container
  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 to files or directories on a container's
filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit
codes

Run 'docker container COMMAND --help' for more information on a command.
```

1.4.1 启动容器

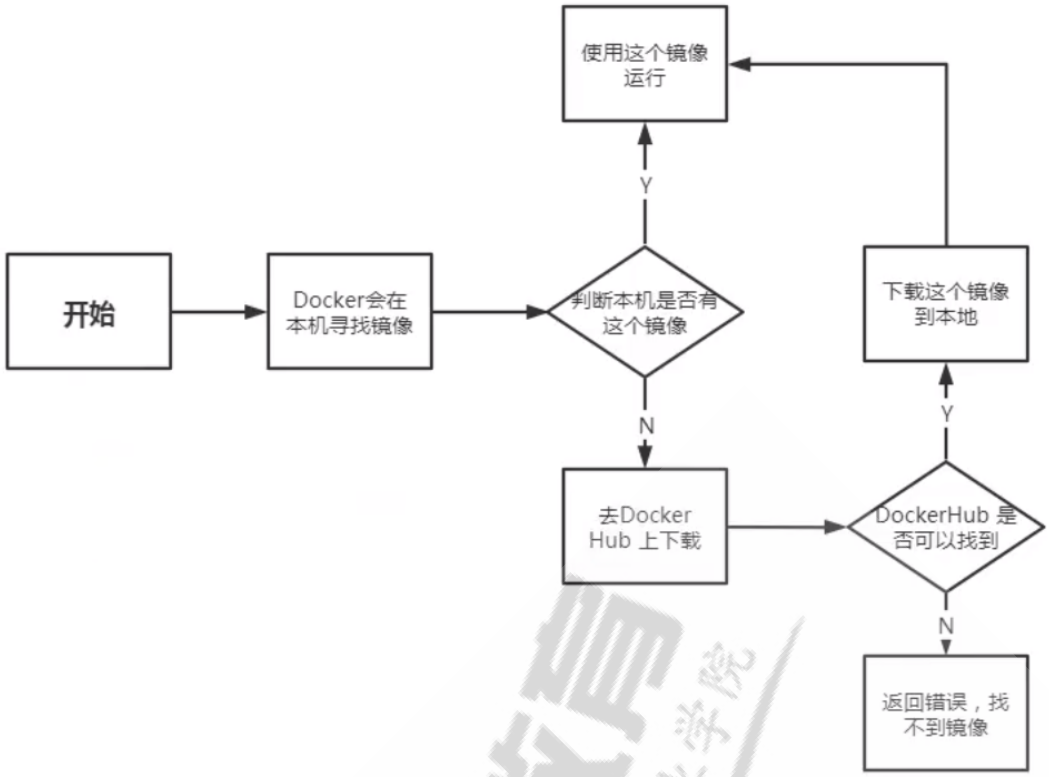
docker run 可以启动容器，进入到容器，并随机生成容器ID和名称

1.4.1.1 启动第一个容器

范例: 运行docker 的 hello world

```
[root@centos8 ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:9572f7cdcee8591948c2963463447a53466950b3fc15a247fcad1917ca215a2f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

[root@centos8 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
hello-world         latest              fce289e99eb9        12 months ago
1.84kB
[root@centos8 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
7f53a2a74edc        hello-world         "/hello"            21 seconds ago
Exited (0) 19 seconds ago                       nifty_yalow
```

1.4.1.2 启动容器的流程



1.4.1.3 启动容器用法

帮助: man docker-run

命令格式:

```
docker run [选项]  [镜像名]  [shell命令] [参数]

#选项:
-i, --interactive    Keep STDIN open even if not attached，通常和-t一起使用
-t, --tty            分配pseudo-TTY，通常和-i一起使用,注意对应的容器必须运行shell才支持进入
-d, --detach         Run container in background and print container ID,台后运行，默认前台
--name string        Assign a name to the container
--h, --hostname string  Container host name
--rm                 Automatically remove the container when it exits
-p, --publish list   Publish a container's port(s) to the host
-P, --publish-all    Publish all exposed ports to random ports
--dns list           Set custom DNS servers
--entrypoint string  Overwrite the default ENTRYPOINT of the image
--restart policy
--privileged         Give extended privileges to container
-e, --env=[]         Set environment variables
--env-file=[]        Read in a line delimited file of environment variables
```

--restart 可以指定四种不同的policy

policy	说明
no	Default is no，Do not automatically restart the container when it exits.
on-failure[:max-retries]	on-failure[:max-retries] Restart only if the container exits with a non-zero exit status. Optionally, limit the number of restart retries the Docker daemon attempts.
always	Always restart the container regardless of the exit status. When you specify always, the Docker daemon will try to restart the container indefinitely. The container will also always start on daemon startup, regardless of the current state of the container.
unless-stopped	Always restart the container regardless of the exit status, but do not start it on daemon startup if the container has been put to a stopped state before.

注意: 容器启动后,如果容器内没有前台运行的进程,将自动退出停止

从容器内退出,并停止容器

```
 exit
```

从容器内退出,且容器不停止

```
同时按三个键，ctrl+p+q
```

范例: 运行容器

```
#启动容器时会自动随机字符作为容器名
[root@ubuntu1804 ~]#docker run  alpine
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
95967eaefd6a        alpine              "/bin/sh"           8 seconds ago
Exited (0) 6 seconds ago                        confident_elion
```

范例: 一次性运行容器中命令

```
#启动的容器在执行完shell命令就退出，用于测试
[root@ubuntu1804 ~]#docker run  busybox echo "Hello laowang"
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
91f30d776fb2: Pull complete
Digest: sha256:9ddee63a712cea977267342e8750ecbc60d3aab25f04ceacfa795e6fce341793
Status: Downloaded newer image for busybox:latest
Hello laowang

[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
7873aed1b5dd        busybox             "echo 'Hello laowang'"   19 seconds ago
     Exited (0) 18 seconds ago                       pedantic_varahamihira
```

范例: 指定容器名称

```
#注意每个容器的名称要唯一
[root@ubuntu1804 ~]#docker run --name a1 alpine
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
dd06368a4f56        alpine              "/bin/sh"           2 minutes ago
Exited (0) 8 seconds ago                        a1
```

范例: 运行交互式容器并退出

```
[root@ubuntu1804 ~]#docker run -it docker.io/busybox  sh
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
bdbbaa22dec6: Pull complete
Digest: sha256:6915be4043561d64e0ab0f8f098dc2ac48e077fe23f488ac24b665166898115a
Status: Downloaded newer image for busybox:latest
/ # exit

#用exit退出后容器也停止
[root@ubuntu1804 ~]#docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
cd8c2a7f39d5        busybox             "sh"                8 seconds ago
Exited (0) 1 second ago                         vigorous_leakey


[root@ubuntu1804 ~]#docker run -it docker.io/busybox  sh
Unable to find image 'busybox:latest' locally
latest: Pulling from library/busybox
bdbbaa22dec6: Pull complete
Digest: sha256:6915be4043561d64e0ab0f8f098dc2ac48e077fe23f488ac24b665166898115a
Status: Downloaded newer image for busybox:latest
/ #同时按三个键:ctrl+p+q


#用同时按三个键ctrl+p+q退出后容器不会停止
[root@ubuntu1804 ~]#docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
cd8c2a7f39d5        busybox             "sh"                8 seconds ago
Up 10 seconds               silly_villani
```

范例: 设置容器内的主机名

```
[root@ubuntu1804 ~]#docker run -it --name a1 -h a1.magedu.org alpine
/ # hostname
a1.magedu.org
/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
```

```
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  a1.magedu.org a1
/ # cat /etc/resolv.conf
# This file is managed by man:systemd-resolved(8). Do not edit.
#
# This is a dynamic resolv.conf file for connecting local clients directly to
# all known uplink DNS servers. This file lists all configured search domains.
#
# Third party programs must not access this file directly, but only through the
# symlink at /etc/resolv.conf. To manage man:resolv.conf(5) in a different way,
# replace this symlink by a static file or a different symlink.
#
# See man:systemd-resolved.service(8) for details about the supported modes of
# operation for /etc/resolv.conf.

nameserver 180.76.76.76
nameserver 223.6.6.6
search magedu.com magedu.org
```

范例: 一次性运行容器，退出后立即删除，用于测试

```
[root@ubuntu1804 ~]#docker run  --rm  alpine cat /etc/issue
Welcome to Alpine Linux 3.11
Kernel \r on an \m (\l)

[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

范例: 创建容器后直接进入并退出

退出两种方式:

- exit 容器也停止
- 按ctrl+p+q 容器不停止

```
#，执行exit退出后容器关闭
[root@ubuntu1804 ~]#docker run -it --name alpine2 alpine
/ # cat /etc/issue
Welcome to Alpine Linux 3.11
Kernel \r on an \m (\l)


/ # exit  #退出容器,容器也停止运行
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
6d64f47a83e6        alpine              "/bin/sh"           13 seconds ago
Exited (0) 6 seconds ago                        alpine2
edd2ac2690e6        alpine              "/bin/sh"           52 seconds ago
Exited (0) 51 seconds ago                       alpine1
[root@ubuntu1804 ~]#docker run -it --name alpine3 alpine
#同时按ctrl+p+q 三个键退出后,容器不停止
```

```
/ # [root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
df428caf7128        alpine              "/bin/sh"           8 seconds ago
Up 7 seconds                                            alpine3
6d64f47a83e6        alpine              "/bin/sh"           26 seconds ago
Exited (0) 20 seconds ago                               alpine2
edd2ac2690e6        alpine              "/bin/sh"           About a minute ago
Exited (0) About a minute ago                           alpine1
```

什么是守护式容器:

- 能够长期运行
- 无需交互式会话
- 适合运行应用程序和服务

范例: 启动前台守护式容器

```
[root@ubuntu1804 ~]#docker run nginx
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
6ec8c9369e08: Pull complete
d3cb09a117e5: Pull complete
7ef2f1459687: Pull complete
e4d1bf8c9482: Pull complete
795301d236d7: Pull complete
Digest: sha256:0e188877aa60537d1a1c6484b8c3929cfe09988145327ee47e8e91ddf6f76f5c
Status: Downloaded newer image for nginx:latest
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to
perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-
default.sh
10-listen-on-ipv6-by-default.sh: Getting the checksum of
/etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: Enabled listen on IPv6 in
/etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-
templates.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
172.17.0.1 - - [28/Jul/2020:13:09:19 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.58.0" "-"
172.17.0.4 - - [28/Jul/2020:13:12:49 +0000] "GET / HTTP/1.1" 200 612 "-" "Wget"
"-"



[root@ubuntu1804 ~]#docker run --rm --name b1 busybox  wget -qO - 172.17.0.3
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
```

```
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

范例: 启动后台守护式容器

```
[root@ubuntu1804 ~]#docker run -d nginx
888685a2487cf8150d264cb3086f78d0c3bddeb07b8ea9786aa3a564157a4cb8
[root@ubuntu1804 ~]#docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
888685a2487c        nginx               "/docker-entrypoint.…"   8 seconds ago
    Up 6 seconds        80/tcp              busy_goodall

#有些容器后台启动不会持续运行
[root@ubuntu1804 ~]#docker run -d --name alpine4 alpine
3a05bbf66dac8a6ac9829c876bdb5fcb70832bf4a2898d68f6979cd8e8c517cb
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
3a05bbf66dac        alpine              "/bin/sh"           3 seconds ago
Exited (0) 2 seconds ago                            alpine4
df428caf7128        alpine              "/bin/sh"           30 seconds ago
Up 28 seconds                                       alpine3
6d64f47a83e6        alpine              "/bin/sh"           48 seconds ago
Exited (0) 41 seconds ago                           alpine2
edd2ac2690e6        alpine              "/bin/sh"           About a minute ago
Exited (0) About a minute ago                       alpine1
[root@ubuntu1804 ~]#docker run -td --name alpine5 alpine
868b33da850cfcc7db8b84150fb9c7686b577889f10425bb4c5e17f28cf68a29
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                          PORTS               NAMES
868b33da850c        alpine              "/bin/sh"           2 seconds ago
Up 1 second                                         alpine5
3a05bbf66dac        alpine              "/bin/sh"           23 seconds ago
Exited (0) 23 seconds ago                           alpine4
df428caf7128        alpine              "/bin/sh"           50 seconds ago
Up 49 seconds                                       alpine3
6d64f47a83e6        alpine              "/bin/sh"           About a minute ago
Exited (0) About a minute ago                       alpine2
edd2ac2690e6        alpine              "/bin/sh"           About a minute ago
Exited (0) About a minute ago                       alpine1
```

```
[root@ubuntu1804 ~]#
```

范例: 开机自动运行容器

```
#默认容器不会自动启动
[root@ubuntu1804 ~]#docker run -d --name nginx  -p 80:80  nginx
bce473b8b1d2f728847cdc32b664cca1bd7578bf7bdac850b501e2e5557a718a
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
      STATUS              PORTS                                      NAMES
bce473b8b1d2        nginx               "nginx -g 'daemon of…"   3 seconds ago
      Up 2 seconds        0.0.0.0:80->80/tcp
[root@ubuntu1804 ~]#reboot
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE       COMMAND     CREATED     STATUS        PORTS
  NAMES

#设置容器总是运行
[root@ubuntu1804 ~]#docker run -d --name nginx --restart=always -p 80:80  nginx
[root@ubuntu1804 ~]#reboot
[root@ubuntu1804 ~]#docker ps
CONTAINER ID   IMAGE        COMMAND     CREATED        STATUS      PORTS
NAMES
dbdba90076e1 nginx"nginx -g 'daemon of…"  About a minute agoUp 49
seconds0.0.0.0:80->80/tcp   nginx
```

--privileged 选项

大约在0.6版，--privileged 选项被引入docker。使用该参数，container内的root拥有真正的root权限。
否则，container内的root只是外部的一个普通用户权限。privileged启动的容器，可以看到很多host上的设备，并且可以执行mount。甚至允许你在docker容器中启动docker容器。

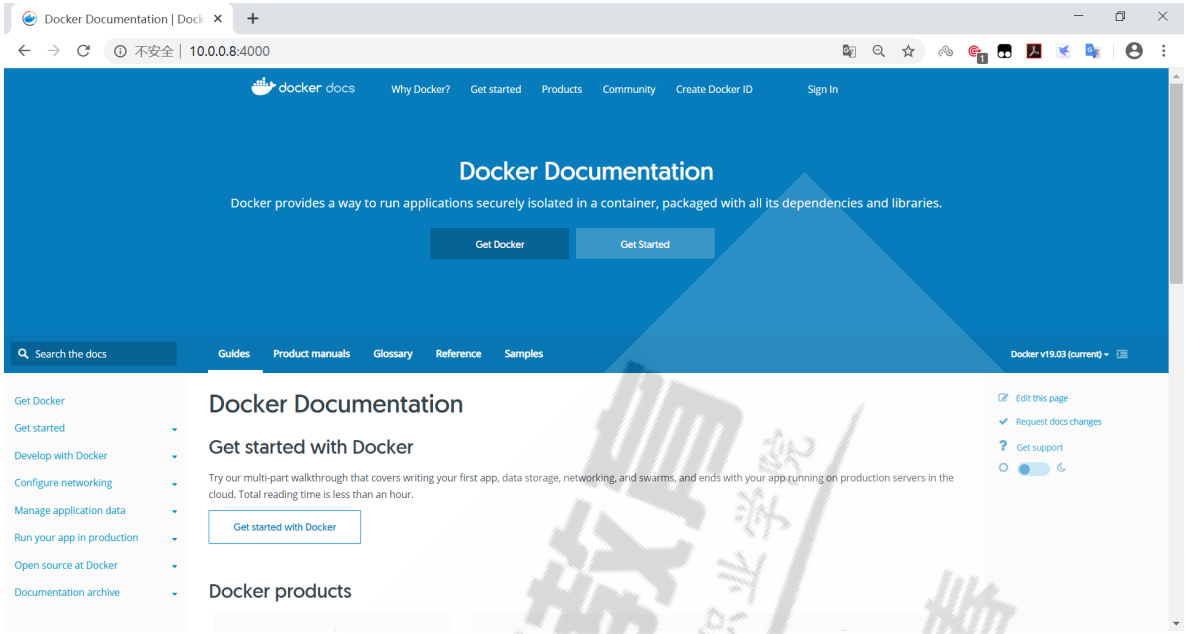
范例: 使用--privileged 让容器获取 root 权限

```
[root@centos8 ~]#podman run -it   centos
[root@382ab09932a7 /]#cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)
[root@382ab09932a7 /]# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0  200G  0 disk
|-sda1   8:1    0    1G  0 part
|-sda2   8:2    0  100G  0 part
|-sda3   8:3    0   50G  0 part
|-sda4   8:4    0    1K  0 part
`-sda5   8:5    0    2G  0 part [SWAP]
sr0     11:0    1    7G  0 rom
[root@382ab09932a7 /]# mount /dev/sda3 /mnt
mount: /mnt: permission denied.
[root@382ab09932a7 /]# exit
exit

#利用--privileged 选项运行容器
[root@centos8 ~]#podman run -it --privileged   centos
#可以看到宿主机的设备
[root@a6391a8f82e3 /]# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
```

```
sda      8:0    0  200G  0 disk
|-sda1   8:1    0    1G  0 part
|-sda2   8:2    0  100G  0 part
|-sda3   8:3    0   50G  0 part
|-sda4   8:4    0    1K  0 part
`-sda5   8:5    0    2G  0 part [SWAP]
sr0     11:0    1    7G  0 rom
[root@a6391a8f82e3 /]# df
Filesystem     1K-blocks    Used Available Use% Mounted on
overlay        104806400 2754832 102051568   3% /
tmpfs              65536       0     65536   0% /dev
tmpfs             408092    5892    402200   2% /etc/hosts
shm                64000       0     64000   0% /dev/shm
tmpfs             408092       0    408092   0% /sys/fs/cgroup

[root@a6391a8f82e3 /]# mount /dev/sda3 /mnt
[root@a6391a8f82e3 /]# df
Filesystem     1K-blocks    Used Available Use% Mounted on
overlay        104806400 2754632 102051768   3% /
tmpfs              65536       0     65536   0% /dev
tmpfs             408092    5892    402200   2% /etc/hosts
shm                64000       0     64000   0% /dev/shm
tmpfs             408092       0    408092   0% /sys/fs/cgroup
/dev/sda3       52403200  619068  51784132   2% /mnt
[root@a6391a8f82e3 /]# touch /mnt/containter.txt
[root@a6391a8f82e3 /]# echo container data > /mnt/containter.txt
[root@a6391a8f82e3 /]# cat  /mnt/containter.txt
container data
[root@a6391a8f82e3 /]#

#在宿主机查看是否生成文件
[root@centos8 ~]#lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda      8:0    0  200G  0 disk
├─sda1   8:1    0    1G  0 part /boot
├─sda2   8:2    0  100G  0 part /
├─sda3   8:3    0   50G  0 part /data
├─sda4   8:4    0    1K  0 part
└─sda5   8:5    0    2G  0 part [SWAP]
sr0     11:0    1    7G  0 rom
[root@centos8 ~]#ll /data/containter.txt
-rw-r--r-- 1 root root 25 Feb 29 12:26 /data/containter.txt
[root@centos8 ~]#cat /data/containter.txt
container data
[root@centos8 ~]#echo host data >> /data/containter.txt
[root@centos8 ~]#cat /data/containter.txt
container data
host data

#在容器内可看文件是否发生变化
[root@a6391a8f82e3 /]# cat  /mnt/containter.txt
container data
host data
```

范例: 运行docker官方文档容器

```
[root@centos8 ~]#podman run -it -d -p 4000:4000 docs/docker.github.io:latest
[root@centos8 ~]#podman images docs/docker.github.io
REPOSITORY                          TAG      IMAGE ID       CREATED       SIZE
docker.io/docs/docker.github.io     latest   ffd9131eeee7   2 days ago    1.99 GB
#用浏览器访问http://localhost:4000/可以看到下面docker文档资料
```



1.4.2 查看容器信息

1.4.2.1 显示当前存在容器

格式

```
docker ps [OPTIONS]
docker container ls [OPTIONS]
选项:
-a, --all             Show all containers (default shows just running)
-q, --quiet           Only display numeric IDs
-s, --size            Display total file sizes
-f, --filter filter   Filter output based on conditions provided
-l, --latest          Show the latest created container (includes all states)
-n, --last int        Show n last created containers (includes all states)
(default -1)
```

范例:

```
#显示运行的容器
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
d7ece7f62532        centos              "/bin/bash"         23 seconds ago
Up 22 seconds                           ecstatic_franklin

#显示全部容器，包括退出状态的容器
[root@ubuntu1804 ~]#docker ps  -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
d7ece7f62532        centos              "/bin/bash"              27 seconds ago
     Up 26 seconds                                  ecstatic_franklin
```

```
dcdf71d17177        busybox             "/bin/echo 'hello wo…"   8 minutes ago
    Exited (0) 8 minutes ago                       cool_jepsen

#只显示容器ID
[root@ubuntu1804 ~]#docker ps  -a -q
d7ece7f62532
dcdf71d17177

#显示容器大小
[root@ubuntu1804 ~]#docker ps  -a -s
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES               SIZE
d7ece7f62532        centos              "/bin/bash"              51 seconds ago
     Up 50 seconds                                   ecstatic_franklin   0B
(virtual 237MB)
dcdf71d17177        busybox             "/bin/echo 'hello wo…"   8 minutes ago
    Exited (0) 8 minutes ago                       cool_jepsen         0B
(virtual 1.22MB)
[root@ubuntu1804 ~]

#显示最新创建的容器(停止的容器也能显示)
[root@ubuntu1804 ~]#docker ps  -l
```

范例: 显示指定状态的容器

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   19 minutes ago
     Exited (137) 11 minutes ago                        nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   19 minutes ago
     Up 2 minutes                80/tcp              nginx1
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
1f3f82995e05        nginx               "nginx -g 'daemon of…"   19 minutes ago
     Up 2 minutes        80/tcp              nginx1
[root@ubuntu1804 ~]#docker ps  -f 'status=exited'
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   19 minutes ago
     Exited (137) 11 minutes ago                        nginx2
[root@ubuntu1804 ~]#
```

1.4.2.2 查看容器内的进程

```
docker top CONTAINER [ps OPTIONS]
```

范例:

```
[root@ubuntu1804 ~]#docker run -d  httpd
db144f1978148242dc20bd0be951628f1c00371b2c69dee53d84469c52995d8f
[root@ubuntu1804 ~]#docker top db144f19
UID        PID    PPID   C    STIME TTY    TIME      CMD
root       9821   9797   3    22:02 ?      00:00:00 httpd -DFOREGROUND
daemon     9872   9821   0    22:02 ?      00:00:00 httpd -DFOREGROUND
```

```
daemon    9873  9821  0   22:02 ?     00:00:00 httpd -DFOREGROUND
daemon    9874  9821  0   22:02 ?     00:00:00 httpd -DFOREGROUND

[root@ubuntu1804 ~]#docker run -d  alpine /bin/sh -c 'i=1;while true;do echo
hello$i;let i++;sleep 1;done'
9997053f9766d4adf709d46161d7ec6739eacafbe8d4721133874b89112ad1a6
[root@ubuntu1804 ~]#docker top 9997
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                10023               9997                3
22:03               ?                   00:00:00            /bin/sh -c i=1;while
true;do echo hello$i;let i++;sleep 1;done
root                10074               10023               0
22:03               ?                   00:00:00            sleep 1
[root@ubuntu1804 ~]#
```

1.4.2.3 查看容器资源使用情况

```
docker stats [OPTIONS] [CONTAINER...]

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

Options:
-a, --all              Show all containers (default shows just running)
    --format string    Pretty-print images using a Go template
    --no-stream        Disable streaming stats and only pull the first result
    --no-trunc         Do not truncate output
```

范例:

```
[root@ubuntu1804 ~]#docker stats 251c7c7cf2aa
CONTAINER ID   NAME   CPU %   MEM USAGE / LIMIT     MEM %   NET I/O     BLOCK I/O
PIDS
251c7c7cf2aa   busy_l0.00%   3.742MiB / 1.924GiB   0.19%   1.29kB / 0B0B / 8.19kB
 2
CONTAINER ID   NAME   CPU %   MEM USAGE / LIMIT     MEM %   NET I/O     BLOCK I/O
PIDS
251c7c7cf2aa   busy_l0.00%   3.742MiB / 1.924GiB   0.19%   1.29kB / 0B0B / 8.19kB
 2

#默认启动elasticsearch会使用较多的内存
[root@ubuntu1804 ~]#docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300
-e "discovery.type=single-node" elasticsearch:7.6.2
[root@ubuntu1804 ~]#curl 10.0.0.100:9200
{
  "name" : "29282e91d773",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "w5lp_XmITliWa2Yc-XwJFw",
  "version" : {
    "number" : "7.6.2",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "ef48eb35cf30adf4db14086e8aabd07ef6fb113f",
    "build_date" : "2020-03-26T06:34:37.794943Z",
    "build_snapshot" : false,
```

```
    "lucene_version" : "8.4.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}


#查看所有容器
[root@ubuntu1804 ~]#docker stats
CONTAINER ID NAME   CPU %   MEM USAGE / LIMIT MEM % NET I/O  BLOCK I/O      PIDS
5e470e7970f6 suspi  0.00%   3.992MiB / 1.924Gi0.20% 656B / 0B9.2MB / 8.19kB    2
829bcebbc9f6 elast  0.58%   1.24GiB / 1.924GiB64.43%2.97kB / 512kB / 729kB    47


#限制内存使用大小
[root@ubuntu1804 ~]#docker run -d --name elasticsearch -p 9200:9200 -p 9300:9300
-e "discovery.type=single-node" -e ES_JAVA_OPTS="-Xms64m -Xmx128m"
elasticsearch:7.6.2

[root@ubuntu1804 ~]#docker stats
CONTAINER ID NAME  CPU % MEM USAGE / LIMIT     MEM %  NET I/O   BLOCK     PIDS
29282e91d773 elasti254.23310.5MiB / 1.924GiB   15.76% 766B / 0B 766kB /46kB 22
```

1.4.2.4 查看容器的详细信息

docker inspect 可以查看docker各种对象的详细信息,包括:镜像,容器,网络等

```
docker inspect [OPTIONS] NAME|ID [NAME|ID...]
Options:
-f, --format string   Format the output using the given Go template
-s, --size            Display total file sizes if the type is container
```

范例:

```
[root@ubuntu1804 ~]#docker inspect 9997
[
    {
        "Id":
"9997053f9766d4adf709d46161d7ec6739eacafbe8d4721133874b89112ad1a6",
        "Created": "2020-02-25T14:03:00.790597711Z",
        "Path": "/bin/sh",
        "Args": [
            "-c",
            "i=1;while true;do echo hello$i;let i++;sleep 1;done"
        ],
        "State": {
            "Status": "running",
            "Running": true,
            "Paused": false,
            "Restarting": false,
            "OOMKilled": false,
            "Dead": false,
            "Pid": 10023,
            "ExitCode": 0,
            "Error": "",
```

```
            "StartedAt": "2020-02-25T14:03:01.407282144Z",
            "FinishedAt": "0001-01-01T00:00:00Z"
        },
        "Image":
"sha256:e7d92cdc71feacf90708cb59182d0df1b911f8ae022d29e8e95d75ca6a99776a",
        "ResolvConfPath":
"/var/lib/docker/containers/9997053f9766d4adf709d46161d7ec6739eacafbe8d472113387
4b89112ad1a6/resolv.conf",
        "HostnamePath":
"/var/lib/docker/containers/9997053f9766d4adf709d46161d7ec6739eacafbe8d472113387
4b89112ad1a6/hostname",
        "HostsPath":
"/var/lib/docker/containers/9997053f9766d4adf709d46161d7ec6739eacafbe8d472113387
4b89112ad1a6/hosts",
        "LogPath":
"/var/lib/docker/containers/9997053f9766d4adf709d46161d7ec6739eacafbe8d472113387
4b89112ad1a6/9997053f9766d4adf709d46161d7ec6739eacafbe8d4721133874b89112ad1a6-
json.log",
        "Name": "/gracious_wescoff",
        "RestartCount": 0,
        "Driver": "overlay2",
        "Platform": "linux",
        "MountLabel": "",
        "ProcessLabel": "",
        "AppArmorProfile": "docker-default",
        "ExecIDs": null,
        "HostConfig": {
            "Binds": null,
            "ContainerIDFile": "",
            "LogConfig": {
                "Type": "json-file",
                "Config": {}
            },
            "NetworkMode": "default",
            "PortBindings": {},
            "RestartPolicy": {
                "Name": "no",
                "MaximumRetryCount": 0
            },
            "AutoRemove": false,
            "VolumeDriver": "",
            "VolumesFrom": null,
            "CapAdd": null,
            "CapDrop": null,
            "Capabilities": null,
            "Dns": [],
            "DnsOptions": [],
            "DnsSearch": [],
            "ExtraHosts": null,
            "GroupAdd": null,
            "IpcMode": "private",
            "Cgroup": "",
            "Links": null,
            "OomScoreAdj": 0,
            "PidMode": "",
            "Privileged": false,
            "PublishAllPorts": false,
            "ReadonlyRootfs": false,
```

```
            "SecurityOpt": null,
            "UTSMode": "",
            "UsernsMode": "",
            "ShmSize": 67108864,
            "Runtime": "runc",
            "ConsoleSize": [
                0,
                0
            ],
            "Isolation": "",
            "CpuShares": 0,
            "Memory": 0,
            "NanoCpus": 0,
            "CgroupParent": "",
            "BlkioWeight": 0,
            "BlkioWeightDevice": [],
            "BlkioDeviceReadBps": null,
            "BlkioDeviceWriteBps": null,
            "BlkioDeviceReadIOps": null,
            "BlkioDeviceWriteIOps": null,
            "CpuPeriod": 0,
            "CpuQuota": 0,
            "CpuRealtimePeriod": 0,
            "CpuRealtimeRuntime": 0,
            "CpusetCpus": "",
            "CpusetMems": "",
            "Devices": [],
            "DeviceCgroupRules": null,
            "DeviceRequests": null,
            "KernelMemory": 0,
            "KernelMemoryTCP": 0,
            "MemoryReservation": 0,
            "MemorySwap": 0,
            "MemorySwappiness": null,
            "OomKillDisable": false,
            "PidsLimit": null,
            "Ulimits": null,
            "CpuCount": 0,
            "CpuPercent": 0,
            "IOMaximumIOps": 0,
            "IOMaximumBandwidth": 0,
            "MaskedPaths": [
                "/proc/asound",
                "/proc/acpi",
                "/proc/kcore",
                "/proc/keys",
                "/proc/latency_stats",
                "/proc/timer_list",
                "/proc/timer_stats",
                "/proc/sched_debug",
                "/proc/scsi",
                "/sys/firmware"
            ],
            "ReadonlyPaths": [
                "/proc/bus",
                "/proc/fs",
                "/proc/irq",
                "/proc/sys",
```

```
                "/proc/sysrq-trigger"
            ]
        },
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/27f7bf9c29a0303e2526bd7fa7d61fcc3a2ef400e6948b5d9487b5
b30f506853-
init/diff:/var/lib/docker/overlay2/cb3f60a3d09e0555d06525b70cc30975a3cb70e23c87e
7e46bd44b6cfffbcda0/diff",
                "MergedDir":
"/var/lib/docker/overlay2/27f7bf9c29a0303e2526bd7fa7d61fcc3a2ef400e6948b5d9487b5
b30f506853/merged",
                "UpperDir":
"/var/lib/docker/overlay2/27f7bf9c29a0303e2526bd7fa7d61fcc3a2ef400e6948b5d9487b5
b30f506853/diff",
                "WorkDir":
"/var/lib/docker/overlay2/27f7bf9c29a0303e2526bd7fa7d61fcc3a2ef400e6948b5d9487b5
b30f506853/work"
            },
            "Name": "overlay2"
        },
        "Mounts": [],
        "Config": {
            "Hostname": "9997053f9766",
            "Domainname": "",
            "User": "",
            "AttachStdin": false,
            "AttachStdout": false,
            "AttachStderr": false,
            "Tty": false,
            "OpenStdin": false,
            "StdinOnce": false,
            "Env": [

 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
            ],
            "Cmd": [
                "/bin/sh",
                "-c",
                "i=1;while true;do echo hello$i;let i++;sleep 1;done"
            ],
            "Image": "alpine",
            "Volumes": null,
            "WorkingDir": "",
            "Entrypoint": null,
            "OnBuild": null,
            "Labels": {}
        },
        "NetworkSettings": {
            "Bridge": "",
            "SandboxID":
"4823297a262eaad3b63778c2fe04a74e38ce401c22404f05f3cf54ab7dc188e2",
            "HairpinMode": false,
            "LinkLocalIPv6Address": "",
            "LinkLocalIPv6PrefixLen": 0,
            "Ports": {},
            "SandboxKey": "/var/run/docker/netns/4823297a262e",
```

```
            "SecondaryIPAddresses": null,
            "SecondaryIPv6Addresses": null,
            "EndpointID":
"019896bd99c9e158ef9f97b1617f1638fee94c7f92c9250e1ac1bfe58c97c911",
            "Gateway": "172.17.0.1",
            "GlobalIPv6Address": "",
            "GlobalIPv6PrefixLen": 0,
            "IPAddress": "172.17.0.3",
            "IPPrefixLen": 16,
            "IPv6Gateway": "",
            "MacAddress": "02:42:ac:11:00:03",
            "Networks": {
                "bridge": {
                    "IPAMConfig": null,
                    "Links": null,
                    "Aliases": null,
                    "NetworkID":
"d407310ec8ff57a6623ae2fa01ba860ab0615135585a00b19f0751561c99ab16",
                    "EndpointID":
"019896bd99c9e158ef9f97b1617f1638fee94c7f92c9250e1ac1bfe58c97c911",
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.3",
                    "IPPrefixLen": 16,
                    "IPv6Gateway": "",
                    "GlobalIPv6Address": "",
                    "GlobalIPv6PrefixLen": 0,
                    "MacAddress": "02:42:ac:11:00:03",
                    "DriverOpts": null
                }
            }
        }
    }
]

#选择性查看
[root@ubuntu1804 ~]#docker inspect -f "{{.Metadata}}" test:v1.0
{2020-07-24 21:56:42.247448035 +0800 CST}

[root@ubuntu1804 ~]#docker inspect -f "{{.Created}}" c1
2020-07-24T13:37:11.006574248Z
[root@ubuntu1804 ~]#docker inspect --format "{{.Created}}" c1
2020-07-24T13:37:11.006574248Z
[root@ubuntu1804 ~]#docker inspect --format="{{.Created}}" c1
2020-07-24T13:37:11.006574248Z
```

1.4.3 删除容器

docker rm 可以删除容器，即使容器正在运行当中，也可以被强制删除掉

格式

```
docker rm [OPTIONS] CONTAINER [CONTAINER...]
docker container rm [OPTIONS] CONTAINER [CONTAINER...]


#选项:
-f, --force      Force the removal of a running container (uses SIGKILL)
```

```
-v, --volumes    Remove the volumes associated with the container

#删除停止的容器
docker container prune [OPTIONS]
Options:
      --filter filter    Provide filter values (e.g. 'until=<timestamp>')
  -f, --force            Do not prompt for confirmation
```

范例:

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
868b33da850c        alpine              "/bin/sh"           2 minutes ago
Up 2 minutes                                    alpine5
3a05bbf66dac        alpine              "/bin/sh"           3 minutes ago
Exited (0) 3 minutes ago                        alpine4
df428caf7128        alpine              "/bin/sh"           3 minutes ago
Up 3 minutes                                    alpine3
6d64f47a83e6        alpine              "/bin/sh"           3 minutes ago
Exited (0) 3 minutes ago                        alpine2
edd2ac2690e6        alpine              "/bin/sh"           4 minutes ago
Exited (0) 4 minutes ago                        alpine1
[root@ubuntu1804 ~]#docker rm 3a05bbf66dac
3a05bbf66dac
[root@ubuntu1804 ~]#docker rm alpine5
Error response from daemon: You cannot remove a running container
868b33da850cfcc7db8b84150fb9c7686b577889f10425bb4c5e17f28cf68a29. Stop the
container before attempting removal or force remove
[root@ubuntu1804 ~]#docker rm -f  alpine5
alpine5
```

范例: 删除所有容器

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
df428caf7128        alpine              "/bin/sh"           4 minutes ago
Up 4 minutes                                    alpine3
6d64f47a83e6        alpine              "/bin/sh"           4 minutes ago
Exited (0) 4 minutes ago                        alpine2
edd2ac2690e6        alpine              "/bin/sh"           5 minutes ago
Exited (0) 5 minutes ago                        alpine1
[root@ubuntu1804 ~]#docker rm -f `docker ps -a -q`
df428caf7128
6d64f47a83e6
edd2ac2690e6
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@ubuntu1804 ~]#docker ps -a -q | xargs docker rm -f
```

范例: 删除指定状态的容器

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                          PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   22 minutes ago
     Exited (137) 14 minutes ago                         nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   22 minutes ago
     Up 4 minutes                    80/tcp              nginx1
[root@ubuntu1804 ~]#docker rm `docker ps -qf status=exited`
dd002f947cbe
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
1f3f82995e05        nginx               "nginx -g 'daemon of…"   22 minutes ago
     Up 4 minutes        80/tcp              nginx1
[root@ubuntu1804 ~]#
```

范例: 删除所有停止的容器

```
[root@ubuntu1804 ~]#docker container prune  -f
Deleted Containers:
37ba6ef81e33102a9cf4547ed10095d2298e29c8d67991b31390d5db8001dbcf
6c674c7ced422c7c29f218118c8b5da734ccebb9da31cdd3f64e7c658d2882a4

Total reclaimed space: 0B
```

1.4.4 容器的启动和停止

格式

```
docker start|stop|restart|pause|unpause 容器ID
```

批量正常启动或关闭所有容器

```
docker start $(docker ps -a -q)
docker stop $(docker ps -a -q)
```

范例:

```
[root@ubuntu1804 ~]#docker run -d  --name nginx1 nginx
8d9342b35589b72c3f7f01f4d9fe8797e974cda8ba28d2bac69ee578aa592ca2
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
8d9342b35589        nginx               "nginx -g 'daemon of…"   5 seconds ago
    Up 4 seconds        80/tcp              nginx1
[root@ubuntu1804 ~]#docker stop nginx1
nginx1
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
8d9342b35589        nginx               "nginx -g 'daemon of…"   15 seconds ago
     Exited (0) 2 seconds ago                        nginx1
```

```
[root@ubuntu1804 ~]#docker start nginx1
nginx1
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
8d9342b35589        nginx               "nginx -g 'daemon of…"   21 seconds ago
     Up 1 second         80/tcp              nginx1
[root@ubuntu1804 ~]#docker restart nginx1
nginx1
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
8d9342b35589        nginx               "nginx -g 'daemon of…"   30 seconds ago
     Up 1 second         80/tcp              nginx1
[root@ubuntu1804 ~]#
```

范例: 启动并进入容器

```
[root@ubuntu1804 ~]#docker run --name=c1 -it ubuntu bash
root@539722b55b76:/# exit
exit
[root@ubuntu1804 ~]#docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                     PORTS               NAMES
539722b55b76        ubuntu              "bash"              4 seconds ago
Exited (0) 1 second ago                        c1

[root@ubuntu1804 ~]#docker start c1
c1
[root@ubuntu1804 ~]#docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                          PORTS               NAMES
539722b55b76        ubuntu              "bash"                   18 seconds ago
     Up 2 seconds                                         c1

[root@ubuntu1804 ~]#docker stop c1
c1
[root@ubuntu1804 ~]#docker ps -l
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
539722b55b76        ubuntu              "bash"                   43 seconds ago
     Exited (0) 1 second ago                          c1

#启动并进入容器
[root@ubuntu1804 ~]#docker start -i c1
root@539722b55b76:/# exit
exit
[root@ubuntu1804 ~]#docker ps -l
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                     PORTS               NAMES
539722b55b76        ubuntu              "bash"              4 minutes ago
Exited (0) 5 seconds ago                       c1
```

范例: 启动和停止所有容器

```
[root@ubuntu1804 ~]#docker rm -f `docker ps -a -q`
b722c745406c
8d9342b35589
[root@ubuntu1804 ~]#docker run -d  --name nginx1 nginx
1f3f82995e052647678fd27bfa27a5b5615efc129270698cbaac3120544d6609
[root@ubuntu1804 ~]#docker run -d  --name nginx2 nginx
dd002f947cbe786ac0e834e06744337556f82d5850f4b16e01f12b9b3759f83e
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   4 seconds ago
    Up 3 seconds        80/tcp              nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   7 seconds ago
    Up 6 seconds        80/tcp              nginx1
[root@ubuntu1804 ~]#docker stop `docker ps -a -q`
dd002f947cbe
1f3f82995e05
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   22 seconds ago
     Exited (0) 2 seconds ago                        nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   25 seconds ago
     Exited (0) 2 seconds ago                        nginx1
[root@ubuntu1804 ~]#docker start `docker ps -a -q`
dd002f947cbe
1f3f82995e05
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   2 minutes ago
    Up 1 second         80/tcp              nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   2 minutes ago
    Up 1 second         80/tcp              nginx1
```

范例: 暂停和恢复容器

```
[root@ubuntu1804 ~]#docker run -d --name n1 nginx
48a8278f5df1d0b0c2c42c01d4e53d335df7e3e866fc7b68563cc2ac545fc07d
[root@ubuntu1804 ~]#docker top n1
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                2104                2076                0
22:51               ?                   00:00:00            nginx: master
process nginx -g daemon off;
systemd+            2168                2104                0
22:51               ?                   00:00:00            nginx: worker
process
[root@ubuntu1804 ~]#ps aux|grep nginx
root       2104  0.3  0.2  10628  5324 ?        Ss   22:51   0:00 nginx: master
process nginx -g daemon off;
systemd+   2168  0.0  0.1  11056  2580 ?        S    22:51   0:00 nginx: worker
process
root       2188  0.0  0.0  14428  1040 pts/0    S+   22:51   0:00 grep --
color=auto nginx

[root@ubuntu1804 ~]#docker pause n1
```

```
n1
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                  PORTS               NAMES
48a8278f5df1        nginx               "/docker-entrypoint.…"   3 minutes ago
    Up 3 minutes (Paused)   80/tcp              n1

[root@ubuntu1804 ~]#ps aux|grep nginx
root       2104  0.0  0.2  10628  5324 ?        Ds   22:51   0:00 nginx: master
process nginx -g daemon off;
systemd+   2168  0.0  0.1  11056  2580 ?        D    22:51   0:00 nginx: worker
process
root       2494  0.0  0.0  14428  1004 pts/0    R+   22:54   0:00 grep --
color=auto nginx

[root@ubuntu1804 ~]#docker unpause n1
n1
[root@ubuntu1804 ~]#ps aux|grep nginx
root       2104  0.0  0.2  10628  5324 ?        Ss   22:51   0:00 nginx: master
process nginx -g daemon off;
systemd+   2168  0.0  0.1  11056  2580 ?        S    22:51   0:00 nginx: worker
process
root       2521  0.0  0.0  14428  1028 pts/0    S+   22:55   0:00 grep --
color=auto nginx
```

范例: 容器的暂停和恢复

```
[root@ubuntu1804 ~]#docker run -itd centos
708bedcbd31be0ecac11aa21a7d15718d440e4bf65e3e6a8670f7391de21f301
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
708bedcbd31b        centos              "/bin/bash"         4 seconds ago
Up 1 second                             blissful_payne
[root@ubuntu1804 ~]#docker pause   blissful_payne
blissful_payne
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                     PORTS               NAMES
708bedcbd31b        centos              "/bin/bash"         19 seconds ago
Up 17 seconds (Paused)                          blissful_payne
[root@ubuntu1804 ~]#docker unpause  blissful_payne
blissful_payne
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
708bedcbd31b        centos              "/bin/bash"         33 seconds ago
Up 31 seconds                           blissful_payne
```

1.4.5 给正在运行的容器发信号

docker kill 可以给容器发信号,默认号SIGKILL,即9信号

格式

```
docker kill [OPTIONS] CONTAINER [CONTAINER...]

#选项:
-s, --signal string   Signal to send to the container (default "KILL")
```

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   2 minutes ago
    Up 1 second         80/tcp              nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   2 minutes ago
    Up 1 second         80/tcp              nginx1
[root@ubuntu1804 ~]#docker  kill nginx1
nginx1
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                          PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   5 minutes ago
    Up 3 minutes                    80/tcp              nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   5 minutes ago
    Exited (137) 2 seconds ago                          nginx1
[root@ubuntu1804 ~]#
```

范例: 关闭所有容器

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
dd002f947cbe        nginx               "nginx -g 'daemon of…"   7 minutes ago
    Up 2 seconds        80/tcp              nginx2
1f3f82995e05        nginx               "nginx -g 'daemon of…"   7 minutes ago
    Up 3 seconds        80/tcp              nginx1

#强制关闭所有运行中的容器
[root@ubuntu1804 ~]#docker  kill `docker ps -a -q`
dd002f947cbe
1f3f82995e05
```

1.4.6 进入正在运行的容器

1.4.6.1 使用attach命令

docker attach 容器名，attach 类似于vnc，操作会在同一个容器的多个会话界面同步显示，所有使用此方式进入容器的操作都是同步显示的，且使用exit退出后容器自动关闭，**不推荐使用**，需要进入到有shell环境的容器

格式:

```
docker attach [OPTIONS] CONTAINER
```

范例:

```
[root@ubuntu1804 ~]#docker run -it centos
[root@94a5c5c69b14 /]# cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core) #ctrl+p+q 退出
[root@94a5c5c69b14 /]# [root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
94a5c5c69b14        centos              "/bin/bash"         14 seconds ago
Up 14 seconds                           unruffled_ellis
[root@ubuntu1804 ~]#docker attach  94a5
[root@94a5c5c69b14 /]#cat /etc/redhat-release

#同时在第二个终端attach到同一个容器，执行命令，可以在前一终端看到显示图面是同步的
[root@ubuntu1804 ~]#docker attach  94a5
[root@94a5c5c69b14 /]#cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)
[root@92a8279611a9 /]# exit  #两个终端都同时退出
exit
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
92a8279611a9        centos              "/bin/bash"         4 minutes ago
Exited (0) 39 seconds ago                       agitated_tesla
```

1.4.6.2 使用exec命令

在运行中的容器启动新进程,可以执行单次命令，以及进入容器

测试环境使用此方式，使用exit退出,但容器还在运行，**此为推荐方式**

格式:

```
docker exec [OPTIONS] CONTAINER COMMAND [ARG...]
常用选项:
-d, --detach               Detached mode: run command in the background
-e, --env list             Set environment variables
-i, --interactive          Keep STDIN open even if not attached
-t, --tty                  Allocate a pseudo-TTY

#常见用法
docker exec -it 容器ID sh|bash
```

范例:

```
[root@ubuntu1804 ~]#docker run -itd centos
24788f69cec65e1f511387c1bae354a66e5b7ae29261e68957bc6dcc4818af6b
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
24788f69cec6        centos              "/bin/bash"         3 seconds ago
Up 1 second                             keen_jennings
#执行一次性命令
[root@ubuntu1804 ~]#docker exec 2478 cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)

#进入容器，执行命令，exit退出但容器不停止
```

```
[root@ubuntu1804 ~]#docker exec  -it 2478  bash
[root@24788f69cec6 /]# cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)
[root@24788f69cec6 /]# exit
exit
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
24788f69cec6        centos              "/bin/bash"         4 minutes ago
Up 4 minutes                            keen_jennings
[root@ubuntu1804 ~]#
```

1.4.6.3 使用nsenter命令

nsenter命令需要通过PID进入到容器内部，且退出后仍然正常运行：不过需要事先使用docker inspect获取到容器的PID， 目前此方式使用较少，此工具来自于util-linux包

```
#安装nsenter命令
yum -y install util-linux #CentOS
apt -y install util-linux #Ubuntu

#获取容器的IP
docker inspect -f "{{.NetworkSettings.IPAddress}}"  容器ID

#获取到某个docker容器的PID，可以通过PID进入到容器内
docker inspect -f "{{.State.Pid}}"  容器ID
nsenter -t PID -m -u -i -n -p
```

范例:

```
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
ba792a7e0747        centos              "/bin/bash"         17 minutes ago
Up 17 minutes                           festive_babbage
[root@ubuntu1804 ~]#docker inspect -f {{.State}}  ba792a7e0747
{running true false false false false 20536 0  2020-01-26T10:44:16.123961829Z
0001-01-01T00:00:00Z <nil>}
[root@ubuntu1804 ~]#docker inspect -f {{.State.Status}}  ba792a7e0747
running
[root@ubuntu1804 ~]#docker inspect -f {{.State.Pid}}  ba792a7e0747
20536
[root@ubuntu1804 ~]#nsenter -t 20536 -m -u -i -n -p
[root@ba792a7e0747 /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3  12024  3172 pts/0    Ss+  10:44   0:00 /bin/bash
root         46  0.0  0.3  12028  3312 ?        S    11:02   0:00 -bash
root         61  0.0  0.3  43960  3352 ?        R+   11:02   0:00 ps aux
[root@ba792a7e0747 /]# exit
logout
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

```
ba792a7e0747        centos              "/bin/bash"         18 minutes ago
Up 18 minutes                           festive_babbage
[root@ubuntu1804 ~]#
```

1.4.6.4 脚本方式

将nsenter命令写入到脚本进行调用，方便进入容器看日志或排错

如下:

```
cat docker-in.sh
#!/bin/bash

docker_in(){
   NAME_ID=$1
   PID=$(docker inspect -f "{{.State.Pid}}" ${NAME_ID})
   nsenter -t ${PID} -m -u -i -n -p
}

docker_in $1
```

范例:

```
[root@ubuntu1804 ~]#vim docker-in.sh
[root@ubuntu1804 ~]#cat docker-in.sh
#!/bin/bash

docker_in(){
   NAME_ID=$1
   PID=$(docker inspect -f "{{.State.Pid}}" ${NAME_ID})
   nsenter -t ${PID} -m -u -i -n -p
}

docker_in $1

[root@ubuntu1804 ~]#chmod +x docker-in.sh
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
ba792a7e0747        centos              "/bin/bash"         20 minutes ago
Up 20 minutes                           festive_babbage
[root@ubuntu1804 ~]#./docker-in.sh  ba792a7e0747
[root@ba792a7e0747 /]# cat /etc/redhat-release
CentOS Linux release 8.1.1911 (Core)
[root@ba792a7e0747 /]# exit
logout
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
ba792a7e0747        centos              "/bin/bash"         23 minutes ago
Up 23 minutes                           festive_babbage
```

1.4.7 暴露所有容器端口

容器启动后,默认处于预定义的NAT网络中,所以外部网络的主机无法直接访问容器中网络服务

docker run -P 可以将事先容器预定义的所有端口映射宿主机的网卡的随机端口，默认从32768开始

使用随机端口 时,当停止容器后再启动可能会导致端口发生变化

```
-P , --publish-all= true | false默认为false

#示例:
docker run -P docker.io/nginx  #映射容器所有暴露端口至随机本地端口
```

docker port 可以查看容器的端口映射关系

格式

```
docker port CONTAINER [PRIVATE_PORT[/PROTO]]
```

范例:

```
[root@centos7 ~]#docker port nginx-c1
443/tcp -> 0.0.0.0:8443
53/udp -> 0.0.0.0:8053
80/tcp -> 0.0.0.0:8080
[root@centos7 ~]#docker port nginx-c1 53/udp
0.0.0.0:8053
```

范例:

```
[root@centos7 ~]#docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
8ec398bc0356: Pull complete
a53c868fbde7: Pull complete
79daf9dd140d: Pull complete
Digest: sha256:70821e443be75ea38bdf52a974fd2271babd5875b2b1964f05025981c75a6717
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
[root@centos7 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@centos7 ~]#ss -ntl
State      Recv-Q Send-Q          Local Address:Port
Peer Address:Port
LISTEN     0      128                         *:22
       *:*
LISTEN     0      100                 127.0.0.1:25
       *:*
LISTEN     0      128                        :::22
      :::*
LISTEN     0      100                       ::1:25
      :::*

#前台启动的会话窗口无法进行其他操作，除非退出，但是退出后容器也会退出
[root@centos7 ~]#docker run -P nginx
172.17.0.1 - - [26/Jan/2020:06:44:56 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.29.0" "-"
```

```
#另开一个窗口执行下面命令
[root@centos7 ~]#ss -ntl
State      Recv-Q Send-Q          Local Address:Port
Peer Address:Port
LISTEN     0      128                         *:22
       *:*
LISTEN     0      100                 127.0.0.1:25
       *:*
LISTEN     0      128                        :::22
      :::*
LISTEN     0      100                       ::1:25
      :::*
LISTEN     0      128                        :::32768
       :::*
[root@centos7 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                  NAMES
78086069642b        nginx               "nginx -g 'daemon of…"   23 seconds ago
     Up 21 seconds       0.0.0.0:32768->80/tcp   gallant_austin
[root@centos7 ~]#curl 127.0.0.1:32768
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
[root@centos7 ~]#

#自动生成Iptables规则
[root@centos7 ~]#iptables -vnL -t nat
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

   19  1012 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0
          ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
```

```
 pkts bytes target     prot opt in     out     source               destination


Chain OUTPUT (policy ACCEPT 1 packets, 76 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8
          ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 1 packets, 76 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0

    0     0 MASQUERADE  tcp  --  *      *       172.17.0.2           172.17.0.2
          tcp dpt:80
    0     0 MASQUERADE  tcp  --  *      *       172.17.0.4           172.17.0.4
          tcp dpt:80

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination

    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0

    0     0 DNAT       tcp  --  !docker0 *      0.0.0.0/0            10.0.0.7
          tcp dpt:32768 to:172.17.0.2:80

#回到之前的会话窗口，同时按两个键 ctrl+c 退出容器
[root@centos7 ~]#docker run -P nginx
172.17.0.1 - - [26/Jan/2020:06:44:56 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.29.0" "-"
^C[root@centos7 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
78086069642b        nginx               "nginx -g 'daemon of…"   3 minutes ago
    Exited (0) 5 seconds ago                        gallant_austin
[root@centos7 ~]#
```

端口映射的本质就是利用NAT技术实现的

范例: 端口映射和iptables

```
#端口映射前的iptables规则
[root@ubuntu1804 ~]#iptables -S
-P INPUT ACCEPT
-P FORWARD DROP
-P OUTPUT ACCEPT
-N DOCKER
-N DOCKER-ISOLATION-STAGE-1
-N DOCKER-ISOLATION-STAGE-2
-N DOCKER-USER
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
```

```
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
[root@ubuntu1804 ~]#iptables -S -t nat
-P PREROUTING ACCEPT
-P INPUT ACCEPT
-P OUTPUT ACCEPT
-P POSTROUTING ACCEPT
-N DOCKER
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A DOCKER -i docker0 -j RETURN

[root@ubuntu1804 ~]#iptables -S > pre.filter
[root@ubuntu1804 ~]#iptables -S -t nat > pre.nat

#实现端口映射
[root@ubuntu1804 ~]#docker run -d -P --name nginx1 nginx
286a3dedf159fbf0a4b895741a9d95562c87b44782ea85c8d172474da8860c36
[root@ubuntu1804 ~]#docker exec -it nginx1 hostname -i
172.17.0.2
[root@ubuntu1804 ~]#docker port nginx1
80/tcp -> 0.0.0.0:32769


#端口映射后的iptables规则
[root@ubuntu1804 ~]#iptables -S
-P INPUT ACCEPT
-P FORWARD DROP
-P OUTPUT ACCEPT
-N DOCKER
-N DOCKER-ISOLATION-STAGE-1
-N DOCKER-ISOLATION-STAGE-2
-N DOCKER-USER
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER -d 172.17.0.2/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 80 -j
ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN

[root@ubuntu1804 ~]#iptables -S -t nat
-P PREROUTING ACCEPT
-P INPUT ACCEPT
-P OUTPUT ACCEPT
-P POSTROUTING ACCEPT
-N DOCKER
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
```

```
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A POSTROUTING -s 172.17.0.2/32 -d 172.17.0.2/32 -p tcp -m tcp --dport 80 -j
MASQUERADE
-A DOCKER -i docker0 -j RETURN
-A DOCKER ! -i docker0 -p tcp -m tcp --dport 32769 -j DNAT --to-destination
172.17.0.2:80

#对比端口映射前后的变化
[root@ubuntu1804 ~]#iptables -S > post.filter
[root@ubuntu1804 ~]#iptables -S -t nat > post.nat

[root@ubuntu1804 ~]#diff pre.filter post.filter
13a14
> -A DOCKER -d 172.17.0.2/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 80 -j
ACCEPT

[root@ubuntu1804 ~]#diff pre.nat post.nat
8a9
> -A POSTROUTING -s 172.17.0.2/32 -d 172.17.0.2/32 -p tcp -m tcp --dport 80 -j
MASQUERADE
9a11
> -A DOCKER ! -i docker0 -p tcp -m tcp --dport 32769 -j DNAT --to-destination
172.17.0.2:80


#本地和选程都可以访问
[root@ubuntu1804 ~]#curl 127.0.0.1:32769
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

[root@centos8 ~]#curl 10.0.0.100:32769
<!DOCTYPE html>
<html>
<head>
```

```
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>


#利用iptables 阻止同一个宿主机的其它容器CentOS8的访问
[root@ubuntu1804 ~]#iptables -I DOCKER -s 10.0.0.8 -d 172.17.0.2 -p tcp --dport
80 -j REJECT
[root@ubuntu1804 ~]#iptables -S
-P INPUT ACCEPT
-P FORWARD DROP
-P OUTPUT ACCEPT
-N DOCKER
-N DOCKER-ISOLATION-STAGE-1
-N DOCKER-ISOLATION-STAGE-2
-N DOCKER-USER
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER -s 10.0.0.8/32 -d 172.17.0.2/32 -p tcp -m tcp --dport 80 -j REJECT --
reject-with icmp-port-unreachable
-A DOCKER -d 172.17.0.2/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 80 -j
ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN


#测试访问
[root@centos8 ~]#curl 10.0.0.100:32769
curl: (7) Failed to connect to 10.0.0.100 port 32769: Connection refused

[root@centos7 ~]#curl -I 10.0.0.100:32769
HTTP/1.1 200 OK
```

```
Server: nginx/1.19.1
Date: Thu, 23 Jul 2020 05:14:01 GMT
Content-Type: text/html
Content-Length: 612
Last-Modified: Tue, 07 Jul 2020 15:52:25 GMT
Connection: keep-alive
ETag: "5f049a39-264"
Accept-Ranges: bytes
```

1.4.8 指定端口映射

docker run -p 可以将容器的预定义的指定端口映射到宿主机的相应端口

注意：多个容器映射到宿主机的端口不能冲突，但容器内使用的端口可以相同

方式1：容器80端口映射宿主机本地随机端口

```
docker run  -p 80 --name nginx-test-port1 nginx
```

方式2：容器80端口映射到宿主机本地端口81

```
docker run  -p 81:80 --name nginx-test-port2 nginx
```

方式3：宿主机本地IP:宿主机本地端口:容器端口

```
docker run  -p 10.0.0.100:82:80 --name nginx-test-port3 docker.io/nginx
```

方式4：宿主机本地IP:宿主机本地随机端口:容器端口，默认从32768开始

```
docker run -p 10.0.0.100::80 --name nginx-test-port4 docker.io/nginx
```

方式5：宿主机本机ip:宿主机本地端口:容器端口/协议，默认为tcp协议

```
docker run  -p 10.0.0.100:83:80/udp --name nginx-test-port5 docker.io/nginx
```

方式6：一次性映射多个端口+协议

```
docker run  -p 8080:80/tcp -p 8443:443/tcp -p 53:53/udp --name nginx-test-port6
nginx
```

范例:

```
[root@centos7 ~]#docker run -d  -p 8080:80 -p 8443:443 -p 8053:53/udp nginx
a902b177bb7135ad8a8a179dbf8ce02dcc4806a1136475e59c2310833d7434ab
[root@centos7 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS
            NAMES
a902b177bb71        nginx               "nginx -g 'daemon of…"   5 seconds ago
    Up 4 seconds        0.0.0.0:8053->53/udp, 0.0.0.0:8080->80/tcp,
0.0.0.0:8443->443/tcp   affectionate_aryabhata
```

```
[root@centos7 ~]#ss -ntpul
Netid  State      Recv-Q Send-Q       Local Address:Port
Peer Address:Port
udp    UNCONN     0      0               127.0.0.1:323
        *:*                   users:(("chronyd",pid=6292,fd=1))
udp    UNCONN     0      0                    ::1:323
        :::*                  users:(("chronyd",pid=6292,fd=2))
udp    UNCONN     0      0                    :::8053
       :::*                 users:(("docker-proxy",pid=32671,fd=4))
tcp    LISTEN     0      128                   *:22
        *:*                  users:(("sshd",pid=6623,fd=3))
tcp    LISTEN     0      100             127.0.0.1:25
        *:*                  users:(("master",pid=6748,fd=13))
tcp    LISTEN     0      128                  :::8080
       :::*                 users:(("docker-proxy",pid=32659,fd=4))
tcp    LISTEN     0      128                  :::22
       :::*                 users:(("sshd",pid=6623,fd=4))
tcp    LISTEN     0      100                  ::1:25
       :::*                 users:(("master",pid=6748,fd=14))
tcp    LISTEN     0      128                  :::8443
       :::*                 users:(("docker-proxy",pid=32646,fd=4))
[root@centos7 ~]#iptables -vnL -t nat
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

   19  1012 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0
         ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8
         ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0

    0     0 MASQUERADE  tcp  --  *      *       172.17.0.2           172.17.0.2
         tcp dpt:443
    0     0 MASQUERADE  tcp  --  *      *       172.17.0.2           172.17.0.2
         tcp dpt:80
    0     0 MASQUERADE  udp  --  *      *       172.17.0.2           172.17.0.2
         udp dpt:53

Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination

    0     0 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0

    0     0 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0
          tcp dpt:8443 to:172.17.0.2:443
```

```
    0     0 DNAT       tcp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0
           tcp dpt:8080 to:172.17.0.2:80
    0     0 DNAT       udp  --  !docker0 *       0.0.0.0/0            0.0.0.0/0
           udp dpt:8053 to:172.17.0.2:53

#杀死nginx进程，nginx将关闭，相应端口也会关闭
[root@centos7 ~]#kill  <NGINXPID>
```

实战案例: 修改已经创建的容器的端口映射关系

```
[root@ubuntu1804 ~]#docker run -d -p 80:80 --name nginx01 nginx
dc5d7c1029e582a3e05890fd18565367482232c151bba09ca27e195d39dbcc24

[root@ubuntu1804 ~]#docker port nginx01
80/tcp -> 0.0.0.0:80

[root@ubuntu1804 ~]#lsof -i:80
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
docker-pr 2364 root    4u  IPv6  35929      0t0  TCP *:http (LISTEN)


[root@ubuntu1804 ~]#ls
/var/lib/docker/containers/dc5d7c1029e582a3e05890fd18565367482232c151bba09ca27e1
95d39dbcc24/
checkpoints
hostconfig.json  mounts
config.v2.json
hostname         resolv.conf
dc5d7c1029e582a3e05890fd18565367482232c151bba09ca27e195d39dbcc24-json.log  hosts
           resolv.conf.hash

[root@ubuntu1804 ~]#systemctl stop docker
[root@ubuntu1804 ~]#vim
/var/lib/docker/containers/dc5d7c1029e582a3e05890fd18565367482232c151bba09ca27e1
95d39dbcc24/hostconfig.json
"PortBindings":{"80/tcp":[{"HostIp":"","HostPort":"80"}]}
#PortBindings后80/tcp对应的是容器内部的80端口，HostPort对应的是映射到宿主机的端口80 修改此
处为8000
"PortBindings":{"80/tcp":[{"HostIp":"","HostPort":"8000"}]}

[root@ubuntu1804 ~]#systemctl start docker
[root@ubuntu1804 ~]#docker start nginx01
[root@ubuntu1804 ~]#docker port nginx01
80/tcp -> 0.0.0.0:8000
```

1.4.9 查看容器的日志

docker logs 可以查看容器中运行的进程在控制台输出的日志信息

格式

```
docker logs [OPTIONS] CONTAINER

选项:
--details        Show extra details provided to logs
-f, --follow     Follow log output
--since string   Show logs since timestamp (e.g. 2013-01-02T13:23:37) or
relative (e.g. 42m for   42 minutes)
--tail string    Number of lines to show from the end of the logs (default
"all")
-t, --timestamps     Show timestamps
--until string   Show logs before a timestamp (e.g. 2013-01-02T13:23:37) or
relative (e.g. 42m     for 42 minutes)
```

范例: 查看容器日志

```
[root@ubuntu1804 ~]#docker run  alpine /bin/sh -c 'i=1;while true;do echo
hello$i;let i++;sleep 2;done'
Unable to find image 'alpine:latest' locally
latest: Pulling from library/alpine
188c0c94c7c5: Pull complete
Digest: sha256:c0e9560cda118f9ec63ddefb4a173a2b2a0347082d7dff7dc14272e7841a5b5a
Status: Downloaded newer image for alpine:latest
hello1
hello2
hello3
hello4
hello5
^C[root@ubuntu1804 ~]#

[root@ubuntu1804 ~]#docker run -d  alpine /bin/sh -c 'i=1;while true;do echo
hello$i;let i++;sleep 2;done'
512622b006c05673630eb04f081f8475400b1cda786b0a8a5d1c1c2fd6dc56a7
[root@ubuntu1804 ~]#docker logs  5126
hello1
hello2
hello3
hello4
hello5
hello6
[root@ubuntu1804 ~]#docker logs --tail 3 5126
hello8
hello9
hello10


#显示时间
[root@ubuntu1804 ~]#docker logs --tail 0 -t 5126
2020-02-25T13:30:07.321390731Z hello17

#持续跟踪
[root@ubuntu1804 ~]#docker logs -f 5126
hello1
hello2
hello3
hello4
```

```
hello5
hello6
hello7
hello8
hello9
hello10
hello11
hello12
hello13
hello14
hello15
hello16
hello17
hello18
.....
```

范例: 查看httpd服务日志

```
[root@ubuntu1804 ~]#docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
bb79b6b2107f: Pull complete
26694ef5449a: Pull complete
7b85101950dd: Pull complete
da919f2696f2: Pull complete
3ae86ea9f1b9: Pull complete
Digest: sha256:b82fb56847fcbcca9f8f162a3232acb4a302af96b1b2af1c4c3ac45ef0c9b968
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
[root@ubuntu1804 ~]#docker run -d -p 80:80 --name web1  httpd
9f55b2216f0d65fe010166a78f07f45a47379bb0efa38c4f81f2034a7401907b
[root@ubuntu1804 ~]#docker logs web1
AH00558: httpd: Could not reliably determine the server's fully qualified domain
name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress this
message
AH00558: httpd: Could not reliably determine the server's fully qualified domain
name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress this
message
[Mon Nov 16 01:07:53.780025 2020] [mpm_event:notice] [pid 1:tid 140363582039168]
AH00489: Apache/2.4.46 (Unix) configured -- resuming normal operations
[Mon Nov 16 01:07:53.780218 2020] [core:notice] [pid 1:tid 140363582039168]
AH00094: Command line: 'httpd -D FOREGROUND'
[root@ubuntu1804 ~]#docker logs -f web1
AH00558: httpd: Could not reliably determine the server's fully qualified domain
name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress this
message
AH00558: httpd: Could not reliably determine the server's fully qualified domain
name, using 172.17.0.3. Set the 'ServerName' directive globally to suppress this
message
[Mon Nov 16 01:07:53.780025 2020] [mpm_event:notice] [pid 1:tid 140363582039168]
AH00489: Apache/2.4.46 (Unix) configured -- resuming normal operations
[Mon Nov 16 01:07:53.780218 2020] [core:notice] [pid 1:tid 140363582039168]
AH00094: Command line: 'httpd -D FOREGROUND'
10.0.0.8 - - [16/Nov/2020:01:08:23 +0000] "GET / HTTP/1.1" 200 45
```

范例: 查看nginx服务访问日志

```
#查看一次
[root@centos7 ~]#docker logs nginx-test-port1
10.0.0.1 - - [26/Jan/2020:07:17:16 +0000] "GET /favicon.ico HTTP/1.1" 404 555 "-
" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/63.0.3239.132 Safari/537.36" "-"
2020/01/26 07:17:16 [error] 6#6: *1 open() "/usr/share/nginx/html/favicon.ico"
failed (2: No such file or directory), client: 10.0.0.1, server: localhost,
request: "GET /favicon.ico HTTP/1.1", host: "10.0.0.7:32769"
10.0.0.1 - - [26/Jan/2020:07:17:17 +0000] "GET / HTTP/1.1" 200 612 "-"
"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko" "-"

#持续查看
[root@centos7 ~]#docker logs -f nginx-test-port1
10.0.0.1 - - [26/Jan/2020:07:17:16 +0000] "GET /favicon.ico HTTP/1.1" 404 555 "-
" "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/63.0.3239.132 Safari/537.36" "-"
2020/01/26 07:17:16 [error] 6#6: *1 open() "/usr/share/nginx/html/favicon.ico"
failed (2: No such file or directory), client: 10.0.0.1, server: localhost,
request: "GET /favicon.ico HTTP/1.1", host: "10.0.0.7:32769"
10.0.0.1 - - [26/Jan/2020:07:17:17 +0000] "GET / HTTP/1.1" 200 612 "-"
"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko" "-"
```

1.4.10 传递运行命令

容器需要有一个前台运行的进程才能保持容器的运行，通过传递运行参数是一种方式，另外也可以在构建镜像的时候指定容器启动时运行的前台命令

容器里的PID为1的守护进程的实现方式

- 服务类：如：Nginx，Tomcat，Apache，但服务不能停
- 命令类：如：tail -f /etc/hosts，主要用于测试环境，注意：不要tail -f <服务访问日志> 会产生不必要的磁盘IO

范例：

```
[root@ubuntu1804 ~]#docker run -d alpine
6ec8989f572a41d2d0c7d2cb12ac31de14de38af0a01af405f81dbfcf534b716
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS                      PORTS               NAMES
6ec8989f572a        alpine              "/bin/sh"           3 seconds ago
Exited (0) 2 seconds ago                        gallant_albattani
[root@ubuntu1804 ~]#docker run -d alpine tail -f /etc/hosts
2bc9fa486769a2335f7e9aa67c7d3e7f091ba9b76d38dff868b8fd648251b576
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
  STATUS                      PORTS               NAMES
2bc9fa486769        alpine              "tail -f /etc/hosts"     3 seconds ago
  Up 2 seconds                                    stupefied_keldysh
6ec8989f572a        alpine              "/bin/sh"                23 seconds ago
   Exited (0) 22 seconds ago                       gallant_albattani

[root@ubuntu1804 ~]#docker exec -it 2bc9fa486769 sh
/ # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 tail -f /etc/hosts
   11 root      0:00 sh
```

```
   17 root      0:00 ps aux
/ # exit
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
   STATUS              PORTS               NAMES
1e30dfc283da        alpine              "tail -f /etc/hosts"     About a minute
ago   Up About a minute                       kind_mcclintock
```

1.4.11 容器内部的hosts文件

容器会自动将容器的ID加入自己的/etc/hosts文件中，并解析成容器的IP

```
[root@ubuntu1804 ~]#docker run -it centos /bin/bash
[root@598262a87c46 /]# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  598262a87c46  #默认会将实例的ID 添加到自己的hosts文件
[root@598262a87c46 /]# hostname
598262a87c46
[root@598262a87c46 /]# ping 598262a87c46
PING 598262a87c46 (172.17.0.2) 56(84) bytes of data.
64 bytes from 598262a87c46 (172.17.0.2): icmp_seq=1 ttl=64 time=0.118 ms
64 bytes from 598262a87c46 (172.17.0.2): icmp_seq=2 ttl=64 time=0.085 ms
^C
--- 598262a87c46 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 3ms
rtt min/avg/max/mdev = 0.085/0.101/0.118/0.019 ms

#在另一个会话执行
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
598262a87c46        centos              "/bin/bash"         14 seconds ago
Up 12 seconds                           optimistic_wiles
```

范例: 修改容器的 hosts文件

```
[root@ubuntu1804 ~]#docker run -it --rm --add-host www.wangxiaochun.com:6.6.6.6
--add-host www.magedu.org:8.8.8.8   busybox
/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
6.6.6.6 www.wangxiaochun.com
8.8.8.8 www.magedu.org
172.17.0.2  449bf0468efd
```

1.4.12 指定容器DNS

容器的dns服务器，默认采用宿主机的dns 地址，可以用下面方式指定其它的DNS地址

- 将dns地址配置在宿主机
- 在容器启动时加选项 `--dns=x.x.x.x`
- 在/etc/docker/daemon.json 文件中指定

范例: 容器的DNS默认从宿主机的DNS获取

```
[root@ubuntu1804 ~]#systemd-resolve --status|grep -A1 -i "DNS Servers"
         DNS Servers: 180.76.76.76
                      223.6.6.6
[root@ubuntu1804 ~]#docker run -it --rm   centos bash
[root@1364f98c4227 /]# cat /etc/resolv.conf
nameserver 180.76.76.76
nameserver 223.6.6.6
search magedu.com magedu.org
[root@1364f98c4227 /]# exit
exit
[root@ubuntu1804 ~]#
```

范例: 指定DNS地址

```
[root@ubuntu1804 ~]#docker run -it --rm --dns 1.1.1.1 --dns 8.8.8.8  centos bash
[root@ef9cacc74b58 /]# cat /etc/resolv.conf
search magedu.com magedu.org
nameserver 1.1.1.1
nameserver 8.8.8.8
[root@ef9cacc74b58 /]# exit
exit
[root@ubuntu1804 ~]#
```

范例: 指定domain名

```
[root@ubuntu1804 ~]#docker run -it --rm --dns 1.1.1.1 --dns 8.8.8.8 --dns-search
a.com --dns-search b.com  busybox
/ # cat /etc/resolv.conf
search a.com b.com
nameserver 1.1.1.1
nameserver 8.8.8.8
/ #
```

范例: 配置文件指定DNS和搜索domain名

```
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
[root@ubuntu1804 ~]#cat /etc/docker/daemon.json
{
  "storage-driver": "overlay2",
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"],
   "dns" : [  "114.114.114.114", "119.29.29.29"],
```

```
    "dns-search": [ "magedu.com", "magedu.org"]
}
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#docker run -it --rm   centos bash
[root@7a2d8fac6f6b /]# cat /etc/resolv.conf
search magedu.com magedu.org
nameserver 114.114.114.114
nameserver 119.29.29.29
[root@7a2d8fac6f6b /]# exit
exit

#用--dns指定优先级更高
[root@ubuntu1804 ~]#docker run -it --rm  --dns 8.8.8.8 --dns 8.8.4.4 centos bash
[root@80ffe3547b87 /]# cat /etc/resolv.conf
search magedu.com magedu.org
nameserver 8.8.8.8
nameserver 8.8.4.4
[root@80ffe3547b87 /]# exit
exit
```

1.4.13 容器内和宿主机之间复制文件

```
docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-
docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH
Options:
  -a, --archive       Archive mode (copy all uid/gid information)
  -L, --follow-link   Always follow symbol link in SRC_PATH
```

范例:

```
[root@ubuntu1804 ~]#docker run -itd centos
1311fe67e6708dac71c01f7d1752a6dcb5e85c2f1fa4ac2efcef9edfe4fb6bb5
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
1311fe67e670        centos              "/bin/bash"         2 seconds ago
Up 2 seconds                            elegant_khorana

#将容器内文件复制到宿主机
[root@ubuntu1804 ~]#docker cp -a 1311:/etc/centos-release .
[root@ubuntu1804 ~]#cat centos-release
CentOS Linux release 8.1.1911 (Core)

#将宿主机文件复制到容器内
[root@ubuntu1804 ~]#docker cp /etc/issue  1311:/root/
[root@ubuntu1804 ~]#docker exec 1311 cat /root/issue
Ubuntu 18.04.1 LTS \n \l

[root@ubuntu1804 ~]#
```

1.4.14 使用 systemd 控制容器运行

```
[root@ubuntu1804 ~]#cat  /lib/systemd/system/hello.service
[Unit]
Description=Hello World
```

```
After=docker.service
Requires=docker.service
[Service]
TimeoutStartSec=0
ExecStartPre=-/usr/bin/docker kill busybox-hello
ExecStartPre=-/usr/bin/docker rm busybox-hello
ExecStartPre=/usr/bin/docker pull busybox
ExecStart=/usr/bin/docker run --name busybox-hello busybox /bin/sh -c "while
true; do echo Hello World; sleep 1; done"
ExecStop=/usr/bin/docker kill busybox-hello
[Install]
WantedBy=multi-user.target

[root@ubuntu1804 ~]#systemctl  daemon-reload
[root@ubuntu1804 ~]#systemctl  enable --now hello.service
```

1.4.15 传递环境变量

有些容器运行时，需要传递变量，可以使用 -e <参数> 或 --env-file <参数文件> 实现

范例: 传递变量创建MySQL

变量参考链接: https://hub.docker.com/_/mysql

```
#MySQL容器运行时需要指定root的口令
[root@ubuntu1804 ~]#docker run  --name mysql01 mysql:5.7.32
2020-11-16 01:43:13+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL
Server 5.7.32-1debian10 started.
2020-11-16 01:43:13+00:00 [Note] [Entrypoint]: Switching to dedicated user
'mysql'
2020-11-16 01:43:13+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL
Server 5.7.32-1debian10 started.
2020-11-16 01:43:13+00:00 [ERROR] [Entrypoint]: Database is uninitialized and
password option is not specified
    You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD
and MYSQL_RANDOM_ROOT_PASSWORD

[root@ubuntu1804 ~]#docker run --name mysql-test1 -v /data/mysql:/var/lib/mysql
-e MYSQL_ROOT_PASSWORD=123456 -e MYSQL_DATABASE=wordpress -e MYSQL_USER=wpuser -
e MYSQL_PASSWORD=123456 -d -p 3306:3306 mysql:5.7.30

[root@ubuntu1804 ~]#docker run --name mysql-test2 -v
/root/mysql/:/etc/mysql/conf.d -v /data/mysql2:/var/lib/mysql --env-
file=env.list -d -p 3307:3306 mysql:5.7.30

[root@ubuntu1804 ~]#cat mysql/mysql-test.cnf
[mysqld]
server-id=100
log-bin=mysql-bin

[root@ubuntu1804 ~]#cat env.list
MYSQL_ROOT_PASSWORD=123456
MYSQL_DATABASE=wordpress
MYSQL_USER=wpuser
MYSQL_PASSWORD=wppass
```

1.4.16 podman 管理容器

范例: podman管理容器

```
#安装httpd
[root@centos8 ~]#podman pull httpd
[root@centos8 ~]#podman run -d --name web -p 80:80 httpd
[root@centos8 ~]#curl 127.0.0.1
<html><body><h1>It works!</h1></body></html>
[root@centos8 ~]#podman exec -it web /bin/sh
# ls
bin  build  cgi-bin  conf  error  htdocs  icons  include  logs  modules
# cd htdocs
# cat index.html
<html><body><h1>It works!</h1></body></html>
# echo welcome to magedu > index.html
# exit
[root@centos8 ~]#curl 127.0.0.1
welcome to magedu

#安装nginx
[root@centos8 ~]#podman run -dt -p 80:80 --name nginx -v /data:/data -e
NGINX_VERSION=1.16 nginx:1.16.0
[root@centos8 ~]#podman stop nginx

#将容器设为开机启动
[root@centos8 ~]#vim  /lib/systemd/system/nginx_podman.service
[root@centos8 ~]#cat   /lib/systemd/system/nginx_podman.service
[Unit]
Description=Podman Nginx Service
After=network.target
After=network-online.target

[Service]
Type=simple
ExecStart=/usr/bin/podman start -a nginx  # -a, --attach Attach container's
STDOUT and STDERR
ExecStop=/usr/bin/podman stop -t 10 nginx
Restart=always

[Install]
WantedBy=multi-user.target

[root@centos8 ~]#systemctl daemon-reload
[root@centos8 ~]#systemctl enable --now nginx_podman.service
[root@centos8 ~]#curl 127.0.0.1

#podman 查看日志
[root@centos8 ~]#podman logs   nginx
10.0.0.8 - - [24/Feb/2020:14:19:45 +0000] "GET / HTTP/1.1" 200 612 "-"
"curl/7.61.1" "-"
10.0.0.1 - - [24/Feb/2020:14:25:54 +0000] "GET / HTTP/1.1" 200 612 "-"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/79.0.3945.117 Safari/537.36" "-"
[root@centos8 ~]#podman  port nginx
```

```
80/tcp -> 0.0.0.0:80

[root@centos8 ~]#ss -ntl
State         Recv-Q         Send-Q    Local Address:Port     Peer Address:Port

LISTEN        0              128             0.0.0.0:80            0.0.0.0:*

LISTEN        0              128             0.0.0.0:22            0.0.0.0:*

LISTEN        0              128                [::]:22               [::]:*

[root@centos8 ~]#systemctl stop nginx_podman.service
[root@centos8 ~]#ss -ntl
State         Recv-Q         Send-Q      Local Address:Port      Peer Address:Port

LISTEN        0              128               0.0.0.0:22             0.0.0.0:*

LISTEN        0              128                  [::]:22                [::]:*

#查看进程信息
[root@centos8 ~]#systemctl start nginx_podman.service
[root@centos8 ~]#pstree -p
systemd(1)─┬─NetworkManager(664)─┬─{NetworkManager}(698)
           |                     └─{NetworkManager}(699)
           ├─VGAuthService(659)
           ├─agetty(766)
           ├─atd(763)
           ├─auditd(626)───{auditd}(627)
           ├─automount(789)─┬─{automount}(796)
           |                ├─{automount}(797)
           |                ├─{automount}(805)
           |                └─{automount}(822)
           ├─conmon(2378)─┬─nginx(2388)───nginx(2401)
           |              └─{conmon}(2380)
           ├─crond(762)
           ├─dbus-daemon(661)
           ├─podman(2308)─┬─{podman}(2309)
           |              ├─{podman}(2310)
           |              ├─{podman}(2311)
           |              ├─{podman}(2312)
           |              ├─{podman}(2313)
           |              ├─{podman}(2316)
           |              ├─{podman}(2321)
           |              ├─{podman}(2326)
           |              └─{podman}(2399)
           ├─polkitd(668)─┬─{polkitd}(697)
           |              ├─{polkitd}(700)
           |              ├─{polkitd}(703)
           |              ├─{polkitd}(704)
           |              └─{polkitd}(754)
           ├─rngd(667)───{rngd}(677)
           ├─rsyslogd(788)─┬─{rsyslogd}(795)
           |               └─{rsyslogd}(798)
           ├─sshd(711)─┬─sshd(1361)───sshd(1375)───bash(1377)
           |           └─sshd(1362)───sshd(1376)───bash(1380)───pstree(2504)
           ├─sssd(658)─┬─sssd_be(730)
           |           └─sssd_nss(758)
           ├─systemd(1366)───(sd-pam)(1369)
```

```
           ├─systemd-journal(553)
           ├─systemd-logind(760)
           ├─systemd-udevd(586)
           ├─tuned(702)─┬─{tuned}(1073)
           |            ├─{tuned}(1076)
           |            └─{tuned}(1088)
           └─vmtoolsd(660)

#nginx进程杀死后还会自动启动
[root@centos8 ~]#kill 2388
[root@centos8 ~]#ps aux|grep nginx
root       2939  1.1  8.4 908244 69240 ?        Ssl  22:45   0:00
/usr/bin/podman start -a nginx
root       3009  0.0  0.3 142832  2652 ?        Ssl  22:45   0:00
/usr/libexec/podman/conmon -s -c
9198c59a8a3db50801c52ceaa39521b4381ac46ab7c16907130244d2a328e823 -u
9198c59a8a3db50801c52ceaa39521b4381ac46ab7c16907130244d2a328e823 -n nginx -r
/usr/bin/runc -b /var/lib/containers/storage/overlay-
containers/9198c59a8a3db50801c52ceaa39521b4381ac46ab7c16907130244d2a328e823/user
data -p /var/run/containers/storage/overlay-
containers/9198c59a8a3db50801c52ceaa39521b4381ac46ab7c16907130244d2a328e823/user
data/pidfile --exit-dir /var/run/libpod/exits --exit-command /usr/bin/podman --
exit-command-arg --root --exit-command-arg /var/lib/containers/storage --exit-
command-arg --runroot --exit-command-arg /var/run/containers/storage --exit-
command-arg --log-level --exit-command-arg error --exit-command-arg --cgroup-
manager --exit-command-arg systemd --exit-command-arg --tmpdir --exit-command-
arg /var/run/libpod --exit-command-arg --runtime --exit-command-arg runc --exit-
command-arg --storage-driver --exit-command-arg overlay --exit-command-arg --
events-backend --exit-command-arg journald --exit-command-arg container --exit-
command-arg cleanup --exit-command-arg
9198c59a8a3db50801c52ceaa39521b4381ac46ab7c16907130244d2a328e823 --socket-dir-
path /var/run/libpod/socket -t -l k8s-file:/var/lib/containers/storage/overlay-
containers/9198c59a8a3db50801c52ceaa39521b4381ac46ab7c16907130244d2a328e823/user
data/ctr.log --log-level error
root       3019  2.5  0.6  32656  5364 pts/0    Ss+  22:45   0:00 nginx: master
process nginx -g daemon off;
101        3031  0.0  0.3  33144  2636 pts/0    S+   22:45   0:00 nginx: worker
process
root       3034  0.0  0.1  12108  1072 pts/1    S+   22:45   0:00 grep --
color=auto nginx


[root@centos8 ~]#podman top nginx
USER    PID   PPID   %CPU    ELAPSED           TTY      TIME   COMMAND
root    1     0      0.000   4m38.979412738s   pts/0    0s     nginx: master
process nginx -g daemon off;
nginx   6     1      0.000   4m37.979473913s   pts/0    0s     nginx: worker
process
[root@centos8 ~]#podman stats nginx
ID             NAME    CPU %   MEM USAGE / LIMIT   MEM %   NET IO          BLOCK
IO   PIDS
9198c59a8a3d   nginx   --      2.474MB / 835.8MB   0.30%   2.25kB / 1.742kB  --
/ --    2
```

2 Docker 镜像制作和管理

2.1 Docker 镜像说明

2.1.1 Docker 镜像中有没有内核

从镜像大小上面来说，一个比较小的镜像只有1MB多点或几MB，而内核文件需要几十MB， 因此镜像里面是没有内核的，镜像在被启动为容器后将直接使用宿主机的内核，而镜像本身则只提供相应的rootfs，即系统正常运行所必须的用户空间的文件系统，比如: /dev/，/proc，/bin，/etc等目录，容器当中/boot目录是空的，而/boot当中保存的就是与内核相关的文件和目录。

2.1.2 为什么没有内核

由于容器启动和运行过程中是直接使用了宿主机的内核，不会直接调用物理硬件，所以也不会涉及到硬件驱动，因此也无需容器内拥有自己的内核和驱动。而如果使用虚拟机技术，对应每个虚拟机都有自己独立的内核

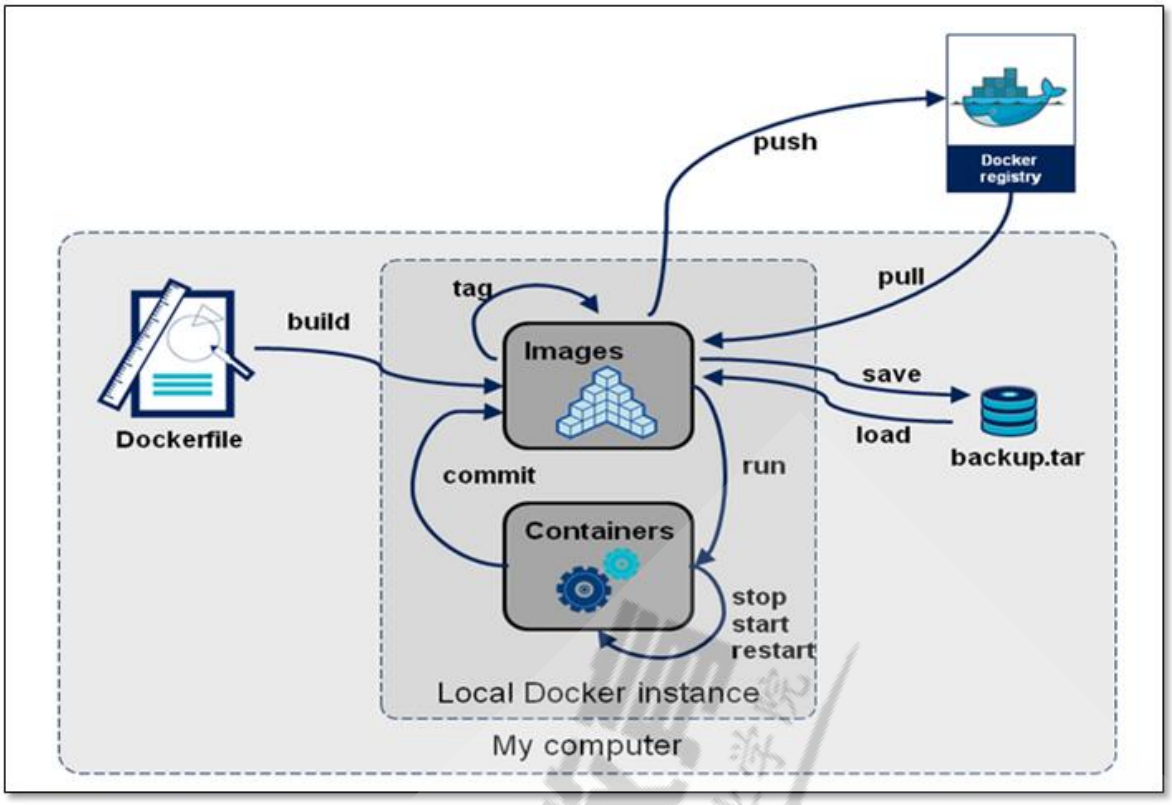
2.1.3 容器中的程序后台运行会导致此容器启动后立即退出

Docker容器如果希望启动后能持续运行,就必须有一个能前台持续运行的进程，如果在容器中启动传统的服务，如:httpd,php-fpm等均为后台进程模式运行,就导致 docker 在前台没有运行的应用,这样的容器启动后会立即退出。所以一般会将服务程序以前台方式运行，对于有一些可能不知道怎么实现前台运行的程序,只需要在你启动的该程序之后添加类似于 tail ， top 这种可以前台运行的程序即可. 比较常用的方法，如 `tail -f /etc/hosts`。

范例:

```
#httpd
ENTRYPOINT [ "/usr/sbin/apache2" ]
CMD ["-D", "FOREGROUND"]

#nginx
ENTRYPOINT [ "/usr/sbin/nginx", "-g", "daemon off;" ]

#用脚本运行容器
cat run_haproxy.sh
#!/bin/bash
haproxy -f /etc/haproxy/haproxy.cfg
tail -f /etc/hosts

tail -n1 Dockerfile
CMD ["run_haproxy.sh"]
```

2.1.4 docker 镜像生命周期



2.1.5 制作镜像方式

Docker 镜像制作类似于虚拟机的镜像（模版）制作，即按照公司的实际业务需求将需要安装的软件、相关配置等基础环境配置完成，然后将其做成镜像，最后再批量从镜像批量生成容器实例，这样可以极大的简化相同环境的部署工作.

Docker的镜像制作分为手动制作（基于容器）和自动制作(基于DockerFile)，企业通常都是基于Dockerfile制作镜像

```
docker commit #通过修改现有容器,将之手动构建为镜像
docker build  #通过Dockerfile文件,批量构建为镜像
```

2.2 将现有容器通过 docker commit 手动构建镜像

2.2.1 基于容器手动制作镜像步骤

docker commit 格式

```
docker commit [OPTIONS] CONTAINER [REPOSITORY[:TAG]]

#选项
-a, --author string    Author (e.g., "John Hannibal Smith <hannibal@a-
team.com>")
-c, --change list      Apply Dockerfile instruction to the created image
-m, --message string   Commit message
-p, --pause            Pause container during commit (default true)

#说明:
制作镜像和CONTAINER状态无关,停止状态也可以制作镜像
如果没有指定[REPOSITORY[:TAG]],REPOSITORY和TAG都为<none>
提交的时候标记TAG号:  生产当中常用，后期可以根据TAG标记创建不同版本的镜像以及创建不同版本的容器
```

基于容器手动制作镜像步骤具体如下:

1. 下载一个系统的官方基础镜像，如: CentOS 或 Ubuntu
2. 基于基础镜像启动一个容器,并进入到容器
3. 在容器里面做配置操作
    - 安装基础命令
    - 配置运行环境
    - 安装服务和配置服务
    - 放业务程序代码
4. 提交为一个新镜像 `docker commit`
5. 基于自己的的镜像创建容器并测试访问

2.2.2 实战案例: 基于 busybox 制作 httpd 镜像

```
[root@ubuntu1804 ~]#docker run -it --name b1 busybox
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
/ # mkdir /data/html -p
/ # echo httpd website in busybox > /data/html/index.html
/ # httpd --help
BusyBox v1.32.0 (2020-06-27 00:20:57 UTC) multi-call binary

Usage: httpd [-ifv[v]] [-c CONFFILE] [-p [IP:]PORT] [-u USER[:GRP]] [-r REALM]
[-h HOME]
or httpd -d/-e/-m STRING

Listen for incoming HTTP requests

    -i      Inetd mode
    -f      Don't daemonize
    -v[v]       Verbose 显示访问日志
    -p [IP:]PORT    Bind to IP:PORT (default *:80)
    -u USER[:GRP]   Set uid/gid after binding to port
    -r REALM    Authentication Realm for Basic Authentication
    -h HOME     Home directory (default .)
    -c FILE     Configuration file (default {/etc,HOME}/httpd.conf)
    -m STRING   MD5 crypt STRING
    -e STRING   HTML encode STRING
    -d STRING   URL decode STRING
/ # exit


#格式1
[root@ubuntu1804 ~]#docker commit -a "wang<root@wangxiaochun.com>" -c 'CMD
/bin/httpd -fv -h /data/html' -c "EXPOSE 80" b1 httpd-busybox:v1.0
#格式2
[root@ubuntu1804 ~]#docker commit -a "wang<root@wangxiaochun.com>" -c 'CMD
["/bin/httpd", "-f", "-v","-h", "/data/html"]' -c "EXPOSE 80" b1 httpd-
busybox:v1.0

[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
httpd-busybox       v1.0                e7883146c119        6 minutes ago
1.22MB
```

```
[root@ubuntu1804 ~]#docker run  -d -P --name httpd01 httpd-busybox:v1.0
ce95174c6385392b9699d12d1a86d7f81bc4dde1400a071ce17d9c78d905cb12

[root@ubuntu1804 ~]#docker port httpd01
80/tcp -> 0.0.0.0:32783

[root@ubuntu1804 ~]#docker inspect -f "
{{.NetworkSettings.Networks.bridge.IPAddress}}" httpd01
172.17.0.2

#对应格式1
[root@ubuntu1804 ~]#docker inspect -f "{{.Config.Cmd}}" httpd01
[/bin/sh -c /bin/httpd -f -h /data/html]

#对应格式2
[root@ubuntu1804 ~]#docker inspect -f "{{.Config.Cmd}}" httpd01
[/bin/httpd -f -h /data/html]

[root@ubuntu1804 ~]#docker exec -it httpd01 sh
/ # pstree -p
httpd(1)
/ # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/httpd -fv -h /data/html
    7 root      0:00 sh
   13 root      0:00 ps aux
/ #

[root@ubuntu1804 ~]#curl 172.17.0.2
httpd website in busybox

[root@ubuntu1804 ~]#curl 127.0.0.1:32783
httpd website in busybox

#再次制作镜像v2.0版
[root@ubuntu1804 ~]#docker commit -a "wang<root@wangxiaochun.com>" b1  httpd-
busybox:v2.0

[root@ubuntu1804 ~]#docker run -d --name web2 -p 81:80   httpd-busybox:v2.0
/bin/httpd -fv -h /data/html
c47bce0de75dcdf88266467accbe0a119190999c23cec32d6af8b8500aed96d4
```

2.2.3 实战案例: 基于官方镜像生成的容器制作 tomcat 镜像

2.2.3.1 下载官方的tomcat镜像并运行

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@ubuntu1804 ~]#docker run -d -p 8080:8080 tomcat
Unable to find image 'tomcat:latest' locally
latest: Pulling from library/tomcat
e9afc4f90ab0: Pull complete
989e6b19a265: Pull complete
```

```
af14b6c2f878: Pull complete
5573c4b30949: Pull complete
fb1a405f128d: Pull complete
612a9f566fdc: Pull complete
4226f9b63dac: Pull complete
cb8bfe875d7f: Pull complete
5aa366608b6d: Pull complete
4b7c9018ca5f: Pull complete
Digest: sha256:46456ccf216f2fde198844d5c7d511f60e3ffc2ea3828e0cce9a2eed566e48e2
Status: Downloaded newer image for tomcat:latest
6fcc5f7c1c7ce7be0b12007e310fc044e1077d489e1292b22621de55e302c455
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                    NAMES
6fcc5f7c1c7c        tomcat              "catalina.sh run"   42 seconds ago
Up 40 seconds       0.0.0.0:8080->8080/tcp   focused_wilbur

[root@ubuntu1804 ~]#curl -I 127.0.0.1:8080
HTTP/1.1 404
Content-Type: text/html;charset=utf-8
Content-Language: en
Transfer-Encoding: chunked
Date: Mon, 20 Jul 2020 15:33:21 GMT
```

2.2.3.2 修改容器

```
[root@ubuntu1804 ~]#docker exec -it 6fcc5f7c1c7c bash
root@6fcc5f7c1c7c:/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@6fcc5f7c1c7c:/usr/local/tomcat# ls webapps
root@6fcc5f7c1c7c:/usr/local/tomcat# ls webapps.dist/
ROOT  docs  examples  host-manager  manager
root@6fcc5f7c1c7c:/usr/local/tomcat# cp -a webapps.dist/* webapps/
root@6fcc5f7c1c7c:/usr/local/tomcat# ls webapps/
ROOT  docs  examples  host-manager  manager

root@6fcc5f7c1c7c:/usr/local/tomcat# exit
exit

[root@ubuntu1804 ~]#curl -I 127.0.0.1:8080
HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Date: Mon, 20 Jul 2020 15:34:24 GMT
```

2.2.3.3 提交新镜像

```
[root@ubuntu1804 ~]#docker commit -m "add webapps app" -a "wangxiaochun"
6fcc5f7c1c7c tomcat:9.0.37-v1
sha256:b8d669ebf99e65d5ed69378d0d53f054e7de4865d335ab7aa0a7a5508e1369f7
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
```

```
tomcat              9.0.37-v1           b8d669ebf99e        4 seconds ago
652MB
tomcat              latest              df72227b40e1        3 days ago
647MB

[root@ubuntu1804 ~]#docker history tomcat:9.0.37-v1
IMAGE               CREATED             CREATED BY
       SIZE                COMMENT
b8d669ebf99e        14 minutes ago      catalina.sh run
        4.84MB              add webapps app
df72227b40e1        3 days ago          /bin/sh -c #(nop)  CMD ["catalina.sh"
"run"]    0B
<missing>           3 days ago          /bin/sh -c #(nop)  EXPOSE 8080
       0B
<missing>           3 days ago          /bin/sh -c set -e  &&
nativeLines="$(catalin…   0B
<missing>           3 days ago          /bin/sh -c set -eux;
savedAptMark="$(apt-m…   20MB
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
TOMCAT_SHA512=077c3e6…   0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
TOMCAT_VERSION=9.0.37    0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV TOMCAT_MAJOR=9
        0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
GPG_KEYS=05AB33110949…   0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
LD_LIBRARY_PATH=/usr/…   0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
TOMCAT_NATIVE_LIBDIR=…   0B
<missing>           3 days ago          /bin/sh -c #(nop) WORKDIR
/usr/local/tomcat     0B
<missing>           3 days ago          /bin/sh -c mkdir -p "$CATALINA_HOME"
       0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
PATH=/usr/local/tomca…   0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
CATALINA_HOME=/usr/lo…   0B
<missing>           3 days ago          /bin/sh -c #(nop)  CMD ["jshell"]
        0B
<missing>           3 days ago          /bin/sh -c set -eux;   dpkgArch="$(dpkg
--pr…   323MB
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
JAVA_URL_VERSION=11.0…   0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
JAVA_BASE_URL=https:/…   0B
<missing>           3 days ago          /bin/sh -c #(nop)  ENV
JAVA_VERSION=11.0.8      0B
<missing>           5 weeks ago         /bin/sh -c { echo '#/bin/sh'; echo 'echo
"$J…   27B
<missing>           5 weeks ago         /bin/sh -c #(nop)  ENV
PATH=/usr/local/openj…   0B
<missing>           5 weeks ago         /bin/sh -c #(nop)  ENV
JAVA_HOME=/usr/local/…   0B
<missing>           5 weeks ago         /bin/sh -c #(nop)  ENV LANG=C.UTF-8
        0B
<missing>           5 weeks ago         /bin/sh -c set -eux;  apt-get update;
apt-g…   11.1MB
```

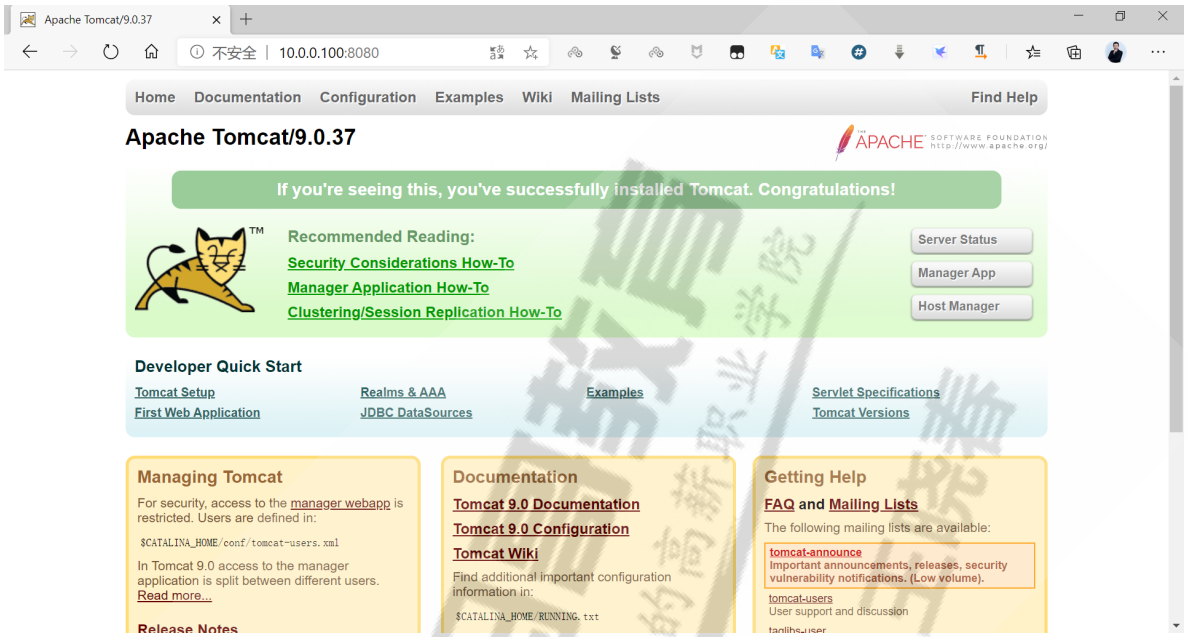
```
<missing>           5 weeks ago         /bin/sh -c apt-get update && apt-get
install…   146MB
<missing>           5 weeks ago         /bin/sh -c set -ex;  if ! command -v gpg
> /…   17.5MB
<missing>           5 weeks ago         /bin/sh -c apt-get update && apt-get
install…   16.5MB
<missing>           5 weeks ago         /bin/sh -c #(nop)  CMD ["bash"]
        0B
<missing>           5 weeks ago         /bin/sh -c #(nop) ADD
file:1ab357efe422cfed5…   114MB


[root@ubuntu1804 ~]#docker inspect tomcat:9.0.37-v1 |tail -n20
            "Type": "layers",
            "Layers": [

 "sha256:8803ef42039dcbe936755e9baae4bb7b19cb0fb6a438eb3992950cd0afef8e4f",

 "sha256:c2c789d2d3c5fc15428921b5316a63907d8c312c5508cbac9a704bd30bcc740d",

 "sha256:527ade4639e08d1b2ba9d0d46f3b43433310a2e8674ae808aeef5502b099ff3c",

 "sha256:2e5b4ca91984c19a75bad687aedebf02609511a033aa0d3d15ac915593279718",

 "sha256:f5181c7ef9028578eb0040a0051ce940c3f9ba62a51d3eb02ad52c3724db3e9b",

 "sha256:f73b2345c404b2a65abc66ab68af6b112aafb9d2583416d2eb84a47139e4626e",

 "sha256:afa12e842ed4e07340351e64318d3149b3ec8430f631d9bbe8e0d3ac24046cff",

 "sha256:31d7eb4c72a987dd8742add724562e0e800998cb7a57f7ed0c6dffa0587b5af4",

 "sha256:4b1785e8102bdcc4f7bb759d084e1f03b2b26c19fa42aa91c879ce26f36381d7",

 "sha256:aca03caac7b1c380545a00f970a42079bbc8928c9785492e9e289f60206b01e1",

 "sha256:860b0f130267d185ed8328a5fcd3ecd040a7afec64b63f7551c5f3af80709db5"
            ]
        },
        "Metadata": {
            "LastTagTime": "2020-07-20T23:42:00.403415564+08:00"
        }
    }
]

#删除当前的容器
[root@ubuntu1804 ~]#docker rm -f 6fcc5f7c1c7c
6fcc5f7c1c7c
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

2.2.3.4 利用新镜像启动容器

```
[root@ubuntu1804 ~]#docker run  -d -p 8080:8080 --name tomcat tomcat:9.0.37-v1
97688b4c835da4b222f3ca15af36b8c27d4e60ff33480c314daf3f123b1cb50c
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                 COMMAND             CREATED
STATUS              PORTS                     NAMES
97688b4c835d        tomcat:9.0.37-v1      "catalina.sh run"   6 seconds ago
Up 5 seconds        0.0.0.0:8080->8080/tcp    tomcat
```

2.2.3.5 测试新镜像启动的容器

浏览器访问`http://10.0.0.100:8080/`可以看到下面显示



2.2.4 实战案例: 基于Ubuntu的基础镜像利用 apt 安装手动制作 nginx 的镜像

2.2.4.1 启动Ubuntu基础镜像并实现相关的配置

```
[root@ubuntu1804 ~]#docker run -it -p 80 --name nginx_ubuntu ubuntu bash
root@705148273eac:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="20.04 LTS (Focal Fossa)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 20.04 LTS"
VERSION_ID="20.04"
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
PRIVACY_POLICY_URL="https://www.ubuntu.com/legal/terms-and-policies/privacy-
policy"
VERSION_CODENAME=focal
UBUNTU_CODENAME=focal
root@705148273eac:/# ll /etc/apt/sources.list
-rw-r--r-- 1 root root 2743 Jul  3 02:00 /etc/apt/sources.list

root@705148273eac:/# cat > /etc/apt/sources.list
deb http://mirrors.aliyun.com/ubuntu/ focal main restricted universe multiverse
```

```
deb-src http://mirrors.aliyun.com/ubuntu/ focal main restricted universe
multiverse

deb http://mirrors.aliyun.com/ubuntu/ focal-security main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-security main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ focal-updates main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-updates main restricted universe
multiverse

deb http://mirrors.aliyun.com/ubuntu/ focal-proposed main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-proposed main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ focal-backports main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ focal-backports main restricted
universe multiverse
^C
root@705148273eac:/# apt update
Get:1 http://mirrors.aliyun.com/ubuntu focal InRelease [265 kB]
Get:2 http://mirrors.aliyun.com/ubuntu focal-security InRelease [107 kB]
Get:3 http://mirrors.aliyun.com/ubuntu focal-updates InRelease [111 kB]
Get:4 http://mirrors.aliyun.com/ubuntu focal-proposed InRelease [265 kB]
Get:5 http://mirrors.aliyun.com/ubuntu focal-backports InRelease [98.3 kB]
Get:6 http://mirrors.aliyun.com/ubuntu focal/restricted Sources [7198 B]
Get:7 http://mirrors.aliyun.com/ubuntu focal/multiverse Sources [208 kB]
......
Fetched 28.7 MB in 6s (4651 kB/s)

Reading package lists... Done
Building dependency tree
Reading state information... Done
8 packages can be upgraded. Run 'apt list --upgradable' to see them.

root@705148273eac:/# apt -y install nginx
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
........
Configuring tzdata
------------------

Please select the geographic area in which you live. Subsequent configuration
questions will narrow this down
by presenting a list of cities, representing the time zones in which they are
located.

  1. Africa   3. Antarctica  5. Arctic  7. Atlantic  9. Indian    11. SystemV
 13. Etc
  2. America  4. Australia   6. Asia    8. Europe    10. Pacific  12. US
Geographic area: 6
```

```
Please select the city or region corresponding to your time zone.

  1. Aden       16. Brunei       31. Hong_Kong    46. Kuala_Lumpur  61. Pyongyang
     76. Tehran
  2. Almaty     17. Chita        32. Hovd         47. Kuching       62. Qatar
     77. Tel_Aviv
  3. Amman      18. Choibalsan   33. Irkutsk      48. Kuwait        63. Qostanay
      78. Thimphu
  4. Anadyr     19. Chongqing    34. Istanbul     49. Macau         64. Qyzylorda
     79. Tokyo
  5. Aqtau      20. Colombo      35. Jakarta      50. Magadan       65. Rangoon
     80. Tomsk
  6. Aqtobe     21. Damascus     36. Jayapura     51. Makassar      66. Riyadh
      81. Ujung_Pandang
  7. Ashgabat   22. Dhaka        37. Jerusalem    52. Manila        67. Sakhalin
      82. Ulaanbaatar
  8. Atyrau     23. Dili         38. Kabul        53. Muscat        68. Samarkand
     83. Urumqi
  9. Baghdad    24. Dubai        39. Kamchatka    54. Nicosia       69. Seoul
     84. Ust-Nera
  10. Bahrain   25. Dushanbe     40. Karachi      55. Novokuznetsk  70. Shanghai
      85. Vientiane
  11. Baku      26. Famagusta    41. Kashgar      56. Novosibirsk   71. Singapore
     86. Vladivostok
  12. Bangkok   27. Gaza         42. Kathmandu    57. Omsk          72.
Srednekolymsk  87. Yakutsk
  13. Barnaul   28. Harbin       43. Khandyga     58. Oral          73. Taipei
      88. Yangon
  14. Beirut    29. Hebron       44. Kolkata      59. Phnom_Penh    74. Tashkent
      89. Yekaterinburg
  15. Bishkek   30. Ho_Chi_Minh  45. Krasnoyarsk  60. Pontianak     75. Tbilisi
     90. Yerevan
Time zone: 70   #配置时区
......
Setting up nginx-core (1.18.0-0ubuntu1) ...
invoke-rc.d: could not determine current runlevel
invoke-rc.d: policy-rc.d denied execution of start.
Setting up nginx (1.18.0-0ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9) ...
root@705148273eac:/# nginx -v
nginx version: nginx/1.18.0 (Ubuntu)
root@705148273eac:~# grep include /etc/nginx/nginx.conf
include /etc/nginx/modules-enabled/*.conf;
    include /etc/nginx/mime.types;
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
root@705148273eac:~# grep root /etc/nginx/sites-enabled/default
    root /var/www/html;
    # deny access to .htaccess files, if Apache's document root
#   root /var/www/example.com;
root@705148273eac:/# echo Nginx Website in Docker >  /var/www/html/index.html
root@705148273eac:/# exit
exit
```

2.2.4.2 提交为镜像

```
[root@ubuntu1804 ~]#docker commit -a 'wangxiaochun' -m 'nginx-ubuntu:20.04'
nginx_ubuntu nginx_ubuntu20.04:v1.18.0
sha256:2c789ec21d2545c9bfc4af6d4380878153d52fcc03890aac755d09112631742a

[root@ubuntu1804 ~]#docker images
REPOSITORY             TAG                 IMAGE ID            CREATED
  SIZE
nginx_ubuntu20.04      v1.18.0             2c789ec21d25        22 seconds ago
   179MB
```

2.2.4.3 从制作的新镜像启动容器并测试访问

```
[root@ubuntu1804 ~]#docker run -d -p 80 --name nginx-web
nginx_ubuntu20.04:v1.18.0 nginx -g 'daemon off;'
b0c8496a497ba60f7b5bc430b075b00d40c7ace24068e71decac625e84df40de
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
            STATUS              PORTS                    NAMES
b0c8496a497b        nginx_ubuntu20.04:v1.18.0   "nginx -g 'daemon of…"   7
seconds ago       Up 5 seconds        0.0.0.0:32771->80/tcp   nginx-web
[root@ubuntu1804 ~]#docker port nginx-web
80/tcp -> 0.0.0.0:32771
[root@ubuntu1804 ~]#curl http://127.0.0.1:32771
Nginx Website in Docker
```

2.2.5 实战案例: 基于CentOS的基础镜像利用 yum 安装手动制作 nginx 的镜像

2.2.5.1下载基础镜像并初始化系统

基于某个基础镜像之上重新制作，因此需要先有一个基础镜像，本次使用官方提供的centos镜像为基础

```
[root@ubuntu1804 ~]#docker pull centos:centos7.7.1908
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
[root@ubuntu1804 ~]#docker run -it centos:centos7.7.1908 bash

#修改时区
[root@9caa8742e6ce /]#rm -f /etc/localtime
[root@9caa8742e6ce /]#ln -s ../usr/share/zoneinfo/Asia/Shanghai /etc/localtime
[root@9caa8742e6ce /]# yum -y install wget
[root@9caa8742e6ce /]# rm -rf /etc/yum.repos.d/*

#更改yum 源
[root@9caa8742e6ce /]# wget -P /etc/yum.repos.d/
http://mirrors.aliyun.com/repo/Centos-7.repo
[root@9caa8742e6ce /]# wget -P /etc/yum.repos.d/
http://mirrors.aliyun.com/repo/epel-7.repo
```

2.2.5.2 安装相关软件和工具

```
#yum安装nginx
[root@9caa8742e6ce /]# yum install nginx -y

#安装常用命令
[root@9caa8742e6ce /]# yum install -y vim curl iproute net-tools

#清理yum缓存
[root@9caa8742e6ce /]# rm -rf /var/cache/yum/*
```

2.2.5.3 修改服务的配置信息关闭服务后台运行

```
#关闭nginx后台运行
[root@9caa8742e6ce /]# vim /etc/nginx/nginx.conf
user nginx;
daemon off; #关闭后台运行
```

2.2.5.4 准备程序和数据

```
#自定义web界面
[root@9caa8742e6ce ~]# rm -f /usr/share/nginx/html/index.html
[root@9caa8742e6ce ~]# echo "Nginx Page in Docker" >
/usr/share/nginx/html/index.html
```

2.2.5.5 提交为镜像

docker commit 命令在宿主机基于容器ID 提交为镜像

```
#不关闭容器的情况,将容器提交为镜像
[root@ubuntu1804 ~]#docker commit -a "root@wangxiaochun.com" -m "nginx yum v1" -
c "EXPOSE 80 443" 9caa8742e6ce wang/centos7-nginx:1.16.1.v1
sha256:e9d09cc585ed8ee1544b1e68de326ea6dcbe99577fc9b2edad9ab481b7a7e7ec
[root@ubuntu1804 ~]#docker images
REPOSITORY           TAG                 IMAGE ID            CREATED
SIZE
wang/centos7-nginx   1.16.1.v1           e9d09cc585ed        4 seconds ago
442MB
centos               centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.2.5.6 从制作的镜像启动容器

```
[root@ubuntu1804 ~]#docker run -d -p 8080:80 --name my-centos-nginx
wang/centos7-nginx:1.16.1.v1 /usr/sbin/nginx
c60f8373a14210bb3aa06ce03c2258a4b912033b0650ef690f9245fc3afc5bf1

[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                              COMMAND             CREATED
          STATUS              PORTS                                NAMES
c60f8373a142        wang/centos7-nginx:1.16.1.v1   "/usr/sbin/nginx"   6 seconds
ago       Up 5 seconds        443/tcp, 0.0.0.0:8080->80/tcp   my-centos-nginx
9caa8742e6ce        centos:centos7.7.1908          "bash"              35
minutes ago      Up 35 minutes
thirsty_hypatia
[root@ubuntu1804 ~]#
```

2.2.5.7 访问测试镜像

```
[root@ubuntu1804 ~]#curl 127.0.0.1:8080
Nginx Page in Docker
[root@ubuntu1804 ~]#
```

2.2.6 实战案例: 基于CentOS 基础镜像手动制作编译版本 nginx 镜像

在CentOS 基础镜像的容器之上手动编译安装nginx，然后再将此容器提交为镜像

2.2.6.1 下载镜像并初始化系统

```
[root@ubuntu1804 ~]#docker pull centos:centos7.7.1908
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
[root@ubuntu1804 ~]#docker run -it centos:centos7.7.1908 /bin/bash

#生成yum源配置
[root@86a48908bb97 /]# yum -y install wget
[root@64944257fa88 /]# rm -rf /etc/yum.repos.d/*
[root@64944257fa88 /]# wget -P /etc/yum.repos.d/
http://mirrors.aliyun.com/repo/Centos-7.repo
http://mirrors.aliyun.com/repo/epel-7.repo
```

2.2.6.2 编译安装 nginx

```
[root@64944257fa88 /]# useradd -r -s /sbin/nologin nginx
#安装基础包
[root@64944257fa88 /]# yum -y install gcc gcc-c++ automake pcre pcre-devel zlib
zlib-devel openssl openssl-devel

[root@64944257fa88 /]# cd /usr/local/src
[root@64944257fa88 src]# wget http://nginx.org/download/nginx-1.16.1.tar.gz
[root@64944257fa88 src]# tar xf nginx-1.16.1.tar.gz
[root@64944257fa88 src]# cd nginx-1.16.1
[root@64944257fa88 nginx-1.16.1]# ./configure  --prefix=/apps/nginx
[root@64944257fa88 nginx-1.16.1]# make && make install
[root@64944257fa88 nginx-1.16.1]# rm -rf nginx*
[root@64944257fa88 nginx-1.16.1]# rm -rf /var/cache/yum/*
```

2.2.6.3 关闭 nginx 后台运行

```
[root@64944257fa88 nginx-1.16.1]# cd /apps/nginx/
[root@64944257fa88 nginx]# ls
conf  html  logs  sbin
[root@64944257fa88 nginx]# vi conf/nginx.conf
user nginx;
daemon off;
[root@64944257fa88 nginx]# ln -s /apps/nginx/sbin/nginx  /usr/sbin/
[root@64944257fa88 nginx]# ll /usr/sbin/nginx
lrwxrwxrwx 1 root root 22 Jan 28 05:29 /usr/sbin/nginx -> /apps/nginx/sbin/nginx
```

2.2.6.4 准备相关数据自定义web界面

```
[root@64944257fa88 nginx]# echo "Nginx Test Page in Docker" >
/apps/nginx/html/index.html
```

2.2.6.5 提交为镜像

```
#不要退出容器，在另一个终端窗口执行以下命令
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                   COMMAND             CREATED
    STATUS              PORTS               NAMES
64944257fa88        centos:centos7.7.1908   "/bin/bash"         18 seconds ago
   Up 17 seconds                           stupefied_albattani
[root@ubuntu1804 ~]#docker commit -m "nginx1.6.1" 64944257fa88 -c "CMD nginx"
centos7-nginx:1.6.1
sha256:d86d957bb00f35fe09ae38230e1e2d12916f4406e997146c68e34dae7526c079
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos7-nginx       1.6.1               d86d957bb00f        2 minutes ago
486MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.2.6.6 从自己的镜像启动容器

```
[root@ubuntu1804 ~]#docker run -d -p 80:80 centos7-nginx:1.6.1 nginx
ae90b1abf374138a21f7ed104d14c88f1af23c0b2027c3fe099722fd7fbad3a4
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                 COMMAND             CREATED
   STATUS              PORTS               NAMES
ae90b1abf374        centos7-nginx:1.6.1   "nginx"             About a minute ago
  Up About a minute   0.0.0.0:80->80/tcp   naughty_thompson
```

备注：最后面的nginx是运行的命令，即镜像里面要运行一个nginx命令，所以前面软链接到/usr/sbin/nginx，目的为了让系统不需要指定路径就可以执行此命令

2.2.6.7 访问测试

```
[root@ubuntu1804 ~]#curl 127.0.0.1
Nginx Test Page in Docker
```

2.2.6.8 查看Nginx访问日志和进程

```
[root@ubuntu1804 ~]#docker exec -it ae90b1abf374 bash
[root@ae90b1abf374 /]# cat /apps/nginx/logs/access.log
172.17.0.1 - - [28/Jan/2020:05:40:51 +0000] "GET / HTTP/1.1" 200 26 "-"
"curl/7.58.0"

[root@ae90b1abf374 /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.2  20572  2468 ?        Ss   05:40   0:00 nginx: master
process nginx
nginx         6  0.0  0.3  21024  3104 ?        S    05:40   0:00 nginx: worker
process
root          7  0.3  0.2  11840  2928 pts/0    Ss   05:45   0:00 bash
root         21  0.0  0.3  51764  3344 pts/0    R+   05:46   0:00 ps aux
```

2.3 利用 DockerFile 文件执行 docker build 自动构建镜像

2.3.1 Dockfile 使用详解

2.3.1.1 Dockerfile 介绍

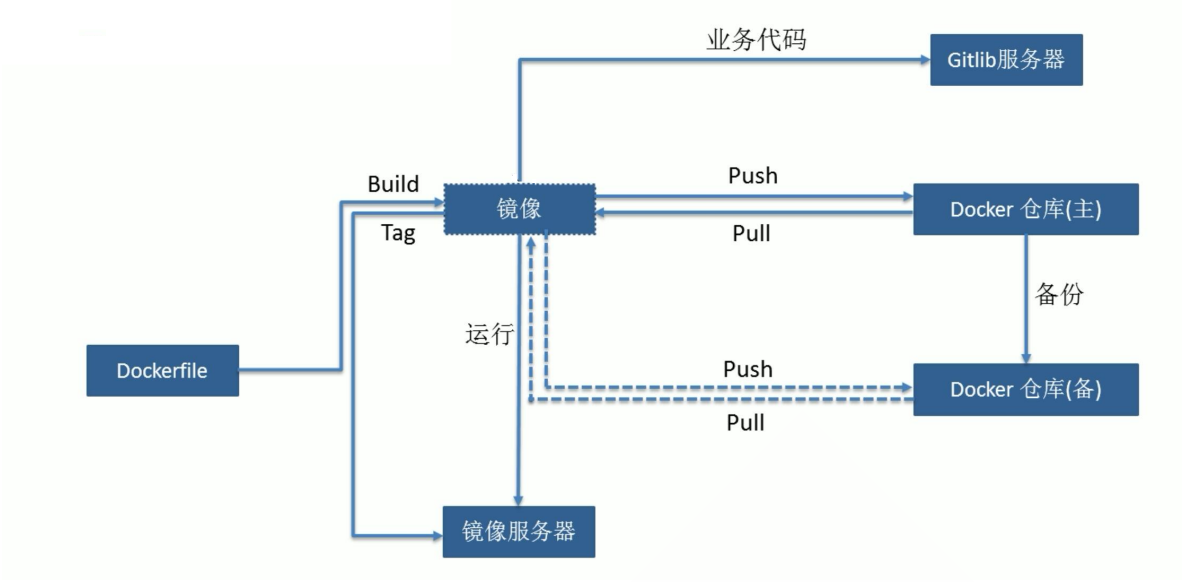
DockerFile 是一种被Docker程序解释执行的脚本，由一条条的命令组成的，每条命令对应linux下面的一条命令，Docker程序将这些DockerFile指令再翻译成真正的linux命令，其有自己的书写方式和支持的命令，Docker程序读取DockerFile并根据指令生成Docker镜像，相比手动制作镜像的方式，DockerFile更能直观的展示镜像是怎么产生的，有了DockerFile，当后期有额外的需求时，只要在之前的DockerFile添加或者修改响应的命令即可重新生成新的Docker镜像，避免了重复手动制作镜像的麻烦,类似与shell脚本一样,可以方便高效的制作镜像

Docker守护程序 `Dockerfile` 逐一运行指令，如有必要，将每个指令的结果提交到新镜像，然后最终输出新镜像的ID。Docker守护程序将自动清理之前发送的上下文

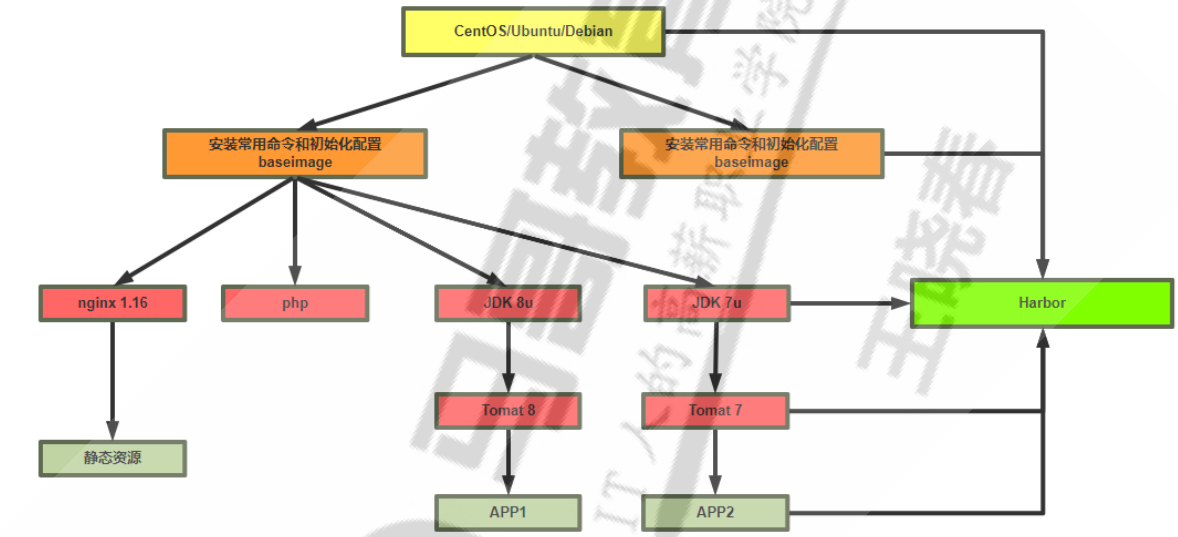
请注意，每条指令都是独立运行的，并会导致创建新镜像，比如 `RUN cd /tmp` 对下一条指令不会有任何影响。

Docker将尽可能重用中间镜像层（缓存），以显著加速 `docker build` 命令的执行过程，这由 `Using cache` 控制台输出中的消息指示

2.3.1.2 Dockerfile 镜像制作和使用流程



2.3.1.3 Dockerfile文件的制作镜像的分层结构



范例:

```
#按照业务类型或系统类型等方式划分创建目录环境，方便后期镜像比较多的时候进行分类
[root@ubuntu1804 ~]#mkdir
/data/dockerfile/{web/{nginx,apache,tomcat,jdk},system/{centos,ubuntu,alpine,deb
ian}} -p
[root@ubuntu1804 ~]#tree /data/dockerfile/
/data/dockerfile/
├── system
│   ├── alpine
│   ├── centos
│   ├── debian
│   └── ubuntu
└── web
    ├── apache
    ├── jdk
    ├── nginx
    └── tomcat

10 directories, 0 files
[root@ubuntu1804 ~]#
```

2.3.1.4 Dockerfile 文件格式

Dockerfile 是一个有特定语法格式的文本文件

dockerfile 官方说明: https://docs.docker.com/engine/reference/builder/

帮助: man 5 dockerfile

Dockerfile 文件说明

- 每一行以Dockerfile的指令开头，指令不区分大小写，但是惯例使用大写
- 使用 # 开始作为注释
- 每一行只支持一条指令，每条指令可以携带多个参数
- 指令按文件的顺序从上至下进行执行
- 每个指令的执行会生成一个新的镜像层，为了减少分层和镜像大小，尽可能将多条指令合并成一条指令
- 制作镜像一般可能需要反复多次，每次执行dockfile都按顺序执行，从头开始，已经执行过的指令已经缓存，不需要再执行，如果后续有一行新的指令没执行过，其往后的指令将会重新执行，所以为加速镜像制作，将最常变化的内容放下dockerfile的文件的后面

2.3.1.5 Dockerfile 相关指令

dockerfile 文件中的常见指令:

```
ADD
COPY
ENV
EXPOSE
FROM
LABEL
STOPSIGNAL
USER
VOLUME
WORKDIR
```

2.3.1.5.1 FROM: 指定基础镜像

定制镜像，需要先有一个基础镜像，在这个基础镜像上进行定制。

FROM 就是指定基础镜像，此指令通常必需放在Dockerfile文件第一个非注释行。后续的指令都是运行于此基准镜像所提供的运行环境

基础镜像可以是任何可用镜像文件，默认情况下，docker build会在docker主机上查找指定的镜像文件，在其不存在时，则会从Docker Hub Registry上拉取所需的镜像文件.如果找不到指定的镜像文件，docker build会返回一个错误信息

如何选择合适的镜像呢?

对于不同的软件官方都提供了相关的docker镜像，比如: nginx、redis、mysql、httpd、tomcat等服务类的镜像，也有操作系统类，如: centos、ubuntu、debian等。建议使用官方镜像，比较安全。

格式:

```
FROM [--platform=<platform>] <image> [AS <name>]
FROM [--platform=<platform>] <image>[:<tag>] [AS <name>]
FROM [--platform=<platform>] <image>[@<digest>] [AS <name>]

#说明:
--platform 指定镜像的平台，比如:  linux/amd64, linux/arm64, or windows/amd64
tag 和 digest是可选项，如果不指定，默认为latest
```

说明: 关于scratch 镜像

```
FROM scratch
参考链接:
https://hub.docker.com/_/scratch?tab=description
https://docs.docker.com/develop/develop-images/baseimages/
该镜像是一个空的镜像，可以用于构建busybox等超小镜像，可以说是真正的从零开始构建属于自己的镜像
该镜像在构建基础镜像（例如debian和busybox）或超最小镜像（仅包含一个二进制文件及其所需内容，例如:hello-world）的上下文中最有用。
```

范例:

```
FROM scratch #所有镜像的起源镜像，相当于Object类
FROM ubuntu
FROM ubuntu:bionic
FROM debian:buster-slim
```

2.3.1.5.2 LABEL: 指定镜像元数据

可以指定镜像元数据，如: 镜像作者等

```
LABEL <key>=<value> <key>=<value> <key>=<value> ...
```

范例:

```
LABEL "com.example.vendor"="ACME Incorporated"
LABEL com.example.label-with-value="foo"
LABEL version="1.0"
LABEL description="This text illustrates \
that label-values can span multiple lines."
```

一个镜像可以有多个label ,还可以写在一行中,即多标签写法,可以减少镜像的的大小

范例: 多标签写法

```
#一行格式
LABEL multi.label1="value1" multi.label2="value2" other="value3"

#多行格式
LABEL multi.label1="value1" \
      multi.label2="value2" \
      other="value3"
```

docker inspect 命令可以查看LABEL

范例:

```
"Labels": {
    "com.example.vendor": "ACME Incorporated"
    "com.example.label-with-value": "foo",
    "version": "1.0",
    "description": "This text illustrates that label-values can span multiple
lines.",
    "multi.label1": "value1",
    "multi.label2": "value2",
    "other": "value3"
},
```

MAINTAINER: 指定维护者信息

此指令已过时，用LABEL代替

```
MAINTAINER <name>
```

范例:

```
MAINTAINER wangxiaochun <root@wangxiaochun.com>
#用LABEL代替
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
```

2.3.1.5.3 RUN: 执行 shell命令

RUN 指令用来在**构建镜像阶段**需要执行 FROM 指定镜像所支持的Shell命令。

通常各种基础镜像一般都支持丰富的shell命令

注意: RUN 可以写多个，每一个RUN指令都会建立一个镜像层，所以尽可能合并成一条指令,比如将多个shell命令通过 && 连接一起成为在一条指令

每个RUN都是独立运行的,和前一个RUN无关

```
#shell 格式: 相当于 /bin/sh -c  <命令>  此种形式支持环境变量
RUN <命令>

#exec 格式: 此种形式不支持环境变量,注意:是双引号,不能是单引号
RUN ["可执行文件", "参数1", "参数2"]

#exec格式可以指定其它shell
RUN ["/bin/bash","-c","echo hello wang"]
```

说明:

```
shell格式中，<command>通常是一个shell命令，且以"/bin/sh -c"来运行它，这意味着此进程在容器中的PID不为1，不能接收Unix信号，因此，当使用docker stop <container>命令停止容器时，此进程接收不到SIGTERM信号

exec格式中的参数是一个JSON格式的数组，其中<executable>为要运行的命令，后面的<paramN>为传递给命令的选项或参数;然而，此种格式指定的命令不会以"/bin/sh -c"来发起，因此常见的shell操作如变量替换以及通配符(?,*等)替换将不会进行;不过，如果要运行的命令依赖于此shell特性的话，可以将其替换为类似下面的格式。
RUN ["/bin/bash", "-c", "<executable>", "<param1>"]
```

范例:

```
RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
RUN ["/bin/bash", "-c", "echo hello world"]
RUN yum -y install epel-release \
      && yum -y install nginx \
      && rm -rf /usr/share/nginx/html/* 
      && echo "<h1> docker test nginx </h1>" > /usr/share/nginx/html/index.html
```

范例: 多个 前后RUN 命令独立无关和shell命令不同

```
#world.txt并不存放在/app内
RUN cd /app
RUN echo "hello" > world.txt
```

2.3.1.5.4 ENV: 设置环境变量

ENV 可以定义环境变量和值，会被后续指令(如:ENV,ADD,COPY,RUN等)通过$KEY或${KEY}进行引用，并在容器运行时保持

```
#变量赋值格式1
ENV <key> <value>    #此格式只能对一个key赋值,<key>之后的所有内容均会被视作其<value>的组成部分

#变量赋值格式2
ENV <key1>=<value1> <key2>=<value2> \  #此格式可以支持多个key赋值,定义多个变量建议使用,减少镜像层
    <key3>=<value3> ...

#如果<value>中包含空格，可以以反斜线\进行转义，也可通过对<value>加引号进行标识;另外，反斜线也可用于续行


#只使用一次变量
RUN <key>=<value> <command>

#引用变量
RUN $key .....

#变量支持高级赋值格式
${key:-word}
${kye:+word}
```

如果运行容器时如果需要修改变量,可以执行下面通过基于`exec`机制实现

注意: 下面方式只影响容器运行时环境,而不影响构建镜像的过程,即只能覆盖docker run时的环境变量,而不会影响docker build时环境变量的值

```
docker run -e|--env <key>=<value>

#说明
-e, --env list          #Set environment variables
    --env-file filename     #Read in a file of environment variables
```

示例: 两种格式功能相同

```
#格式1
ENV myName="John Doe" myDog=Rex\ The\ Dog \
    myCat=fluffy

#格式2
ENV myName John Doe
ENV myDog Rex The  Dog
ENV myCat fluffy
```

范例:

```
ENV VERSION=1.0 DEBUG=on NAME="Happy Feet"
ENV PG_MAJOR 9.3
ENV PG_VERSION 9.3.4
RUN curl -SL http://example.com/postgres-$PG_VERSION.tar.xz | tar -xJC
/usr/src/postgress && …
ENV PATH /usr/local/postgres-$PG_MAJOR/bin:$PATH
```

范例:

```
[root@ubuntu1804 dockerfile]#cat Dockerfile
FROM busybox
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
ENV NAME wang xiao chun
RUN touch $NAME.txt

[root@ubuntu1804 dockerfile]#cat build.sh
#!/bin/bash
#
TAG=$1
docker build -t test:$TAG .
[root@ubuntu1804 dockerfile]#./build.sh v5.0

[root@ubuntu1804 dockerfile]#docker run  --rm  --name c1 test:v5.0 env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=d4e1f89aca71
NAME=wang xiao chun
HOME=/root

[root@ubuntu1804 dockerfile]#docker run  --rm -e NAME=mage --name c1 test:v5.0
env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=b23500aa100d
NAME=mage
HOME=/root

[root@ubuntu1804 dockerfile]#docker run  --rm -e NAME=mage --name c1 test:v5.0
ls -l
total 36
drwxr-xr-x    2 root     root         12288 Jun 27 00:21 bin
-rw-r--r--    1 root     root             0 Jul 24 10:05 chun.txt
drwxr-xr-x    5 root     root           340 Jul 24 10:05 dev
drwxr-xr-x    1 root     root          4096 Jul 24 10:05 etc
drwxr-xr-x    2 nobody   nogroup       4096 Jun 27 00:21 home
dr-xr-xr-x  222 root     root             0 Jul 24 10:05 proc
drwx------    2 root     root          4096 Jun 27 00:21 root
```

```
dr-xr-xr-x   13 root     root            0 Jul 24 10:05 sys
drwxrwxrwt    2 root     root         4096 Jun 27 00:21 tmp
drwxr-xr-x    3 root     root         4096 Jun 27 00:21 usr
drwxr-xr-x    4 root     root         4096 Jun 27 00:21 var
-rw-r--r--    1 root     root            0 Jul 24 10:05 wang
-rw-r--r--    1 root     root            0 Jul 24 10:05 xiao

[root@ubuntu1804 dockerfile]#cat  env.txt
NAME=wang
TITLE=cto
[root@ubuntu1804 dockerfile]#docker run  --rm --env-file env.txt --name c1
test:v5.0 env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOSTNAME=88ddf5b73a6b
NAME=wang
TITLE=cto
HOME=/root
```

2.3.1.5.5 COPY: 复制文本

复制本地宿主机的 到容器中的 。

```
COPY [--chown=<user>:<group>] <src>... <dest>
COPY [--chown=<user>:<group>] ["<src>",... "<dest>"] #路径中有空白字符时,建议使用此格式
```

说明:

- 可以是多个,可以使用通配符，通配符规则满足Go的filepath.Match 规则

  filepath.Match 参考链接: `https://golang.org/pkg/path/filepath/#Match`
- 必须是build上下文中的路径(为 Dockerfile 所在目录的相对路径），**不能是其父目录中的文件**
- 如果是目录，则其内部文件或子目录会被递归复制，**但目录自身不会被复制**
- 如果指定了多个, 或在中使用了通配符，则必须是一个目 录，**且必须以 / 结尾**
- 可以是绝对路径或者是 `WORKDIR` 指定的相对路径
- 使用 COPY 指令，源文件的各种元数据都会保留。比如读、写、执行权限、文件变更时间等
- 如果事先不存在，它将会被自动创建，这包括其父目录路径,即递归创建目录

范例:

```
COPY hom* /mydir/
COPY hom?.txt /mydir/
```

2.3.1.5.6 ADD: 复制和解包文件

该命令可认为是增强版的COPY，不仅支持COPY，还支持自动解缩。可以将复制指定的 到容器中的

```
ADD [--chown=<user>:<group>] <src>... <dest>
ADD [--chown=<user>:<group>] ["<src>",... "<dest>"]
```

说明:

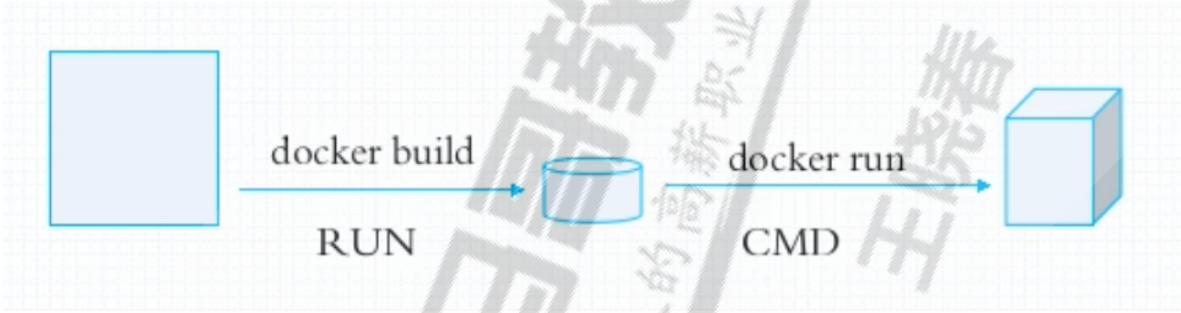
- 可以是Dockerfile所在目录的一个相对路径；也可是一个 URL；还可是一个 tar 文件（自动解压）
- 可以是绝对路径或者是 `WORKDIR` 指定的相对路径

- 如果是目录，只复制目录中的内容，而非目录本身
- 如果是一个 URL ，下载后的文件权限自动设置为 600
- 如果为URL且不以/结尾，则指定的文件将被下载并直接被创建为,如果以 / 结尾，则文件名URL指定的文件将被直接下载并保存为/< filename>
- 如果是一个本地文件系统上的打包文件,如: gz, bz2 ,xz ， 它将被解包 ， 其行为类似于"tar -x"命令,但是通过URL获取到的tar文件将不会自动展开
- 如果有多个，或其间接或直接使用了通配符，则必须是一个以/结尾的目录路径;如果不以/结尾，则其被视作一个普通文件，的内容将被直接写入到

范例:

```
ADD test relativeDir/          # adds "test" to `WORKDIR`/relativeDir/
ADD test /absoluteDir/         # adds "test" to /absoluteDir/
ADD --chown=55:mygroup files* /somedir/
ADD --chown=bin files* /somedir/
ADD --chown=1 files* /somedir/
ADD --chown=10:11 files* /somedir/
ADD ubuntu-xenial-core-cloudimg-amd64-root.tar.gz /
```

2.3.1.5.7 CMD: 容器启动命令



一个容器中需要持续运行的进程一般只有一个,CMD 用来指定启动容器时默认执行的一个命令，且其运行结束后,容器也会停止,所以一般CMD 指定的命令为持续运行且为前台命令.

- 如果docker run没有指定任何的执行命令或者dockerfile里面也没有ENTRYPOINT，那么开启容器时就会使用执行CMD指定的默认的命令
- 前面介绍过的 RUN 命令是在构建镜像进执行的命令,注意二者的不同之处
- 每个 Dockerfile 只能有一条 CMD 命令。如指定了多条，只有最后一条被执行
- 如果用户启动容器时用 docker run xxx 指定运行的命令，则会覆盖 CMD 指定的命令

```
# 使用 exec 执行，推荐方式，第一个参数必须是命令的全路径,此种形式不支持环境变量
CMD ["executable","param1","param2"]

# 在 /bin/sh 中执行，提供给需要交互的应用；此种形式支持环境变量
CMD command param1 param2

# 提供给 ENTRYPOINT 命令的默认参数
CMD ["param1","param2"]
```

范例:

```
CMD ["nginx", "-g", "daemon off;"]
```

范例:

```
[root@ubuntu1804 dockerfile]#cat Dockerfile
```

```
FROM ubuntu:18.04
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
RUN apt update \
&& apt -y install  curl \
&& rm -rf /var/lib/apt/lists/*
CMD [ "curl", "-s","https://ip.cn"]

[root@centos8 ubuntu]#podman  run  9b
{"ip": "111.199.187.36", "country": "北京市", "city": "联通"}

#cat /etc/etc/issue覆盖了curl命令
[root@centos8 ubuntu]#podman  run  9b cat /etc/issue
Ubuntu 18.04.4 LTS \n \l
```

范例:

```
[root@ubuntu1804 dockerfile]#pwd
/data/dockerfile

[root@ubuntu1804 dockerfile]#cat Dockerfile
FROM busybox
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
ENV ROOT /data/website
COPY index.html ${ROOT}/index.html
CMD /bin/httpd -f -h ${ROOT}
EXPOSE 80

[root@ubuntu1804 dockerfile]#cat index.html
Website in Dockerfile


[root@ubuntu1804 dockerfile]#cat build.sh
#!/bin/bash
#
TAG=$1
docker build -t test:$TAG .

[root@ubuntu1804 dockerfile]#./build.sh v1.0
[root@ubuntu1804 dockerfile]#docker run  -d  --rm -P  --name c1  test:v1.0
[root@ubuntu1804 ~]#docker port c1
80/tcp -> 0.0.0.0:32781
[root@ubuntu1804 ~]#curl 127.0.0.1:32781
Website in Dockerfile
[root@ubuntu1804 ~]#
```

范例:

```
[root@ubuntu1804 dockerfile]#pwd
/data/dockerfile
[root@ubuntu1804 dockerfile]#cat Dockerfile
FROM busybox
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
ENV ROOT /data/website
RUN mkdir -p ${ROOT} && echo '<h1> Busybox httpd server in Dockerfile</h1>' >
${ROOT}/index.html
#COPY index.html ${ROOT}/index.html
```

```
CMD ["/bin/sh","-c","/bin/httpd -f -h ${ROOT}"]
#CMD /bin/httpd -f -h ${ROOT}
EXPOSE 80

[root@ubuntu1804 dockerfile]#cat build.sh
#!/bin/bash
#
TAG=$1
docker build -t test:$TAG .

[root@ubuntu1804 dockerfile]#ls
build.sh  Dockerfile

[root@ubuntu1804 dockerfile]#./build.sh v2.0
Sending build context to Docker daemon  4.096kB
Step 1/5 : FROM busybox
 ---> c7c37e472d31
Step 2/5 : ENV ROOT /data/website
 ---> Using cache
 ---> ad654b3213c7
Step 3/5 : RUN mkdir -p ${ROOT} && echo '<h1> Busybox httpd server in
Dockerfile</h1>' > ${ROOT}/index.html
 ---> Running in c7c2b920ccb0
Removing intermediate container c7c2b920ccb0
 ---> d40ce17d4e85
Step 4/5 : CMD ["/bin/sh","-c","/bin/httpd -f -h ${ROOT}"]
 ---> Running in 52be24840eee
Removing intermediate container 52be24840eee
 ---> d22731d921df
Step 5/5 : EXPOSE 80
 ---> Running in 4a50d8d9aa83
Removing intermediate container 4a50d8d9aa83
 ---> 85d6547a3168
Successfully built 85d6547a3168
Successfully tagged test:v2.0

[root@ubuntu1804 dockerfile]#docker run  -d  --rm -P  --name c2  test:v2.0
cf5fea09ce1362f18a90f4fcdffa3000de26b4919927e1cdce0cce6243f5dd24
[root@ubuntu1804 dockerfile]#docker port c2
80/tcp -> 0.0.0.0:32785
[root@ubuntu1804 dockerfile]#curl 127.0.0.1:32785
<h1> Busybox httpd server in Dockerfile</h1>


[root@ubuntu1804 dockerfile]#docker inspect -f "{{.Config}}" test:v2.0
{   false false false map[80/tcp:{}] false false false
[PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
ROOT=/data/website] [/bin/sh -c /bin/httpd -f -h ${ROOT}] <nil> false
sha256:d22731d921df65f9da7452e714cf35ef345529ffa1bd5b92c0024d1d1bd8e714 map[]
[] false  [] map[]  <nil> []}

#查看进程关系
[root@ubuntu1804 ~]#docker exec -it c2 sh
/ # ps
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/httpd -f -h /data/website
    8 root      0:00 sh
   13 root      0:00 ps
```

2.3.1.5.8 ENTRYPOINT: 入口点

功能类似于CMD，配置容器启动后执行的命令及参数

```
# 使用 exec 执行
ENTRYPOINT ["executable", "param1", "param2"]

# shell中执行
ENTRYPOINT command param1 param2
```

- ENTRYPOINT 不能被 docker run 提供的参数覆盖，而是追加,即如果docker run 命令有参数，那么参数全部都会作为ENTRYPOINT的参数
- 如果docker run 后面没有额外参数，但是dockerfile中的CMD里有（即上面CMD的第三种用法），即Dockerfile中即有CMD也有ENTRYPOINT,那么CMD的全部内容会作为ENTRYPOINT的参数
- 如果docker run 后面有额外参数，同时Dockerfile中即有CMD也有ENTRYPOINT,那么docker run 后面的参数覆盖掉CMD参数内容,最终作为ENTRYPOINT的参数
- 可以通过docker run --entrypoint string 参数在运行时替换,注意string不要加空格
- 使用CMD要在运行时重新写命令本身,然后在后面才能追加运行参数，ENTRYPOINT则可以运行时无需重写命令就可以直接接受新参数
- 每个 Dockerfile 中只能有一个 ENTRYPOINT，当指定多个时，只有最后一个生效

范例:

```
[root@ubuntu1804 ~]#docker run -it  --entrypoint cat alpine /etc/issue
Welcome to Alpine Linux 3.12
Kernel \r on an \m (\l)
```

范例:

```
[root@ubuntu1804 dockerfile]#cat Dockerfile
FROM ubuntu:18.04
RUN apt update \
&& apt -y install  curl \
&& rm -rf /var/lib/apt/lists/*
ENTRYPOINT [ "curl", "-s","https://ip.cn"]


[root@centos8 dockerfile]#podman run -it --rm f68e006
{"ip": "111.199.187.36", "country": "北京市", "city": "联通"}

#追加-i参数
[root@centos8 dockerfile]#podman run -it --rm f68e006  -i
HTTP/2 200
date: Sun, 23 Feb 2020 08:05:19 GMT
content-type: application/json; charset=UTF-8
set-cookie: __cfduid=d4a22496ea6f3b2861763354f8ca600711582445119; expires=Tue,
24-Mar-20 08:05:19 GMT; path=/; domain=.ip.cn; HttpOnly; SameSite=Lax
cf-cache-status: DYNAMIC
expect-ct: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-
cgi/beacon/expect-ct"
alt-svc: h3-25=":443"; ma=86400, h3-24=":443"; ma=86400, h3-23=":443"; ma=86400
```

```
server: cloudflare
cf-ray: 5697b1ac1862eb41-LAX

{"ip": "111.199.187.36", "country": "北京市", "city": "联通"}
```

范例: 利用脚本实现指定环境变量动态生成配置文件内容

```
[root@ubuntu1804 ~]#echo 'Nginx Website in Dockerfile' > index.html
[root@ubuntu1804 ~]#cat Dockerfile
FROM nginx:1.16-alpine
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
ENV DOC_ROOT='/data/website/'
ADD index.html ${DOC_ROOT}
ADD entrypoint.sh /bin/
EXPOSE 80/tcp 8080

#HEALTHCHECK --start-period=3s CMD wget -O - -q http://${IP:-0.0.0.0}:
{PORT:-80}/

CMD ["/usr/sbin/nginx","-g", "daemon off;"]  #CMD指令的内容都成为了ENTRYPOINT的参数
ENTRYPOINT [ "/bin/entrypoint.sh"]

[root@ubuntu1804 ~]#cat entrypoint.sh
#!/bin/sh
cat > /etc/nginx/conf.d/www.conf <<EOF
server {
    server_name ${HOSTNAME};
    listen ${IP:-0.0.0.0}:${PORT:-80};
    root   ${DOC_ROOT:-/usr/share/nginx/html};
}
EOF
exec "$@"
[root@ubuntu1804 ~]#chmod +x entrypoint.sh
[root@ubuntu1804 ~]#docker build -t nginx:v1.0 .
[root@ubuntu1804 ~]#docker run --name n1 --rm -P -e "PORT=8080" -e
"HOSTNAME=www.magedu.org" nginx:v1.0
```

2.3.1.5.9 ARG: 构建参数

ARG指令在build 阶段指定变量,和ENV不同的是，容器运行时不会存在这些环境变量

```
ARG <name>[=<default value>]
```

如果和ENV同名，ENV覆盖ARG变量

可以用 docker build --build-arg <参数名>=<值> 来覆盖

范例:

```
[root@ubuntu1804 ~]#cat Dockerfile
FROM busybox
ARG author="wang <root@wangxiaochun.com>"
LABEL maintainer="${author}"

[root@ubuntu1804 ~]#docker build --build-arg author="29308620@qq.com"  -t
busybox:v1.0 .
```

说明: ARG 和 FROM

```
#FROM指令支持由第一个FROM之前的任何ARG指令声明的变量

#示例:
ARG  CODE_VERSION=latest
FROM base:${CODE_VERSION}
CMD  /code/run-app

FROM extras:${CODE_VERSION}
CMD  /code/run-extras


#在FROM之前声明的ARG在构建阶段之外，所以它不能在FROM之后的任何指令中使用。 要使用在第一个FROM
之前声明的ARG的默认值，请在构建阶段内使用没有值的ARG指令

#示例:
ARG VERSION=latest
FROM busybox:$VERSION
ARG VERSION
RUN echo $VERSION > image_version
```

2.3.1.5.11 VOLUME: 匿名卷

在容器中创建一个可以从本地主机或其他容器挂载的挂载点，一般用来存放数据库和需要保持的数据等，一般会将宿主机上的目录挂载至VOLUME 指令指定的容器目录。即使容器后期被删除，此宿主机的目录仍会保留，从而实现容器数据的持久保存。

宿主机目录为

```
/var/lib/docker/volumes/<volume_id>/_data
```

语法:

```
VOLUME <容器内路径>
VOLUME ["<容器内路径1>", "<容器内路径2>"...]
```

注意:

- Dockerfile中的VOLUME实现的是匿名数据卷,无法指定宿主机路径和容器目录的挂载关系
- 通过docker rm -fv <容器ID> 可以删除容器的同时删除VOLUME指定的卷

范例: 在容器创建两个/data/ ,/data2的挂载点

```
VOLUME [ "/data1","/data2" ]
```

范例:

```
[root@centos8 ~]#cat /data/dockerfile/system/alpine/Dockerfile
FROM alpine:3.11
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
COPY repositories /etc/apk/repositories
VOLUME [ "/testdata","/testdata2" ]

[root@centos8 alpine]#podman run -it --rm 8ef61dd3959da3f sh
/ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
overlay              104806400   3656380 101150020   3% /
tmpfs                    65536         0     65536   0% /dev
/dev/sda2            104806400   3656380 101150020   3% /testdata2
/dev/sda2            104806400   3656380 101150020   3% /testdata
/ # cp /etc/issue  /testdata/f1.txt
/ # cp /etc/issue  /testdata2/f2.txt

[root@centos8 ~]#tree /var/lib/containers/storage/volumes/
/var/lib/containers/storage/volumes/
├── 725f0f67921bdbffbe0aaf9b015d663a6e3ddd24674990d492025dfcf878529b
|   └── _data
|       └── f1.txt
└── fbd13e5253deb375e0dea917df832d2322e96b04ab43bae061584dcdbe7e89f2
    └── _data
        └── f2.txt

4 directories, 2 files
```

2.3.1.5.12 EXPOSE: 暴露端口

指定服务端的容器需要对外暴露(监听)的端口号，以实现容器与外部通信。

EXPOSE 仅仅是声明容器打算使用什么端口而已,并不会真正暴露端口,即不会自动在宿主进行端口映射

因此，在启动容器时需要通过 -P 或-p ， Docker 主机才会真正分配一个端口转发到指定暴露的端口才可使用

注意: 即使 Dockerfile没有EXPOSE 端口指令,也可以通过docker run -p 临时暴露容器内程序真正监听的端口,所以EXPOSE 相当于指定默认的暴露端口,可以通过docker run -P 进行真正暴露

```
EXPOSE <port>[/ <protocol>] [<port>[/ <protocol>] ..]

#说明
<protocol>用于指定传输层协议，可为tcp或udp二者之一，默认为TCP协议
```

范例:

```
EXPOSE 80 443
EXPOSE 11211/udp 11211/tcp
```

范例:

```
[root@ubuntu1804 dockerfile]#pwd
/data/dockerfile
[root@ubuntu1804 dockerfile]#echo Website in Dockerfile > index.html

[root@ubuntu1804 dockerfile]#vim Dockerfile
```

```
FROM busybox
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
COPY index.html /data/website/
EXPOSE 80

[root@ubuntu1804 dockerfile]#cat build.sh
#!/bin/bash
#
TAG=$1
docker build -t test:$TAG .
[root@ubuntu1804 dockerfile]#chmod +x build.sh
[root@ubuntu1804 dockerfile]#./build.sh v1.0
[root@ubuntu1804 dockerfile]#ls
build.sh  Dockerfile  index.html

[root@ubuntu1804 dockerfile]#docker run  --rm -P --name c1 test:v1.0  /bin/httpd
-f  -h /data/website

[root@ubuntu1804 ~]#docker port c1
80/tcp -> 0.0.0.0:32773
[root@ubuntu1804 ~]#curl 127.0.0.1:32773
Website in Dockerfile
[root@ubuntu1804 ~]#docker kill c1
c1
```

2.3.1.5.13 WORKDIR: 指定工作目录

为后续的 RUN、CMD、ENTRYPOINT 指令配置工作目录，当容器运行后，进入容器内WORKDIR指定的默认目录

WORKDIR 指定工作目录（或称当前目录），以后各层的当前目录就被改为指定的目录，如该目录不存在，WORKDIR 会自行创建

```
WORKDIR /path/to/workdir
```

范例:

```
#两次RUN独立运行,不在同一个目录,
RUN cd /app
RUN echo "hello" > world.txt

#如果想实现相同目录可以使用WORKDIR
WORKDIR /app
RUN echo "hello" > world.txt
```

可以使用多个 WORKDIR 指令，后续命令如果参数是相对路径，则会基于之前命令指定的路径。例如

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

则最终路径为 /a/b/c

2.3.1.5.14 ONBUILD: 子镜像引用父镜像的指令

可以用来配置当构建当前镜像的子镜像时，会自动触发执行的指令,但在当前镜像构建时,并不会执行,即延迟到子镜像构建时才执行

```
ONBUILD [INSTRUCTION]
```

例如，Dockerfile 使用如下的内容创建了镜像 image-A。

```
...
ONBUILD ADD http://www.magedu.com/wp-content/uploads/2017/09/logo.png /data/
ONBUILD RUN rm -rf /*
ONBUILD RUN /usr/local/bin/python-build --dir /app/src...
```

如果基于 image-A 创建新的镜像image-B时，新的Dockerfile中使用 FROM image-A指定基础镜像时，会自动执行ONBUILD 指令内容，等价于在后面添加了两条指令。

```
FROM image-A

#Automatically run the following
ADD http://www.magedu.com/wp-content/uploads/2017/09/logo.png /data
RUN /usr/local/bin/python-build --dir /app/src
```

说明:

- 尽管任何指令都可注册成为触发器指令，但ONBUILD不能自我能套，且不会触发FROM和MAINTAINER指令
- 使用 `ONBUILD` 指令的镜像，推荐在标签中注明，例如 `ruby:1.9-onbuild`

2.3.1.5.15 USER: 指定当前用户

指定运行容器时的用户名或 UID，后续的 RUN 也会使用指定用户

当服务不需要管理员权限时，可以通过该命令指定运行用户

这个用户必须是事先建立好的，否则无法切换

如果没有指定 USER,默认是 root 身份执行

```
USER <user>[:<group>]
USER <UID>[:<GID>]
```

范例:

```
RUN groupadd -r mysql && useradd -r -g mysql mysql
USER mysql
```

2.3.1.5.16 HEALTHCHECK: 健康检查

检查容器的健康性

```
HEALTHCHECK [选项] CMD <命令> #设置检查容器健康状况的命令
HEALTHCHECK NONE #如果基础镜像有健康检查指令，使用这行可以屏蔽掉其健康检查指令

HEALTHCHECK 支持下列选项:
--interval=<间隔>  #两次健康检查的间隔，默认为 30 秒
--timeout=<时长>    #健康检查命令运行超时时间，如果超过这个时间，本次健康检查就被视为失败，默认 30 秒
--retries=<次数>    #当连续失败指定次数后，则将容器状态视为 unhealthy，默认3次
--start-period=<FDURATION> #default: 0s

#检查结果返回值:
0  #success    the container is healthy and ready for use
1  #unhealth   the container is not working correctly
2  #reserved   do not use this exit code
```

范例

```
FROM nginx
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/*
HEALTHCHECK --interval=5s --timeout=3s \
CMD curl -fs http://localhost/ || exit 1
```

2.3.1.5.17 STOPSIGNAL: 退出容器的信号

该 `STOPSIGNAL` 指令设置将被发送到容器退出的系统调用信号。该信号可以是与内核syscall表中的位置匹配的有效无符号数字（例如9），也可以是SIGNAME格式的信号名称（例如SIGKILL）

```
STOPSIGNAL signal
```

2.3.1.5.18 SHELL : 指定shell

SHELL指令允许覆盖用于命令的shell形式的默认SHELL, 必须在Dockerfile中以JSON形式编写SHELL指令。

```
SHELL ["executable", "parameters"]
```

在Linux上默认SHELL程序为[“/bin/sh”，“-c”]，在Windows上，默认SHELL程序为[“cmd”，“/S”，“/C”]。

SHELL指令在Windows上特别有用，在Windows上有两个常用且完全不同的本机SHELL:cmd和powershell，以及包括sh在内的备用shell。

SHELL指令可以出现多次。 每个SHELL指令将覆盖所有先前的SHELL指令，并影响所有后续的指令

```
FROM microsoft/windowsservercore

# Executed as cmd /S /C echo default
RUN echo default

# Executed as cmd /S /C powershell -command Write-Host default
RUN powershell -command Write-Host default

# Executed as powershell -command Write-Host hello
SHELL ["powershell", "-command"]
RUN Write-Host hello
```

```
# Executed as cmd /S /C echo hello
SHELL ["cmd", "/S", "/C"]
RUN echo hello
```

2.3.1.5.18 .dockerignore文件

官方文档: https://docs.docker.com/engine/reference/builder/#dockerignore-file
与.gitignore文件类似，生成构建上下文时Docker客户端应忽略的文件和文件夹指定模式

.dockerignore 使用 Go 的文件路径规则 filepath.Match

参考链接: https://golang.org/pkg/path/filepath/#Match

完整的语法

```
#       #以#开头的行为注释
*       #匹配任何非分隔符字符序列
?       #匹配任何单个非分隔符
\\      #表示  \

**      #匹配任意数量的目录（包括零）例如，**/*.go将排除在所有目录中以.go结尾的所有文件，包括构建上下文的根。
!       #表示取反，可用于排除例外情况
```

Rule	Behavior
`# comment`	Ignored.
`*/temp*`	Exclude files and directories whose names start with `temp` in any immediate subdirectory of the root. For example, the plain file `/somedir/temporary.txt` is excluded, as is the directory `/somedir/temp`.
`*/*/temp*`	Exclude files and directories starting with `temp` from any subdirectory that is two levels below the root. For example, `/somedir/subdir/temporary.txt` is excluded.
`temp?`	Exclude files and directories in the root directory whose names are a one-character extension of `temp`. For example, `/tempa` and `/tempb` are excluded.

范例:

```
#排除 test 目录下的所有文件
test/*
#排除 md 目录下的 xttblog.md 文件
md/xttblog.md
#排除 xttblog 目录下的所有 .md 的文件
xttblog/*.md
#排除以 xttblog 为前缀的文件和文件夹
xttblog?
#排除所有目录下的 .sql 文件夹
**/*.sql
```

范例:

```
#除了README的md不排外，排除所有md文件，但不排除README-secret.md
*.md
!README*.md
README-secret.md


#除了所有README的md文件以外的md都排除
*.md
README-secret.md
!README*.md
```

2.3.1.5.19 Dockerfile 构建过程和指令总结

Dockerfile 构建过程

- 从基础镜像运行一个容器
- 执行一条指令，对容器做出修改
- 执行类似docker commit的操作，提交一个新的中间镜像层(可以利用中间层镜像创建容器进行调试和排错)
- 再基于刚提交的镜像运行一个新容器
- 执行Dockerfile中的下一条指令，直至所有指令执行完毕

Dockerfile 指令总结

BUILD	RUN	BOTH
FROM	CMD	WORKDIR
LABEL	VOLUME	USER
COPY	EXPOSE	ENV
ADD	ENTRYPOINT	
RUN		
ONBUILD		
.dockerignore		

2.3.1.6 构建镜像docker build 命令

docker build命令使用Dockerfile文件创建镜像

```
docker build [OPTIONS] PATH | URL | -

说明:
PATH | URL | -      #可以使是本地路径，也可以是URL路径。若设置为 - ，则从标准输入获取Dockerfile的内容
-f, --file string  #Dockerfile文件名,默认为 PATH/Dockerfile
--force-rm         #总是删除中间层容器,创建镜像失败时，删除临时容器
--no-cache         #不使用之前构建中创建的缓存
-q  --quiet=false  #不显示Dockerfile的RUN运行的输出结果
--rm=true          #创建镜像成功时，删除临时容器
-t --tag list      #设置注册名称、镜像名称、标签。格式为 <注册名称>/<镜像名称>:<标签>（标签默认为latest）
```

范例:

```
docker build .
docker build /usr/local/src/nginx
docker build -f /path/to/a/Dockerfile .
docker build -t shykes/myapp .
docker build -t shykes/myapp:1.0.2 -t shykes/myapp:latest .
docker build -t test/myapp .
docker build -t nginx:v1 /usr/local/src/nginx
```

查看镜像的构建历史: docker history 镜像ID

范例:

```
[root@centos8 ~]#podman history  90201858b1fc
ID             CREATED          CREATED BY
SIZE      COMMENT
90201858b1fc   17 minutes ago   /bin/sh -c #(nop) CMD ["tail" ,"-f","/etc/...
0B
<missing>      2 hours ago      /bin/sh -c apt-get update && apt-get insta...
14.83MB
<missing>      35 hours ago     /bin/sh -c #(nop) CMD ["/bin/bash"]
14.83MB
<missing>      35 hours ago     /bin/sh -c mkdir -p /run/systemd && echo '...
3.072kB
<missing>      35 hours ago     /bin/sh -c set -xe && echo '#!/bin/sh' > /...
15.87kB
<missing>      35 hours ago     /bin/sh -c [ -z "$(apt-get indextargets)" ]
991.2kB
<missing>      35 hours ago     /bin/sh -c #(nop) ADD file:91a750fb184711f...
65.58MB
```

范例：利用Dockerfile构建基于CentOS的nginx镜像

```
[root@ubuntu1804 ~]#vim /data/Dockerfile
[root@ubuntu1804 ~]#cat /data/Dockerfile
FROM centos
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
RUN yum install -y nginx && echo Nginx Website in Docker >
/usr/share/nginx/html/index.html
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
#ENTRYPOINT ["nginx", "-g", "daemon off;"]

[root@ubuntu1804 ~]#docker build -t nginx_centos8.2:v1.14.1 /data/
Sending build context to Docker daemon  209.2MB
Step 1/6 : FROM centos
 ---> 831691599b88
Step 2/6 : LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
 ---> Running in 3fc4487e80f9
Removing intermediate container 3fc4487e80f9
 ---> 598318841b8a
Step 3/6 : RUN yum install -y nginx
 ---> Running in 8a6d9866e4ae
CentOS-8 - AppStream                            2.1 MB/s | 5.8 MB     00:02
CentOS-8 - Base                                 1.8 MB/s | 2.2 MB     00:01
CentOS-8 - Extras                               8.6 kB/s | 7.0 kB     00:00
.......
Complete!
Removing intermediate container 8a6d9866e4ae
 ---> 8963fb608c33
Step 4/6 : RUN echo Nginx Website in Docker > /usr/share/nginx/html/index.html
 ---> Running in 04d4287aac49
Removing intermediate container 04d4287aac49
 ---> 9a95e56b9bc0
Step 5/6 : EXPOSE 80
 ---> Running in 8534523d8aa6
Removing intermediate container 8534523d8aa6
 ---> 23cca5737903
Step 6/6 : CMD ["nginx", "-g", "daemon off;"]
```

```
 ---> Running in d52fcc21444f
Removing intermediate container d52fcc21444f
 ---> afdaec99eb35
Successfully built afdaec99eb35
Successfully tagged nginx_centos8.2:v1.14.1

[root@ubuntu1804 ~]#docker run -d -P --name nginx-web nginx_centos8.2:v1.14.1
3faba5a49f63ab3a1b031da5e4940303b10ddf349069184597e703c572d257d8
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                       COMMAND                  CREATED
          STATUS              PORTS                     NAMES
3faba5a49f63        nginx_centos8.2:v1.14.1   "nginx -g 'daemon of…"   4 seconds
ago       Up 3 seconds        0.0.0.0:32775->80/tcp   nginx-web
[root@ubuntu1804 ~]#curl http://127.0.0.1/32775
curl: (7) Failed to connect to 127.0.0.1 port 80: Connection refused
[root@ubuntu1804 ~]#curl http://127.0.0.1:32775
Nginx Website in Docker
[root@ubuntu1804 ~]#curl -I http://127.0.0.1:32775
HTTP/1.1 200 OK
Server: nginx/1.14.1
Date: Wed, 22 Jul 2020 17:09:15 GMT
Content-Type: text/html
Content-Length: 24
Last-Modified: Wed, 22 Jul 2020 17:04:40 GMT
Connection: keep-alive
ETag: "5f1871a8-18"
Accept-Ranges: bytes
```

范例: 刷新镜像缓存重新构建新镜像

```
[root@ubuntu1804 ~]#cat /data/Dockerfile
FROM centos
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
RUN yum install -y nginx
RUN echo Nginx Website in Docker > /usr/share/nginx/html/index.html
#修改下面行,从下面行开始不再使用缓存
ENV REFRESH_DATA 2020-01-01
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

[root@ubuntu1804 ~]#docker build   -t nginx_centos8.2:v1.14.1 /data/
Sending build context to Docker daemon  209.2MB
Step 1/7 : FROM centos
 ---> 831691599b88
Step 2/7 : LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
 ---> Using cache
 ---> 598318841b8a
Step 3/7 : RUN yum install -y nginx
 ---> Using cache
 ---> 8963fb608c33
Step 4/7 : RUN echo Nginx Website in Docker > /usr/share/nginx/html/index.html
 ---> Using cache
 ---> 9a95e56b9bc0
Step 5/7 : ENV REFRESH_DATA 2020-01-01  #从此行开始不再利用缓存
 ---> Running in 4607ee0d0e77
Removing intermediate container 4607ee0d0e77
 ---> d6235889f336
```

```
Step 6/7 : EXPOSE 80
 ---> Running in 6924aab5c5c8
Removing intermediate container 6924aab5c5c8
 ---> 545393760683
Step 7/7 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in 345bbc6179d8
Removing intermediate container 345bbc6179d8
 ---> 4bafc2d0c7e0
Successfully built 4bafc2d0c7e0
Successfully tagged nginx_centos8.2:v1.14.1


#全部不利用缓存重新构建镜像
[root@ubuntu1804 ~]#docker build  --no-cache -t nginx_centos8.2:v1.14.1 /data/
Sending build context to Docker daemon  209.2MB
Step 1/7 : FROM centos
 ---> 831691599b88
Step 2/7 : LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
 ---> Running in 41f2aab6657f
Removing intermediate container 41f2aab6657f
 ---> 091969d0ed9e
Step 3/7 : RUN yum install -y nginx
 ---> Running in 6e174d492348
CentOS-8 - AppStream                            4.2 MB/s | 5.8 MB     00:01
CentOS-8 - Base                                 1.7 MB/s | 2.2 MB     00:01
CentOS-8 - Extras                               1.2 kB/s | 7.0 kB     00:05
Dependencies resolved.
......

Complete!
Removing intermediate container 6e174d492348
 ---> ba62ac34b951
Step 4/7 : RUN echo Nginx Website in Docker > /usr/share/nginx/html/index.html
 ---> Running in d6f785b28ef6
Removing intermediate container d6f785b28ef6
 ---> 6e15fdc84e21
Step 5/7 : ENV REFRESH_DATA 2020-06-06
 ---> Running in c6fd87ed95f6
Removing intermediate container c6fd87ed95f6
 ---> 328b8621ec36
Step 6/7 : EXPOSE 80
 ---> Running in 1af3d6964d81
Removing intermediate container 1af3d6964d81
 ---> 7c513643b182
Step 7/7 : CMD ["nginx", "-g", "daemon off;"]
 ---> Running in fd9216490941
Removing intermediate container fd9216490941
 ---> 0b2b61dd0445
Successfully built 0b2b61dd0445
Successfully tagged nginx_centos8.2:v1.14.1
```

2.3.2 实战案例: Dockerfile 制作基于基础镜像的Base镜像

2.3.2.1 准备目录结构，下载镜像并初始化系统

```
#按照业务类型或系统类型等方式划分创建目录环境，方便后期镜像比较多的时候进行分类
[root@ubuntu1804 ~]#mkdir
/data/dockerfile/{web/{nginx,apache,tomcat,jdk},system/{centos,ubuntu,alpine,deb
ian}} -p
[root@ubuntu1804 ~]#tree /data/dockerfile/
/data/dockerfile/
├── system
|   ├── alpine
|   ├── centos
|   ├── debian
|   └── ubuntu
└── web
    ├── apache
    ├── jdk
    ├── nginx
    └── tomcat

10 directories, 0 files
[root@ubuntu1804 ~]#

#下载基础镜像
[root@ubuntu1804 ~]#docker pull centos:centos7.7.1908
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.3.2.2 先制作基于基础镜像的系统Base镜像

```
#先制作基于基础镜像的系统base镜像
[root@ubuntu1804 ~]#cd /data/dockerfile/system/centos/
#创建Dockerfile，注意可以是dockerfile，但无语法着色功能

[root@ubuntu1804 centos]#vim Dockerfile
[root@ubuntu1804 centos]#cat Dockerfile
FROM centos:centos7.7.1908

LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"

RUN yum -y install wget && rm -f /etc/yum.repos.d/* && wget -P /etc/yum.repos.d/
http://mirrors.aliyun.com/repo/Centos-7.repo \
    && wget -P /etc/yum.repos.d/ http://mirrors.aliyun.com/repo/epel-7.repo \
    && yum -y install  vim-enhanced tcpdump lrzsz tree telnet bash-completion
net-tools wget curl bzip2 lsof  zip unzip nfs-utils gcc make gcc-c++ glibc
glibc-devel pcre pcre-devel openssl  openssl-devel systemd-devel zlib-devel \
    && yum clean all \
    && rm -f /etc/localtime  \
    && ln -s ../usr/share/zoneinfo/Asia/Shanghai /etc/localtime

[root@ubuntu1804 centos]#vim build.sh
[root@ubuntu1804 centos]#cat build.sh
#!/bin/bash
#
docker build -t centos7-base:v1  .
[root@ubuntu1804 centos]#chmod +x build.sh
```

```
[root@ubuntu1804 centos]#./build.sh
[root@ubuntu1804 centos]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos7-base        v1                  1ba1317e06dc        23 seconds ago
402MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
[root@ubuntu1804 centos]#docker image history centos7-base:v1
IMAGE               CREATED             CREATED BY
       SIZE                COMMENT
1ba1317e06dc        43 seconds ago      /bin/sh -c yum -y install wget && rm -f
/etc…   198MB
6b87f2843eb9        About an hour ago   /bin/sh -c #(nop)  LABEL
maintainer=wangxiao…   0B
08d05d1d5859        2 months ago        /bin/sh -c #(nop)  CMD ["/bin/bash"]
       0B
<missing>           2 months ago        /bin/sh -c #(nop)  LABEL org.label-
schema.sc…   0B
<missing>           2 months ago        /bin/sh -c #(nop) ADD
file:3e2a127b44ed01afc…   204MB
```

2.3.3 实战案例: Dockerfile 制作基于Base镜像的 nginx 镜像

2.3.3.1 在Dockerfile目录下准备编译安装的相关文件

```
[root@ubuntu1804 ~]#mkdir /data/dockerfile/web/nginx/1.16
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16
[root@ubuntu1804 1.16]#wget http://nginx.org/download/nginx-1.16.1.tar.gz
[root@ubuntu1804 1.16]#mkdir app/
[root@ubuntu1804 1.16]#echo  "Test Page in app" > app/index.html
[root@ubuntu1804 1.16]#tar zcf app.tar.gz app
[root@ubuntu1804 1.16]#ls
app  app.tar.gz nginx-1.16.1.tar.gz
```

2.3.3.2 在一台测试机进行编译安装同一版本的nginx 生成模版配置文件

```
[root@centos7 ~]#yum -y install vim-enhanced tcpdump lrzsz tree telnet bash-
completion net-tools wget bzip2 lsof tmux man-pages zip unzip nfs-utils gcc make
gcc-c++ glibc glibc-devel pcre pcre-devel openssl  openssl-devel systemd-devel
zlib-devel
[root@centos7 ~]#wget -P /usr/local/src http://nginx.org/download/nginx-
1.16.1.tar.gz
[root@centos7 ~]#cd /usr/local/src/
[root@centos7 src]#tar xvf nginx-1.16.1.tar.gz
[root@centos7 src]#cd nginx-1.16.1/
[root@centos7 nginx-1.16.1]#./configure  --prefix=/apps/nginx && make && make
install

#将配置文件复制到nginx镜像的服务器相应目录下
[root@centos7 ~]#scp /apps/nginx/conf/nginx.conf
10.0.0.100:/data/dockerfile/web/nginx/1.16

#准备配置文件
[root@ubuntu1804 1.16]#vim /data/dockerfile/web/nginx/1.16/nginx.conf
worker_processes  1;
```

```
user nginx;
daemon off;   #增加此行,前台运行nginx
```

2.3.3.3 编写Dockerfile文件

```
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx
[root@ubuntu1804 nginx]#vim  Dockerfile
[root@ubuntu1804 nginx]#cat Dockerfile
FROM centos7-base:v1

LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"

ADD nginx-1.16.1.tar.gz /usr/local/src

RUN cd /usr/local/src/nginx-1.16.1 && \
    && ./configure --prefix=/apps/nginx \
    && make && make install \
    && rm -f /usr/local/src/nginx* \
    && useradd -r nginx

COPY nginx.conf /apps/nginx/conf/

ADD  app.tar.gz /apps/nginx/html/

EXPOSE 80 443

CMD ["/apps/nginx/sbin/nginx"]
[root@ubuntu1804 nginx]#
```

2.3.3.4 生成nginx镜像

```
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16
[root@ubuntu1804 1.16]#ls
app  app.tar.gz  build.sh  Dockerfile  nginx-1.16.1.tar.gz  nginx.conf
[root@ubuntu1804 1.16]#vim build.sh
[root@ubuntu1804 1.16]#cat build.sh
#!/bin/bash
#
docker build -t nginx-centos7:1.6.1 .
[root@ubuntu1804 1.16]#chmod +x build.sh
[root@ubuntu1804 1.16]#./build.sh
[root@ubuntu1804 1.16]##docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx-centos7       1.6.1               73e4b4b95bca        10 minutes ago
412MB
centos7-base        v1                  1ba1317e06dc        About an hour ago
402MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.3.3.5 生成的容器测试镜像

```
[root@ubuntu1804 ~]#docker run  -d -p 80:80  nginx-centos7:1.6.1
e8e733c6dc96bfb212a15dec04cfcfcac72daf400f5d2423c707aeb778a1859d
```

```
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED
      STATUS              PORTS                           NAMES
e8e733c6dc96        centos7-nginx:1.6.1    "/apps/nginx/sbin/ng…"   4 seconds ago
      Up 2 seconds        0.0.0.0:80->80/tcp, 443/tcp   cool_germain
[root@ubuntu1804 ~]#docker exec -it e8e733c6dc96 bash
[root@e8e733c6dc96 /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.2  0.2  20572  2468 ?        Ss   03:36   0:00 nginx: master
process /apps/nginx/sbin/nginx
nginx        12  0.0  0.2  21024  2344 ?        S    03:36   0:00 nginx: worker
process
root         13  4.0  0.3  12364  3536 pts/0    Ss   03:37   0:00 bash
root         32  0.0  0.3  51764  3460 pts/0    R+   03:37   0:00 ps aux
[root@e8e733c6dc96 /]# exit
exit
[root@ubuntu1804 ~]#curl 127.0.0.1/app/
Test Page in app
```

2.3.4 实战案例: Dockerfile 直接制作 nginx 镜像

2.3.4.1 在Dockerfile目录下准备编译安装的相关文件

```
[root@ubuntu1804 ~]#mkdir /data/dockerfile/web/nginx/1.16.1
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16.1
[root@ubuntu1804 1.16.1]#vim nginx.conf
user  nginx;
worker_processes  1;
#daemon off;
[root@ubuntu1804 1.16.1]#wget http://nginx.org/download/nginx-1.16.1.tar.gz
```

2.3.4.2 编写Dockerfile文件

```
[root@ubuntu1804 1.16.1]#pwd
/data/dockerfile/web/nginx/1.16.1
[root@ubuntu1804 1.16.1]#vim Dockerfile
[root@ubuntu1804 1.16.1]#cat Dockerfile
#Nginx Dockerfile

FROM centos:centos7.7.1908

MAINTAINER wangxiaochun <root@wangxiaochun.com>

RUN yum install -y  gcc gcc-c++  pcre pcre-devel zlib zlib-devel openssl
openssl-devel \
    && useradd -r -s /sbin/nologin nginx \
    && yum clean all

ADD nginx-1.16.1.tar.gz /usr/local/src/

RUN cd /usr/local/src/nginx-1.16.1 \
    && ./configure --prefix=/apps/nginx \
    && make \
    && make install \
    && rm -rf /usr/local/src/nginx*
```

```
ADD nginx.conf /apps/nginx/conf/nginx.conf

COPY index.html /apps/nginx/html/

RUN ln -s /apps/nginx/sbin/nginx /usr/sbin/nginx

EXPOSE 80 443

CMD ["nginx","-g","daemon off;"]
```

2.3.4.3 生成nginx镜像

```
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16.1
[root@ubuntu1804 1.16.1]#vim build.sh
[root@ubuntu1804 1.16.1]#cat build.sh
#!/bin/bash
#
docker build -t nginx-centos7:1.6.1-v2 .
[root@ubuntu1804 1.16.1]#chmod +x build.sh
[root@ubuntu1804 1.16.1]#ls
build.sh  Dockerfile  index.html  nginx-1.16.1.tar.gz  nginx.conf
[root@ubuntu1804 1.16.1]#./build.sh
[root@ubuntu1804 1.16.1]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx-centos7       1.6.1-v2            1918d29d5f45        17 minutes ago
328MB
nginx-centos7       1.6.1               8c16774437a5        13 hours ago
412MB
centos7-base        v1                  1ba1317e06dc        15 hours ago
402MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.3.4.4 生成容器测试镜像

```
[root@ubuntu1804 ~]#docker run  -d -p 80:80  nginx-centos7:1.6.1-v2
21c954ad4fb902076832cc9a52dd1502aca43d9bcd2b46a2f164382e4ac7b3f6
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                      COMMAND                  CREATED
          STATUS              PORTS                              NAMES
21c954ad4fb9        centos7-nginx:1.6.1-v2     "nginx -g 'daemon of…"   6 seconds
ago       Up 4 seconds        0.0.0.0:80->80/tcp, 443/tcp        inspiring_goldwasser
[root@ubuntu1804 ~]#curl 127.0.0.1
Test Page v2 in Docker
[root@ubuntu1804 ~]#docker exec -it 21c954ad4fb9 bash
[root@21c954ad4fb9 /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.5  0.2  20572  2372 ?        Ss   03:30   0:00 nginx: master
process nginx -g daemon off;
nginx         6  0.0  0.2  21024  2316 ?        S    03:30   0:00 nginx: worker
process
root          7 11.5  0.2  11840  2880 pts/0    Ss   03:31   0:00 bash
root         20  0.0  0.3  51764  3376 pts/0    R+   03:31   0:00 ps aux
[root@21c954ad4fb9 /]# exit
exit
```

```
[root@ubuntu1804 ~]#
```

2.4 生产案例: 制作自定义tomcat业务镜像

基于官方提供的centos、debian、ubuntu、alpine等基础 镜像构建 JDK (Java环 境)，然后再基于自定义的 JDK 镜像构建出业务需要的tomcat 镜像

2.4.1 自定义 Centos 系统基础镜像

先基于官方提供的基础镜像，制作出安装了常用命令的自定义基础镜像

```
[root@ubuntu1804 ~]#docker pull centos:centos7.7.1908
[root@ubuntu1804 ~]#mkdir -p
/data/dockerfile/{web/{nginx,tomcat,jdk},system/{centos,ubuntu,alpine,debian}}
[root@ubuntu1804 ~]#cd /data/dockerfile/system/centos/
[root@ubuntu1804 centos]#vim Dockerfile
[root@ubuntu1804 centos]#cat Dockerfile
# Centos Base Image
FROM centos:centos7.7.1908
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
RUN yum -y install wget && rm -f /etc/yum.repos.d/* && wget -P /etc/yum.repos.d/
http://mirrors.aliyun.com/repo/Centos-7.repo \
    && wget -P /etc/yum.repos.d/ http://mirrors.aliyun.com/repo/epel-7.repo \
    && yum -y install  vim-enhanced tcpdump lrzsz tree telnet bash-completion
net-tools wget bzip2 lsof  zip unzip nfs-utils gcc make gcc-c++ glibc glibc-
devel pcre pcre-devel openssl  openssl-devel systemd-devel zlib-devel \
    && yum clean all \
    && rm -f /etc/localtime  \
    && ln -s ../usr/share/zoneinfo/Asia/Shanghai /etc/localtime
#添加系统账户
RUN  groupadd www -g 2019 && useradd www -u 2019 -g www

[root@ubuntu1804 centos]#vim build.sh

#通过脚本构建镜像
[root@ubuntu1804 centos]#cat build.sh
#!/bin/bash
docker build -t centos7-base:v1 .

[root@ubuntu1804 centos]#bash build.sh
[root@ubuntu1804 centos]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos7-base        v1                  34ab3afcd3b3        4 seconds ago
403MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.4.2 构建JDK 镜像

2.4.2.1 上传JDK压缩包和profile文件上传到Dockerfile当前目录

```
#将CentOS7主机上的/etc/profile文件传到 Dockerfile 所在目录下
[root@ubuntu1804 ~]#scp centos7:/etc/profile 10.0.0.100:/data/dockerfile/web/jdk

#修改profile文件，加下面四行相关变量
[root@ubuntu1804 ~]#vim /data/dockerfile/web/jdk/profile
[root@ubuntu1804 ~]#tail -n 5 /data/dockerfile/web/jdk/profile

export JAVA_HOME=/usr/local/jdk
export TOMCAT_HOME=/apps/tomcat
export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$TOMCAT_HOME/bin:$PATH
export
CLASSPATH=.$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib:$JAVA_HOME/lib/tools.jar

#下载jdk文件传到Dockfile目录下
#https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-
2133151.html
[root@ubuntu1804 ~]#tree /data/dockerfile/web/jdk
/data/dockerfile/web/jdk
├── jdk-8u212-linux-x64.tar.gz
└── profile

0 directories, 2 files
```

2.4.2.2 准备Dockerfile文件

```
[root@ubuntu1804 ~]#vim  /data/dockerfile/web/jdk/Dockerfile
[root@ubuntu1804 ~]#cat /data/dockerfile/web/jdk/Dockerfile
#JDK Base Image
FROM centos7-base:v1
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
ADD jdk-8u212-linux-x64.tar.gz /usr/local/src/
RUN ln -s /usr/local/src/jdk1.8.0_212 /usr/local/jdk
ADD profile /etc/profile
ENV JAVA_HOME /usr/local/jdk
ENV JRE_HOME $JAVA_HOME/jre
ENV CLASSPATH $JAVA_HOME/lib/:$JRE_HOME/lib/
ENV PATH $PATH:$JAVA_HOME/bin
```

2.4.2.3 执行构建脚本制作镜像

```
[root@ubuntu1804 ~]#vim /data/dockerfile/web/jdk/build.sh
[root@ubuntu1804 ~]#cat /data/dockerfile/web/jdk/build.sh
#!/bin/bash
docker build -t centos7-jdk:8u212 .

[root@ubuntu1804 ~]#tree /data/dockerfile/web/jdk/
/data/dockerfile/web/jdk/
├── build.sh
├── Dockerfile
├── jdk-8u212-linux-x64.tar.gz
└── profile

0 directories, 4 files
[root@ubuntu1804 ~]#cd /data/dockerfile/web/jdk/
[root@ubuntu1804 jdk]#bash build.sh
```

```
[root@ubuntu1804 jdk]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
centos7-jdk         8u212               fdbeb8a49ea6        59 seconds ago
809MB
centos7-base        v1                  34ab3afcd3b3        44 minutes ago
403MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.3.4.4 从镜像启动容器测试

```
[root@ubuntu1804 jdk]#docker run -it --rm centos7-jdk:8u212 bash
[root@25c9c0266bd2 /]# java -version
java version "1.8.0_212"
Java(TM) SE Runtime Environment (build 1.8.0_212-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.212-b10, mixed mode)
```

2.4.3 从JDK镜像构建tomcat 8 Base镜像

基于自定义的 JDK 基础镜像，构建出通用的自定义 Tomcat 基础镜像，此镜像后期会被多个业务的多个服务共同引用(相同的JDK 版本和Tomcat 版本)

2.4.3.1 上传tomcat 压缩包

```
[root@ubuntu1804 ~]#mkdir -p /data/dockerfile/web/tomcat/tomcat-base-8.5.50
[root@ubuntu1804 ~]#cd /data/dockerfile/web/tomcat/tomcat-base-8.5.50
[root@ubuntu1804 tomcat-base-8.5.50]#
wget http://mirrors.tuna.tsinghua.edu.cn/apache/tomcat/tomcat-
8/v8.5.50/bin/apache-tomcat-8.5.50.tar.gz
```

2.4.3.2 编辑Dockerfile

```
[root@ubuntu1804 ~]#cat /data/dockerfile/web/tomcat/tomcat-base-
8.5.50/Dockerfile
#Tomcat Base Image
FROM centos7-jdk:8u212
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
#env
ENV TZ "Asia/Shanghai"
ENV LANG en_US.UTF-8
ENV TERM xterm
ENV TOMCAT_MAJOR_VERSION 8
ENV TOMCAT_MINOR_VERSION 8.5.50
ENV CATALINA_HOME /apps/tomcat
ENV APP_DIR ${CATALINA_HOME}/webapps

RUN mkdir /apps
ADD apache-tomcat-8.5.50.tar.gz /apps
RUN ln -s /apps/apache-tomcat-8.5.50 /apps/tomcat
```

2.4.3.3 通过脚本构建tomcat 基础镜像

```
[root@ubuntu1804 tomcat-base-8.5.50]#vim  build.sh
```

```
[root@ubuntu1804 tomcat-base-8.5.50]#cat build.sh
#!/bin/bash
docker build -t tomcat-base:v8.5.50 .

[root@ubuntu1804 tomcat-base-8.5.50]#tree
.
├── apache-tomcat-8.5.50.tar.gz
├── build.sh
└── Dockerfile

0 directories, 3 files

[root@ubuntu1804 tomcat-base-8.5.50]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
tomcat-base         v8.5.50             8d5395cb72c4        3 seconds ago
824MB
centos7-jdk         8u212               e0fe770a7ccd        22 minutes ago
809MB
centos7-base        v1                  34ab3afcd3b3        2 hours ago
403MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.4.3.4 验证镜像构建完成

```
[root@ubuntu1804 tomcat-base-8.5.50]#docker run -it --rm -p 8080:8080  tomcat-
base:v8.5.50 bash
[root@d0a387e0ccc9 /]# /apps/tomcat/bin/catalina.sh start
Using CATALINA_BASE:   /apps/tomcat
Using CATALINA_HOME:   /apps/tomcat
Using CATALINA_TMPDIR: /apps/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:       /apps/tomcat/bin/bootstrap.jar:/apps/tomcat/bin/tomcat-
juli.jar
Tomcat started.
[root@d0a387e0ccc9 /]# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:8009            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN
[root@d0a387e0ccc9 /]#
```

2.4.4 构建业务镜像1

创建tomcat-app1和tomcat-app2两个目录，代表不同的两个基于tomcat的业务。

2.4.4.1 准备tomcat的配置文件

```
[root@ubuntu1804 ~]#mkdir -p /data/dockerfile/web/tomcat/tomcat-app{1,2}
[root@ubuntu1804 ~]#tree /data/dockerfile/web/tomcat/
/data/dockerfile/web/tomcat/
├── tomcat-app1
├── tomcat-app2
└── tomcat-base-8.5.50
    ├── apache-tomcat-8.5.50.tar.gz
```

```
    ├── build.sh
    └── Dockerfile

3 directories, 3 files

#上传和修改server.xml
[root@ubuntu1804 ~]#cd /data/dockerfile/web/tomcat/tomcat-base-8.5.50
[root@ubuntu1804 tomcat-base-8.5.50]#tar xf apache-tomcat-8.5.50.tar.gz
[root@ubuntu1804 tomcat-base-8.5.50]#cp apache-tomcat-8.5.50/conf/server.xml
/data/dockerfile/web/tomcat/tomcat-app1/
[root@ubuntu1804 tomcat-base-8.5.50]#cd /data/dockerfile/web/tomcat/tomcat-app1/
[root@ubuntu1804 tomcat-app1]#vim server.xml
......
 <Host name="localhost"  appBase="/data/tomcat/webapps"

            unpackWARs="true" autoDeploy="true">
......
```

```
<Host name="localhost"  appBase="/data/tomcat/webapps"  unpackWARs="true" autoDeploy="true">
```

2.4.4.2 准备自定义页面

```
[root@ubuntu1804 tomcat-app1]#mkdir app
[root@ubuntu1804 tomcat-app1]#echo "Tomcat Page in app1" > app/index.jsp
[root@ubuntu1804 tomcat-app1]#tar zcf app.tar.gz app
```

2.4.4.3 准备容器启动执行脚本

```
[root@ubuntu1804 tomcat-app1]#vim run_tomcat.sh
[root@ubuntu1804 tomcat-app1]#cat run_tomcat.sh
#!/bin/bash
echo "nameserver 180.76.76.76" > /etc/resolv.conf
su - www -c "/apps/tomcat/bin/catalina.sh start"
su - www -c "tail -f /etc/hosts"
[root@ubuntu1804 tomcat-app1]#chmod a+x run_tomcat.sh
```

2.4.4.4 准备Dockerfile

```
[root@ubuntu1804 tomcat-app1]#vim Dockerfile
[root@ubuntu1804 tomcat-app1]#cat Dockerfile
#Tomcat Web Image
FROM tomcat-base:v8.5.50
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
ADD server.xml /apps/tomcat/conf/server.xml
ADD run_tomcat.sh /apps/tomcat/bin/run_tomcat.sh
ADD app.tar.gz /data/tomcat/webapps/
RUN chown -R www.www /apps/   /data/tomcat/
EXPOSE 8080  8009
CMD ["/apps/tomcat/bin/run_tomcat.sh"]
```

2.4.4.5 执行构建脚本制作镜像

```
[root@ubuntu1804 tomcat-app1]#vim build.sh
[root@ubuntu1804 tomcat-app1]#cat build.sh
```

```
#!/bin/bash
docker build -t tomcat-web:app1 .

[root@ubuntu1804 tomcat-app1]#pwd
/data/dockerfile/web/tomcat/tomcat-app1
[root@ubuntu1804 tomcat-app1]#tree
.
├── app
│   └── index.jsp
├── app.tar.gz
├── build.sh
├── Dockerfile
├── run_tomcat.sh
└── server.xml

1 directory, 6 files

[root@ubuntu1804 tomcat-app1]#bash build.sh
[root@ubuntu1804 tomcat-app1]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
tomcat-web          app1                3e9eacc5ef86        4 seconds ago
824MB
tomcat-base         v8.5.50             8d5395cb72c4        35 minutes ago
824MB
centos7-jdk         8u212               e0fe770a7ccd        57 minutes ago
809MB
centos7-base        v1                  34ab3afcd3b3        2 hours ago
403MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.4.4.6 从镜像启动测试容器

```
[root@ubuntu1804 tomcat-app1]#docker run -d  -p 8080:8080  tomcat-web:app1
82e6690e36c3a6faf2dae62bd706a89cbba490d567c841c37501f0fba670ea25
```

2.4.4.7 访问测试

```
[root@ubuntu1804 ~]#curl 127.0.0.1:8080/app/
Tomcat Page in app1
[root@ubuntu1804 ~]#docker exec -it 82e6690e36c3  bash
[root@82e6690e36c3 /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.2  15136  2248 ?        Ss   22:14   0:00 /bin/bash
/apps/tomcat/bin/run_tomcat.sh
www          25  0.8  9.7 2241656 95924 ?       Sl   22:14   0:04
/usr/local/jdk/bin/java -Djava.util.logging.config.file=/apps/tomcat
root         26  0.0  0.4  85428  4472 ?        S    22:14   0:00 su - www -c
tail -f /etc/hosts
www          27  0.0  0.0   4416   720 ?        Ss   22:14   0:00 tail -f
/etc/hosts
root         82 25.0  0.3  15800  3820 pts/0    Ss   22:22   0:00 bash
root        101  0.0  0.3  55196  3836 pts/0    R+   22:22   0:00 ps aux
[root@82e6690e36c3 /]# vim /data/tomcat/webapps/app/index.jsp
[root@82e6690e36c3 /]# cat /data/tomcat/webapps/app/index.jsp
```

```
Tomcat Page in app1 v2
[root@82e6690e36c3 /]# /apps/tomcat/bin/catalina.sh stop
Using CATALINA_BASE:   /apps/tomcat
Using CATALINA_HOME:   /apps/tomcat
Using CATALINA_TMPDIR: /apps/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:       /apps/tomcat/bin/bootstrap.jar:/apps/tomcat/bin/tomcat-
juli.jar
[root@82e6690e36c3 /]# /apps/tomcat/bin/catalina.sh start
Using CATALINA_BASE:   /apps/tomcat
Using CATALINA_HOME:   /apps/tomcat
Using CATALINA_TMPDIR: /apps/tomcat/temp
Using JRE_HOME:        /usr/local/jdk/jre
Using CLASSPATH:       /apps/tomcat/bin/bootstrap.jar:/apps/tomcat/bin/tomcat-
juli.jar
Tomcat started.

[root@ubuntu1804 tomcat-app1]#curl 127.0.0.1:8080/app/
Tomcat Page in app1 v2
```

2.4.4 构建业务镜像2

2.4.4.1 准备自定义页面和其它数据

```
[root@ubuntu1804 tomcat]#pwd
/data/dockerfile/web/tomcat
[root@ubuntu1804 tomcat]#cp -a tomcat-app1/* tomcat-app2/
[root@ubuntu1804 tomcat]#tree tomcat-app2/
tomcat-app2/
├── app
│   └── index.jsp
├── app.tar.gz
├── build.sh
├── Dockerfile
├── run_tomcat.sh
└── server.xml

1 directory, 6 files

[root@ubuntu1804 tomcat]#cd tomcat-app2
[root@ubuntu1804 tomcat-app2]#echo "Tomcat Page in app2" > app/index.html
[root@ubuntu1804 tomcat-app2]#rm -f app.tar.gz
[root@ubuntu1804 tomcat-app2]#tar zcf app.tar.gz app
```

2.4.4.2 准备容器启动脚本run_tomcat.sh

和业务1一样不变

2.4.4.3 准备Dockerfile

和业务1一样不变

2.4.4.4 执行构建脚本制作镜像

```
[root@ubuntu1804 tomcat-app2]#vim build.sh
[root@ubuntu1804 tomcat-app2]#cat build.sh
#!/bin/bash
docker build -t tomcat-web:app2 .
[root@ubuntu1804 tomcat-app2]#bash build.sh
[root@ubuntu1804 tomcat-app2]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
tomcat-web          app2                0e1760fe79a6        37 seconds ago
838MB
tomcat-web          app1                76016219a0ca        27 minutes ago
838MB
tomcat-base         v8.5.50             8d5395cb72c4        2 hours ago
824MB
centos7-jdk         8u212               e0fe770a7ccd        2 hours ago
809MB
centos7-base        v1                  34ab3afcd3b3        3 hours ago
403MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.4.4.5 从镜像启动容器测试

```
[root@ubuntu1804 tomcat-app2]#docker run  -d -p 8082:8080 tomcat-web:app2
3fc9437e42099e92f88e8e09bac0507f2d837ac8a6dba8cb1e4efc934bcf81ff
```

2.4.4.6 访问测试

```
[root@ubuntu1804 tomcat-app2]#curl 127.0.0.1:8082/app/
Tomcat Page in app2
```

2.5 生产案例: 构建haproxy镜像

2.5.1 准备相关文件

```
#准备haproxy源码文件
[root@ubuntu1804 ~]#mkdir -p  /data/dockerfile/web/haproxy/2.1.2-centos7
[root@ubuntu1804 ~]#cd /data/dockerfile/web/haproxy/2.1.2-centos7
[root@ubuntu1804 2.1.2-centos7]#wget
http://www.haproxy.org/download/2.1/src/haproxy-2.1.2.tar.gz

#准备haproxy启动脚本
[root@ubuntu1804 2.1.2-centos7]#vim run_haproxy.sh
[root@ubuntu1804 2.1.2-centos7]#cat run_haproxy.sh
#!/bin/bash
haproxy -f /etc/haproxy/haproxy.cfg
tail -f /etc/hosts
```

2.5.2 准备haproxy配置文件

```
#准备haproxy配置文件
[root@ubuntu1804 2.1.2-centos7]#cat haproxy.cfg
global
chroot /apps/haproxy
#stats socket /var/lib/haproxy/haproxy.sock mode 600 level admin
uid 99
gid 99
daemon
nbproc 1
pidfile /apps/haproxy/run/haproxy.pid
log 127.0.0.1 local3 info
defaults
option http-keep-alive
option forwardfor
mode http
timeout connect 300000ms
timeout client 300000ms
timeout server 300000ms
listen stats
 mode http
 bind 0.0.0.0:9999
 stats enable
 log global
 stats uri   /haproxy-status
 stats auth  haadmin:123456

listen web_port
 bind 0.0.0.0:80
 mode http
 log global
 balance roundrobin
 server web1 10.0.0.101:8080 check inter 3000 fall 2 rise 5
 server web2 10.0.0.102:8080 check inter 3000 fall 2 rise 5
```

2.5.3 准备Dockerfile

```
[root@ubuntu1804 2.1.2-centos7]#pwd
/data/dockerfile/web/haproxy/2.1.2-centos7

[root@ubuntu1804 haproxy]# cat Dockerfile
#Haproxy Base Image
FROM centos7-base:v1
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"

ADD haproxy-2.1.2.tar.gz /usr/local/src/

RUN cd /usr/local/src/haproxy-2.1.2 \
    && make ARCH=x86_64 TARGET=linux-glibc USE_PCRE=1 USE_OPENSSL=1 USE_ZLIB=1
USE_SYSTEMD=1 USE_CPU_AFFINITY=1 PREFIX=/apps/haproxy \
    && make install PREFIX=/apps/haproxy \
    && ln -s /apps/haproxy/sbin/haproxy  /usr/sbin/ \
    && mkdir /apps/haproxy/run \
    && rm -rf /usr/local/src/haproxy*

ADD haproxy.cfg /etc/haproxy/
```

```
ADD run_haproxy.sh /usr/bin

EXPOSE 80 9999
CMD ["run_haproxy.sh"]
```

2.5.4 准备构建脚本构建haproxy镜像

```
[root@ubuntu1804 2.1.2-centos7]#vim build.sh
[root@ubuntu1804 2.1.2-centos7]#cat  build.sh
#!/bin/bash
docker build -t haproxy-centos7:2.1.2 .

[root@ubuntu1804 2.1.2-centos7]#ls
build.sh  Dockerfile  haproxy-2.1.2.tar.gz  haproxy.cfg  run_haproxy.sh

[root@ubuntu1804 2.1.2-centos7]#bash build.sh
[root@ubuntu1804 2.1.2-centos7]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
haproxy-centos7     2.1.2               5eccdb29a058        26 minutes ago
428MB
nginx-ubuntu1804    1.16.1              19efdd23ac87        15 hours ago
378MB
alpine-nginx        1.16.1              978a43bbb61d        16 hours ago
211MB
nginx-alpine        1.16.1              978a43bbb61d        16 hours ago
211MB
alpine-base         3.11                b162eecf4da9        17 hours ago
182MB
tomcat-web          app2                0e1760fe79a6        37 hours ago
838MB
tomcat-web          app1                76016219a0ca        37 hours ago
838MB
tomcat-base         v8.5.50             8d5395cb72c4        38 hours ago
824MB
centos7-jdk         8u212               e0fe770a7ccd        39 hours ago
809MB
centos7-base        v1                  34ab3afcd3b3        40 hours ago
403MB
alpine              3.11                e7d92cdc71fe        12 days ago
5.59MB
alpine              latest              e7d92cdc71fe        12 days ago
5.59MB
ubuntu              18.04               ccc6e87d482b        2 weeks ago
64.2MB
ubuntu              bionic              ccc6e87d482b        2 weeks ago
64.2MB
centos              centos7.7.1908      08d05d1d5859        2 months ago
204MB
```

2.5.5 从镜像启动容器

```
[root@ubuntu1804 2.1.2-centos7]#docker run -d -p 80:80 -p 9999:9999 haproxy-centos7:2.1.2
e0a7c827cb5fdd5a630f7dfe58b1f60822da18929a4dfeccb7490fb78403e3df
```

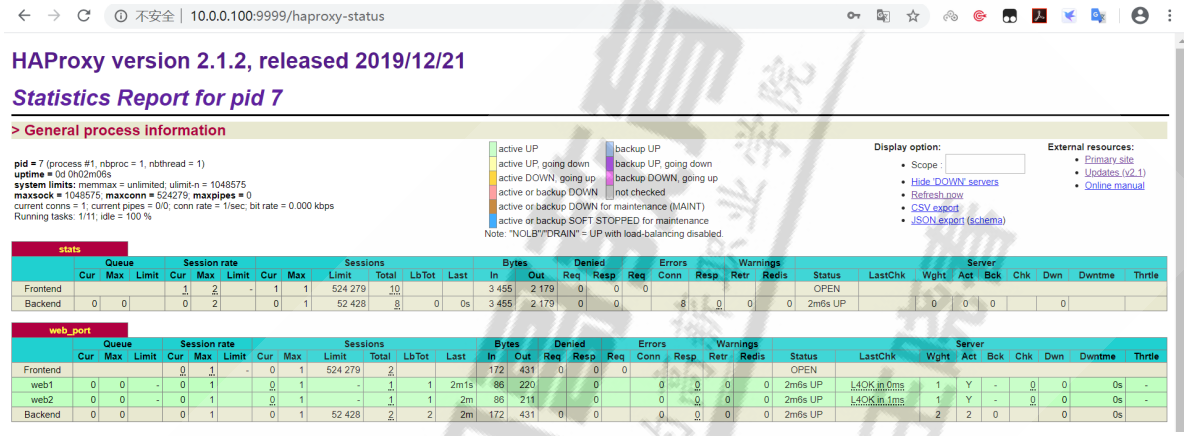
2.5.6 在另外两台主机启动容器

```
#导出本地相关镜像
[root@ubuntu1804 ~]#docker save centos7-base:v1 > /data/centos7-base.tar.gz
[root@ubuntu1804 ~]#docker save centos7-jdk:8u212 > /data/centos7-jdk.tar.gz
[root@ubuntu1804 ~]#docker save tomcat-base:v8.5.50 > /data/tomcat-base.tar.gz
[root@ubuntu1804 ~]#docker save tomcat-web:app1 > /data/tomcat-web-app1.tar.gz
[root@ubuntu1804 ~]#docker save tomcat-web:app2 > /data/tomcat-web-app2.tar.gz
[root@ubuntu1804 ~]#ls /data
centos7-base.tar.gz  centos7-jdk.tar.gz  dockerfile  tomcat-base.tar.gz  tomcat-web-app1.tar.gz  tomcat-web-app2.tar.gz

#将镜像复制到另外两台主机
[root@ubuntu1804 ~]#scp /data/*.gz 10.0.0.101:/data/
[root@ubuntu1804 ~]#scp /data/*.gz 10.0.0.102:/data/

#在另外两台主机上执行下面操作导入镜像
[root@ubuntu1804 ~]#ls /data
centos7-base.tar.gz  lost+found          tomcat-web-app1.tar.gz
centos7-jdk.tar.gz   tomcat-base.tar.gz  tomcat-web-app2.tar.gz
[root@ubuntu1804 ~]#for i in /data/*.gz;do docker load -i $i;done

#在另外两台主机上创建相关容器
[root@ubuntu1804 ~]#docker run -d -p 8080:8080 tomcat-web:app1
781681e73333396b23f404e70d0c781ab464a8e9b578f41c153583d23bd76a46
[root@ubuntu1804 ~]#docker run -d -p 8080:8080 tomcat-web:app2
81fa01a688cb72cf397a5da46acc89a51f2a2f8de3a0072565d701625c43540a
```

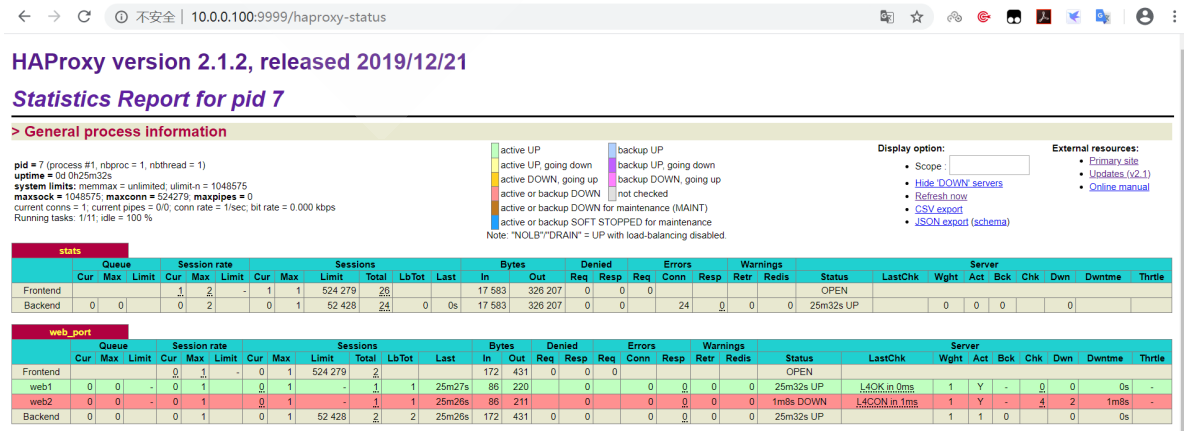
2.5.7 web访问验证

```
[root@ubuntu1804 2.1.2-centos7]#curl http://10.0.0.100/app/
Tomcat Page in app1
[root@ubuntu1804 2.1.2-centos7]#curl http://10.0.0.100/app/
Tomcat Page in app2
[root@ubuntu1804 2.1.2-centos7]#docker exec -it e0a7c827cb5 bash
[root@e0a7c827cb5f /]# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:9999            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
[root@e0a7c827cb5f /]# vim /etc/haproxy/haproxy.cfg
[root@e0a7c827cb5f /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.2  11700  2428 ?        Ss   11:01   0:00 /bin/bash /usr/bin/run_haproxy.sh
nobody        7  0.0  7.1 181076 70324 ?        Ss   11:01   0:00 haproxy -f /etc/haproxy/haproxy.cfg
root          8  0.0  0.0   4416   772 ?        S    11:01   0:00 tail -f /etc/hosts
root          9  0.1  0.3  12488  3696 pts/0    Ss   11:02   0:00 bash
```

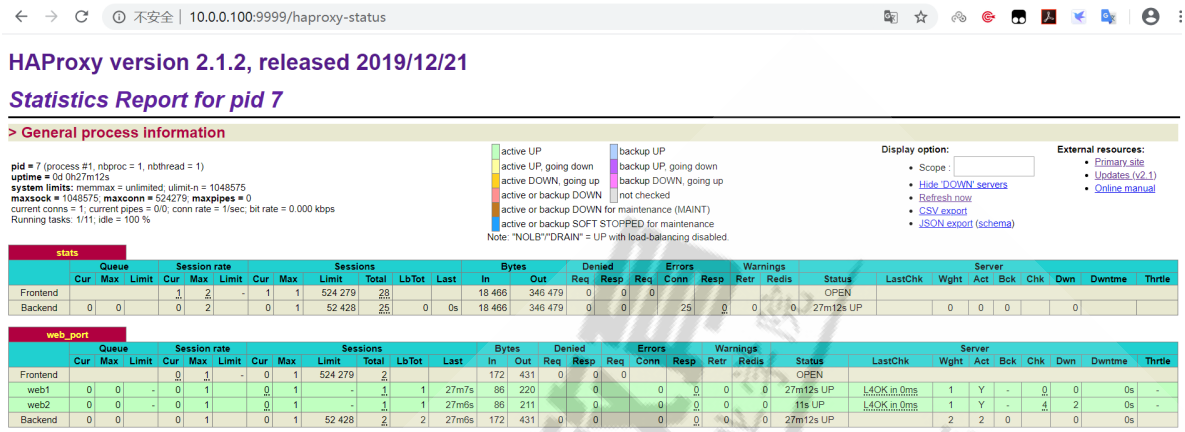
```
root          54  0.0  0.3  51764  3448 pts/0    R+   11:06   0:00 ps aux
```




```
#在第二台主机上停止容器
[root@ubuntu1804 ~]#docker stop 81fa01a688cb
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS                      PORTS               NAMES
81fa01a688cb        tomcat-web:app2     "/apps/tomcat/bin/ru…"   28 minutes ago
     Exited (137) 39 seconds ago
#观察状态页，发现后端服务器down
```



```
#在第二台主机上恢复启动容器
[root@ubuntu1804 ~]#docker start 81fa01a688cb
81fa01a688cb
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                                  NAMES
81fa01a688cb        tomcat-web:app2     "/apps/tomcat/bin/ru…"   30 minutes ago
     Up 14 seconds       8009/tcp, 0.0.0.0:8080->8080/tcp   optimistic_shirley
#再次观察状态页，发现后端服务器上线
```



2.6 基于 alpine 基础镜像制作nginx镜像

2.6.1 制作alpine的自定义系统镜像

```
#下载alpine镜像,打新标签
[root@ubuntu1804 ~]#docker pull alpine
[root@ubuntu1804 ~]#docker tag  alpine alpine:3.11
[root@ubuntu1804 ~]#docker images
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine              3.11                e7d92cdc71fe        11 days ago
5.59MB
alpine              latest              e7d92cdc71fe        11 days ago
5.59MB

#准备相关文件
[root@ubuntu1804 ~]#cd /data/dockerfile/system/alpine
[root@ubuntu1804 alpine]#cat repositories
http://mirrors.aliyun.com/alpine/v3.11/main
http://mirrors.aliyun.com/alpine/v3.11/community

#准备Dockerfile文件
[root@ubuntu1804 alpine]#cat Dockerfile
FROM alpine:3.11
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
COPY repositories /etc/apk/repositories
RUN  apk update && apk --no-cache add  iotop  gcc libgcc libc-dev libcurl libc-
utils pcre-dev zlib-dev  libnfs make  pcre pcre2 zip unzip net-tools pstree wget
libevent libevent-dev iproute2

#准备构建脚本
[root@ubuntu1804 alpine]#cat build.sh
```

```
#!/bin/bash
docker build -t alpine-base:3.11 .
[root@ubuntu1804 alpine]#bash build.sh
[root@ubuntu1804 alpine]#docker images  alp*
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
alpine-base         3.11                b162eecf4da9        5 minutes ago
182MB
alpine              3.11                e7d92cdc71fe        11 days ago
5.59MB
alpine              latest              e7d92cdc71fe        11 days ago
5.59MB
```

2.6.2 制作基于alpine自定义镜像的nginx镜像

```
#准备相关文件
[root@ubuntu1804 ~]#mkdir /data/dockerfile/web/nginx/1.16.1-alpine/
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16.1-alpine/
[root@ubuntu1804 1.16.1-alpine]#wget http://nginx.org/download/nginx-
1.16.1.tar.gz
[root@ubuntu1804 1.16.1-alpine]#echo Test Page based nginx-alpine > index.html
[root@ubuntu1804 1.16.1-alpine]#cp ../1.16.1-centos7/nginx.conf  .
[root@ubuntu1804 1.16.1-alpine]#cat nginx.conf
user  nginx;
worker_processes  1;
daemon off;
...
location / {
            root   /data/nginx/html;
...

#编写Dockerfile文件
[root@ubuntu1804 1.16.1-alpine]#vim Dockerfile
[root@ubuntu1804 1.16.1-alpine]#cat Dockerfile
FROM alpine-base:3.11
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
ADD nginx-1.16.1.tar.gz /usr/local/src

RUN cd /usr/local/src/nginx-1.16.1 && ./configure  --prefix=/apps/nginx && make
&& make install && ln -s /apps/nginx/sbin/nginx  /usr/bin/
RUN addgroup  -g 2019 -S nginx && adduser  -s /sbin/nologin -S -D  -u 2019 -G
nginx nginx
COPY nginx.conf /apps/nginx/conf/nginx.conf
ADD index.html /data/nginx/html/index.html
RUN chown  -R  nginx.nginx  /data/nginx/ /apps/nginx/
EXPOSE 80 443
CMD ["nginx"]

#构建镜像
[root@ubuntu1804 1.16.1-alpine]#vim build.sh
[root@ubuntu1804 1.16.1-alpine]#cat build.sh
#!/bin/bash
#*****************************************************************
docker build -t nginx-alpine:1.16.1 .

[root@ubuntu1804 1.16.1-alpine]#ls
build.sh  Dockerfile  index.html  nginx-1.16.1.tar.gz  nginx.conf
```

```
[root@ubuntu1804 1.16.1-alpine]#docker images “*alpine*”
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx-alpine        1.16.1              344ff9acf58b        13 seconds ago
211MB
alpine-base         3.11                b162eecf4da9        About an hour ago
182MB
alpine              3.11                e7d92cdc71fe        11 days ago
5.59MB
alpine              latest              e7d92cdc71fe        11 days ago
5.59MB

#生成容器测试镜像
[root@ubuntu1804 1.16.1-alpine]#docker run -d -p  80:80 nginx-alpine:1.16.1
1cb16e9fe6cd8e583a61c2718a92ce3031313bbf3656c2f85ac84d34ccfe7e0d
[root@ubuntu1804 1.16.1-alpine]#curl 127.0.0.1
Test Page based nginx-alpine
[root@ubuntu1804 1.16.1-alpine]#docker exec -it 1cb16e9fe6cd sh
/ # ps aux
PID   USER     TIME  COMMAND
    1 root      0:00 nginx: master process nginx
    6 nginx     0:00 nginx: worker process
    7 root      0:00 sh
   12 root      0:00 ps aux
/ # ls /data/nginx/html/ -l
total 4
-rw-r--r--    1 nginx    nginx           29 Jan 29 11:08 index.html
/ # exit
[root@ubuntu1804 1.16.1-alpine]#
```

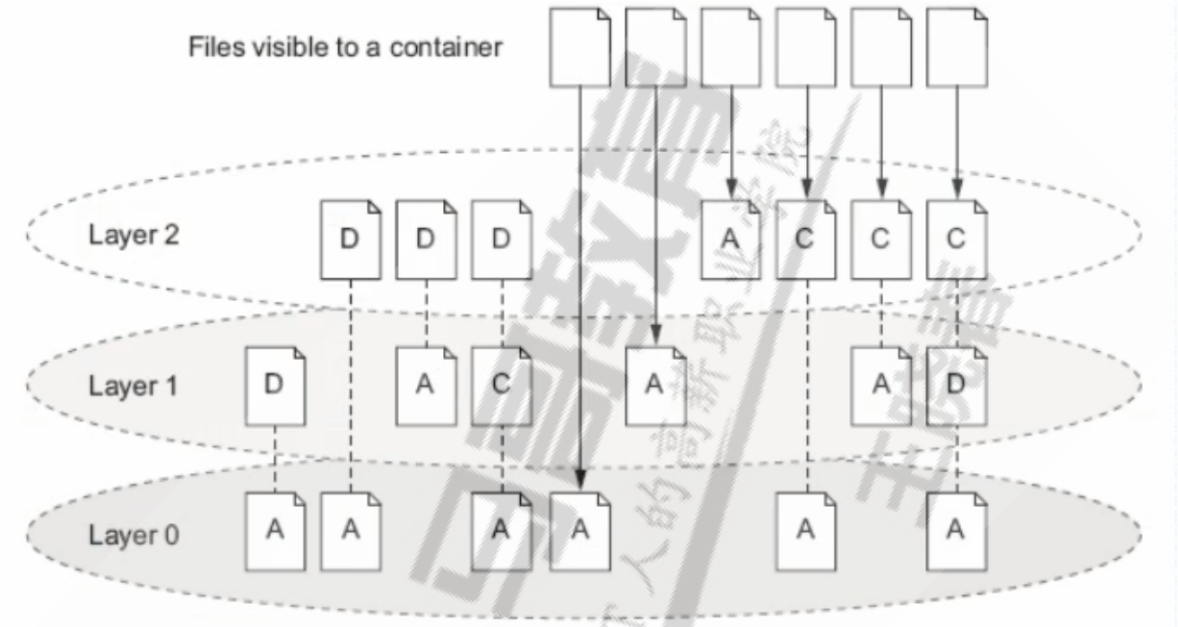
2.7 基于 Ubuntu 基础镜像制作 nginx 镜像

```
#下载ubuntu1804镜像
[root@ubuntu1804 ~]#docker pull ubuntu:18.04
[root@ubuntu1804 ~]#docker images ubuntu*
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
ubuntu              18.04               ccc6e87d482b        13 days ago
64.2MB

#准备相关文件
[root@ubuntu1804 ~]#mkdir /data/dockerfile/web/nginx/1.16.1-ubuntu1804
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16.1-ubuntu1804
[root@ubuntu1804 1.16.1-ubuntu1804]#vim sources.list
[root@ubuntu1804 1.16.1-ubuntu1804]#cat sources.list
deb http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic main restricted universe
multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-security main restricted
universe multiverse
```

```
deb http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-updates main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-proposed main restricted
universe multiverse

deb http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted universe
multiverse
deb-src http://mirrors.aliyun.com/ubuntu/ bionic-backports main restricted
universe multiverse

[root@ubuntu1804 1.16.1-ubuntu1804]#wget http://nginx.org/download/nginx-
1.16.1.tar.gz
[root@ubuntu1804 1.16.1-ubuntu1804]#cp ../1.16.1-alpine/nginx.conf .
[root@ubuntu1804 1.16.1-ubuntu1804]#echo Test Page based nginx-ubuntu1804 >
index.html

#编写Dockerfile文件
[root@ubuntu1804 1.16.1-ubuntu1804]#vim Dockerfile
[root@ubuntu1804 1.16.1-ubuntu1804]#cat Dockerfile
FROM ubuntu:18.04
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
COPY sources.list /etc/apt/sources.list
RUN apt update && apt  install -y  nfs-kernel-server nfs-common  gcc openssh-
server lrzsz tree  openssl libssl-dev libpcre3 libpcre3-dev zlib1g-dev   unzip
zip make
ADD nginx-1.16.1.tar.gz /usr/local/src
RUN cd /usr/local/src/nginx-1.16.1 && ./configure --prefix=/apps/nginx && make
&& make install && ln -s /apps/nginx/sbin/nginx /usr/bin && rm  -rf
/usr/local/src/nginx-1.16.1*
ADD nginx.conf /apps/nginx/conf/nginx.conf
ADD index.html /data/nginx/html/index.html
RUN groupadd  -g 2019 nginx && useradd  -g nginx -s /usr/sbin/nologin -u 2019
nginx && chown -R nginx.nginx /apps/nginx /data/nginx
EXPOSE 80 443
CMD ["nginx"]

#构建镜像
[root@ubuntu1804 1.16.1-ubuntu1804]#vim build.sh
[root@ubuntu1804 1.16.1-ubuntu1804]#cat build.sh
#!/bin/bash
docker build -t nginx-ubuntu1804:1.16.1 .
[root@ubuntu1804 1.16.1-ubuntu1804]#ls
build.sh  Dockerfile  index.html  nginx-1.16.1.tar.gz  nginx.conf  sources.list

[root@ubuntu1804 1.16.1-ubuntu1804]#docker images "nginx*"
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx-ubuntu1804    1.16.1              19efdd23ac87        4 minutes ago
378MB
nginx-alpine        1.16.1              978a43bbb61d        40 minutes ago
211MB
nginx-centos7       1.6.1-v2            1918d29d5f45        17 minutes ago
328MB
```

```
nginx-centos7       1.6.1               8c16774437a5        13 hours ago
412MB

#启动容器测试镜像
[root@ubuntu1804 1.16.1-ubuntu1804]#docker run -d  -p 80:80 nginx-
ubuntu1804:1.16.1
58f8e9a8fd6eebb19bd2b7c27bd8d52a3a4d42637a942e1e9179ec1b2bcc559d
[root@ubuntu1804 1.16.1-ubuntu1804]#curl 127.0.0.1
Test Page based nginx-ubuntu1804
```

3 Docker 数据管理



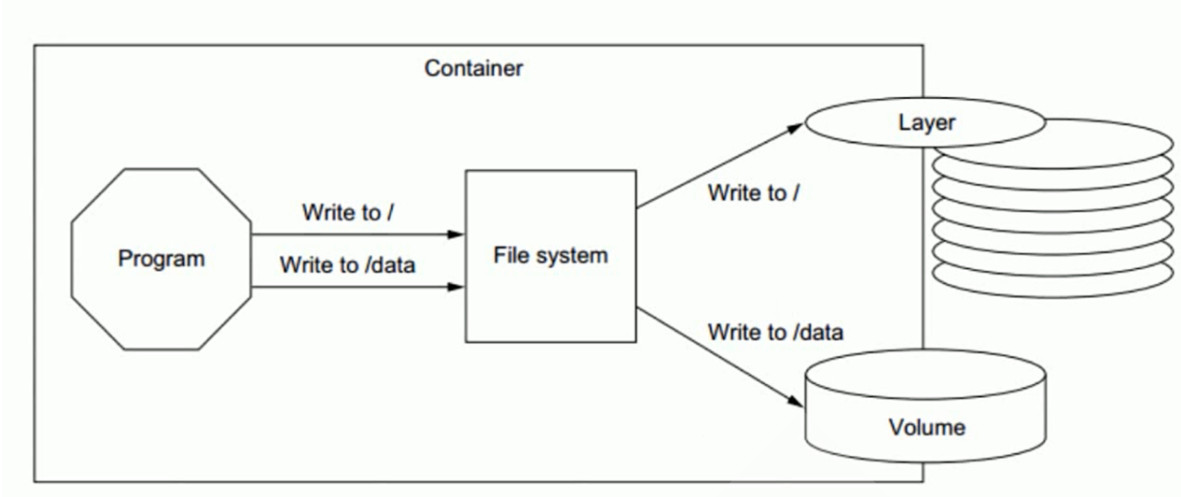
Docker镜像由多个只读层叠加而成，启动容器时，Docker会加载只读镜像层并在镜像栈顶部添加一个读写层

如果运行中的容器修改了现有的一个已经存在的文件，那该文件将会从读写层下面的只读层复制到读写层，该文件的只读版本仍然存在，只是已经被读写层中该文件的副本所隐藏，此即“写时复制(COW copy on write)"机制

如果将正在运行中的容器修改生成了新的数据，那么新产生的数据将会被复制到读写层，进行持久化保存，这个读写层也就是容器的工作目录，也为写时复制(COW) 机制。

COW机制节约空间,但会导致性低下,虽然关闭重启容器,数据不受影响,但会随着容器的删除,其对应的可写层也会随之而删除,即数据也会丢失.如果容器需要持久保存数据,并不影响性能可以用数据卷技术实现

如下图是将对根的数据写入到了容器的可写层，但是把/data 中的数据写入到了一个另外的volume 中用于数据持久化



3.1 容器的数据管理介绍

Docker镜像是分层设计的，镜像层是只读的，通过镜像启动的容器添加了一层可读写的文件系统，用户写入的数据都保存在这一层中。

3.1.1 Docker容器的分层

容器的数据分层目录

- LowerDir：image 镜像层,即镜像本身，只读
- UpperDir：容器的上层,可读写 ,容器变化的数据存放在此处
- MergedDir：容器的文件系统，使用Union FS（联合文件系统）将lowerdir 和 upperdir 合并完成后给容器使用,最终呈现给用户的统一视图
- WorkDir：容器在宿主机的工作目录,挂载后内容会被清空，且在使用过程中其内容用户不可见

范例：查看指定容器数据分层

```
[root@ubuntu1804 ~]#docker inspect  12959f2c152f
        "GraphDriver": {
            "Data": {
                "LowerDir":
"/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9
ed58751761-
init/diff:/var/lib/docker/overlay2/4a259e4cc9da105d17075e316c3e1f7c29abd6f7a37c5
c7c29251a7e5f0e7eb3/diff",
                "MergedDir":
"/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9
ed58751761/merged",
                "UpperDir":
"/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9
ed58751761/diff",
                "WorkDir":
"/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9
ed58751761/work"
            },
            "Name": "overlay2"

[root@ubuntu1804 ~]#ll -i
/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9e
d58751761
total 28
920832 drwx------  5 root root 4096 Jan 31 19:02 ./
```

```
917524 drwx------ 53 root root 4096 Jan 31 19:02 ../
920843 drwxr-xr-x  3 root root 4096 Jan 31 19:02 diff/
920846 -rw-r--r--  1 root root   26 Jan 31 19:02 link
920851 -rw-r--r--  1 root root   57 Jan 31 19:02 lower
920818 drwxr-xr-x  1 root root 4096 Jan 31 19:02 merged/
920847 drwx------  3 root root 4096 Jan 31 19:02 work/


[root@ubuntu1804 ~]#tree -d
/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9e
d58751761/
/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9e
d58751761/
├── diff
│   └── root
├── merged
│   ├── bin
│   ├── dev
│   │   ├── pts
│   │   └── shm
│   ├── etc
│   │   ├── apk
│   │   │   ├── keys
│   │   │   └── protected_paths.d
│   │   ├── conf.d
│   │   ├── crontabs
│   │   ├── init.d
│   │   ├── logrotate.d
│   │   ├── modprobe.d
│   │   ├── modules-load.d
│   │   ├── network
│   │   │   ├── if-down.d
│   │   │   ├── if-post-down.d
│   │   │   ├── if-post-up.d
│   │   │   ├── if-pre-down.d
│   │   │   ├── if-pre-up.d
│   │   │   └── if-up.d
│   │   ├── opt
│   │   ├── periodic
│   │   │   ├── 15min
│   │   │   ├── daily
│   │   │   ├── hourly
│   │   │   ├── monthly
│   │   │   └── weekly
│   │   ├── profile.d
│   │   ├── ssl
│   │   │   ├── certs
│   │   │   ├── misc
│   │   │   └── private
│   │   └── sysctl.d
│   ├── home
│   ├── lib
│   │   ├── apk
│   │   │   └── db
│   │   ├── firmware
│   │   └── mdev
│   ├── media
│   │   ├── cdrom
```

```
|   |   ├── floppy
|   |   └── usb
|   ├── mnt
|   ├── opt
|   ├── proc
|   ├── root
|   ├── run
|   ├── sbin
|   ├── srv
|   ├── sys
|   ├── tmp
|   ├── usr
|   |   ├── bin
|   |   ├── lib
|   |   |   └── engines-1.1
|   |   ├── local
|   |   |   ├── bin
|   |   |   ├── lib
|   |   |   └── share
|   |   ├── sbin
|   |   └── share
|   |       ├── apk
|   |       |   └── keys
|   |       |       ├── aarch64
|   |       |       ├── armhf
|   |       |       ├── ppc64le
|   |       |       ├── s390x
|   |       |       ├── x86
|   |       |       └── x86_64
|   |       ├── man
|   |       ├── misc
|   |       └── udhcpc
|   └── var
|       ├── cache
|       |   ├── apk
|       |   └── misc
|       ├── empty
|       ├── lib
|       |   ├── apk
|       |   ├── misc
|       |   └── udhcpd
|       ├── local
|       ├── lock
|       |   └── subsys
|       ├── log
|       ├── mail
|       ├── opt
|       ├── run -> /run
|       ├── spool
|       |   ├── cron
|       |   └── mail -> /var/mail
|       └── tmp
└── work
    └── work

99 directories
[root@ubuntu1804 ~]#docker run -it  alpine:3.11  sh
/ #  dd if=/dev/zero of=/root/test.img bs=1M count=10
```

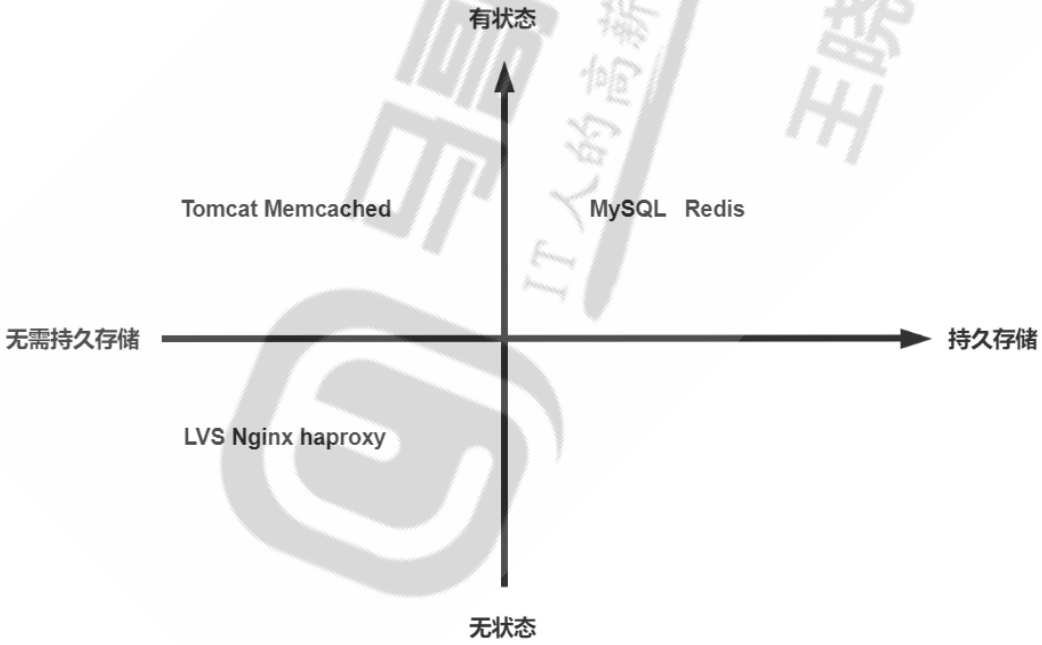
```
10+0 records in
10+0 records out
/ #


#每个镜像层目录中包含了一个文件link，文件内容则是当前层对应的短标识符，镜像层的内容则存放在diff目录
[root@ubuntu1804 ~]#find /var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761 -name test.img -ls
   920903  10240 -rw-r--r--   1 root     root     10485760 Jan 31 19:02 /var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/merged/root/test.img
   920903  10240 -rw-r--r--   1 root     root     10485760 Jan 31 19:02 /var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/diff/root/test.img

[root@ubuntu1804 ~]#mount
overlay on /var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/merged type overlay (rw,relatime,lowerdir=/var/lib/docker/overlay2/l/OIRNCV35BGPGWKRWEMCJD5SYUZ:/var/lib/docker/overlay2/l/VZ7N54CVY2JLDASJAZ6AASRQSJ,upperdir=/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/diff,workdir=/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/work)
nsfs on /run/docker/netns/6ce1d74bc9af type nsfs (rw)

[root@ubuntu1804 ~]#tree /var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/diff/
/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/diff/
└── root
    └── test.img

1 directory, 1 file
[root@ubuntu1804 ~]#ls /var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/merged/
bin  dev  etc  home  lib  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var

[root@ubuntu1804 ~]#docker run -it  alpine:3.11  sh
/ # echo welcome to magedu >> /etc/issue
/ # cat /etc/issue
Welcome to Alpine Linux 3.11
Kernel \r on an \m (\l)

welcome to magedu
/ #

[root@ubuntu1804 ~]#tree /var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/diff/
/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761/diff/
├── etc
```

```
|   └── issue
└── root
    └── test.img

2 directories, 2 files

#删除容器后，所有容器数据目录都随之而删除
[root@ubuntu1804 ~]#docker rm -f 12959f2c152f
[root@ubuntu1804 ~]#ls
/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761
ls: cannot access
'/var/lib/docker/overlay2/848d77064091ba3ddd25a10ea6e0065af15ee701fed06f82804cf9ed58751761': No such file or directory
```

3.1.2 哪些数据需要持久化

有状态的协议

> 有状态协议就是就通信双方要记住双方，并且共享一些信息。而无状态协议的通信每次都是独立的，与上一次的通信没什么关系。
> "状态"可以理解为"记忆"，有状态对应有记忆，无状态对应无记忆



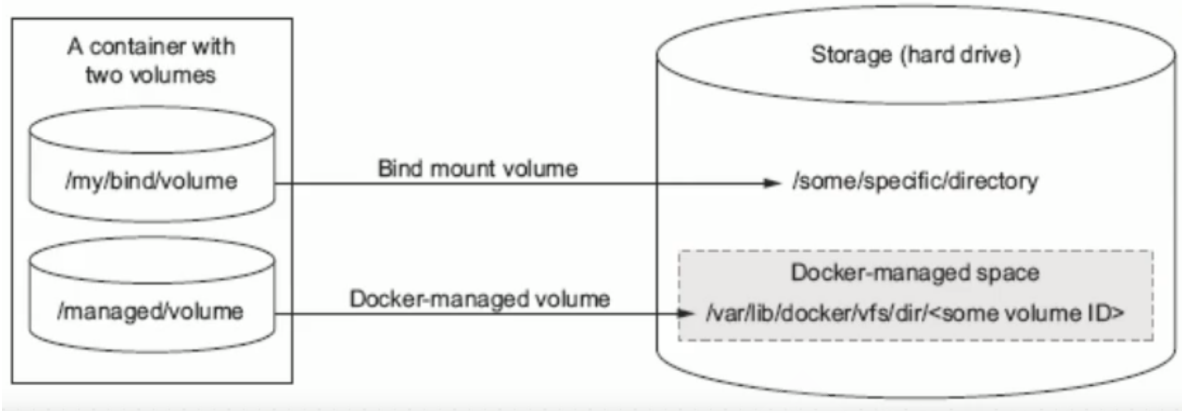
- 左侧是无状态的http请求服务，右侧为有状态
- 下层为不需要存储的服务，上层为需要存储的部分服务

3.1.3 容器数据持久保存方式

如果要将写入到容器的数据永久保存，则需要将容器中的数据保存到宿主机的指定目录

Docker的数据类型分为两种:

- 数据卷(Data Volume)：直接将宿主机目录挂载至容器的指定的目录，推荐使用此种方式，此方式较常用
- 数据卷容器(Data Volume Container)：间接使用宿主机空间，数据卷容器是将宿主机的目录挂载至一个专门的数据卷容器，然后让其他容器通过数据卷容器读写宿主机的数据，此方式不常用



3.2 数据卷(data volume)

3.2.1 数据卷特点和使用

数据卷实际上就是宿主机上的目录或者是文件，可以被直接mount到容器当中使用

实际生成环境中，需要针对不同类型的服务、不同类型的数据存储要求做相应的规划，最终保证服务的可扩展性、稳定性以及数据的安全性

3.2.1.1 数据卷使用场景

- 数据库
- 日志输出
- 静态web页面
- 应用配置文件
- 多容器间目录或文件共享

3.2.1.2 数据卷的特点

- 数据卷是目录或者文件，并且可以在多个容器之间共同使用,实现容器之间共享和重用
- 对数据卷更改数据在所有容器里面会立即更新。
- 数据卷的数据可以持久保存，即使删除使用使用该容器卷的容器也不影响。
- 在容器里面的写入数据不会影响到镜像本身,即数据卷的变化不会影响镜像的更新
- 依赖于宿主机目录，宿主机出问题，上面容器会受影响，当宿主机较多时，不方便统一管理
- 匿名和命名数据卷在容器启动时初始化，如果容器使用的镜像在挂载点包含了数据，会拷贝到新初始化的数据卷中

3.2.1.3 数据卷使用方法

启动容器时，可以指定使用数据卷实现容器数据的持久化,数据卷有三种

- 指定宿主机目录或文件: 指定宿主机的具体路径和容器路径的挂载关系
- 匿名卷: 不指定数据名称,只指定容器内目录路径充当挂载点,docker自动指定宿主机的路径进行挂载
- 命名卷: 指定数据卷的名称和容器路径的挂载关系

docker run 命令的以下格式可以实现数据卷

```
-v, --volume=[host-src:]container-dest[:<options>]

<options>
ro 从容器内对此数据卷是只读，不写此项默认为可读可写
rw 从容器内对此数据卷可读可写,此为默认值
```

方式1

```
#指定宿主机目录或文件格式:
-v    <宿主机绝对路径的目录或文件>:<容器目录或文件>[:ro]  #将宿主机目录挂载容器目录，两个目录都可自动创建
```

方式2

```
#匿名卷,只指定容器内路径,没有指定宿主机路径信息,宿主机自动生成/var/lib/docker/volumes/<卷ID>/_data目录,并挂载至容器指定路径
-v <容器内路径>

#示例:
docker run --name nginx -v /etc/nginx nginx
```

方式3

```
#命名卷将固定的存放在/var/lib/docker/volumes/<卷名>/_data
-v <卷名>:<容器目录路径>
#可以通过以下命令事先创建,如可没有事先创建卷名,docker run时也会自动创建卷
docker volume  create <卷名>

#示例:
docker run -d  -p 80:80 --name nginx01 -v vol1:/usr/share/nginx/html  nginx
```

docker rm 的 -v 选项可以删除容器时，同时删除相关联的匿名卷

```
-v, --volumes   Remove the volumes associated with the container
```

管理卷命令

```
docker volume COMMAND

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes
```

关于匿名数据卷和命名数据卷

命名卷就是有名字的卷，使用 docker volume create <卷名> 形式创建并命名的卷；而匿名卷就是没名字的卷，一般是 docker run -v /data 这种不指定卷名的时候所产生，或者 Dockerfile 里面的定义直接使用的。

有名字的卷，在用过一次后，以后挂载容器的时候还可以使用，因为有名字可以指定。所以一般需要保存的数据使用命名卷保存。
而匿名卷则是随着容器建立而建立，随着容器消亡而淹没于卷列表中（对于 docker run 匿名卷不会被自动删除）。 因此匿名卷只存放无关紧要的临时数据，随着容器消亡，这些数据将失去存在的意义。

Dockerfile中指定VOLUME为匿名数据卷,其目的只是为了将某个路径确定为卷。

按照最佳实践的要求，不应该在容器存储层内进行数据写入操作，所有写入应该使用卷。如果定制镜像的时候，就可以确定某些目录会发生频繁大量的读写操作，那么为了避免在运行时由于用户疏忽而忘记指定卷，导致容器发生存储层写入的问题，就可以在 Dockerfile 中使用 VOLUME 来指定某些目录为匿名卷。这样即使用户忘记了指定卷，也不会产生不良的后果。
这个设置可以在运行时覆盖。通过 docker run 的 -v 参数或者 docker-compose.yml 的 volumes 指定。使用命名卷的好处是可以复用，其它容器可以通过这个命名数据卷的名字来指定挂载，共享其内容（不过要注意并发访问的竞争问题）。

比如，Dockerfile 中说 VOLUME /data，那么如果直接 docker run，其 /data 就会被挂载为匿名卷，向 /data 写入的操作不会写入到容器存储层，而是写入到了匿名卷中。但是如果运行时 docker run -v mydata:/data，这就覆盖了 /data 的挂载设置，要求将 /data 挂载到名为 mydata 的命名卷中。所以说 Dockerfile 中的 VOLUME 实际上是一层保险，确保镜像运行可以更好的遵循最佳实践，不向容器存储层内进行写入操作。

数据卷默认可能会保存于 /var/lib/docker/volumes，不过一般不需要、也不应该访问这个位置。

查看数据卷的挂载关系

```
docker inspect --format="{{.Mounts}}"  <容器ID>
```

范例: 删除所有数据卷

```
[root@ubuntu1804 ~]#docker volume rm `docker volume ls -q`
```

3.2.2 实战案例: 目录数据卷

3.2.2.1 在宿主机创建容器所使用的目录

```
[root@ubuntu1804 ~]#mkdir /data/testdir
[root@ubuntu1804 ~]#echo  Test page  on host > /data/testdir/index.html
```

3.2.2.2 查看容器相关目录路径

```
[root@ubuntu1804 ~]#docker images "*nginx*"
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
nginx-ubuntu1804    1.16.1              19efdd23ac87        2 days ago
378MB
nginx-alpine        1.16.1              978a43bbb61d        2 days ago
211MB
```

```
[root@ubuntu1804 ~]#docker run -it  --rm  nginx-alpine:1.16.1 sh
/ # cat /apps/nginx/conf/nginx.conf
...
    server {
...
        location / {
            root   /data/nginx/html;  #nginx存放网页文件的路径
            index  index.html index.htm;
        }
...
/ # cat  /data/nginx/html/index.html
Test Page based nginx-alpine
/ # exit
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
```

3.2.2.3 引用宿主机的数据卷启动容器

引用数据卷目录，开启多个容器

```
[root@ubuntu1804 ~]#docker run  -d -v /data/testdir:/data/nginx/html/   -p
8001:80 nginx-alpine:1.16.1
56a5460f584bd2de56040c4a1dff86ad8a9723cfd6bf21ed8a538b9629b0874c
[root@ubuntu1804 ~]#docker run  -d -v /data/testdir:/data/nginx/html/   -p
8002:80 nginx-alpine:1.16.1
e7b5bff6ce56fa51ed6411175c9c9f1fb9bf8e7b1b9471080380b01692f89e58
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED
  STATUS              PORTS                       NAMES
e7b5bff6ce56        nginx-alpine:1.16.1    "nginx"             6 seconds ago
  Up 5 seconds        443/tcp, 0.0.0.0:8002->80/tcp   hungry_robinson
56a5460f584b        nginx-alpine:1.16.1    "nginx"             33 seconds ago
Up 31 seconds       443/tcp, 0.0.0.0:8001->80/tcp   stupefied_dubinsky
[root@ubuntu1804 ~]#curl 127.0.0.1:8001
Test page on host
[root@ubuntu1804 ~]#curl 127.0.0.1:8002
Test page on host
```

3.2.2.3 进入到容器内测试写入数据

进入其中一个容器写入数据，可以其它容器的数据也变化

```
[root@ubuntu1804 ~]#docker exec -it e7b5bff6ce56 sh
/ # df
Filesystem           1K-blocks      Used Available Use% Mounted on
overlay              47799020   5294492  40046724  12% /
tmpfs                   65536         0     65536   0% /dev
tmpfs                  492552         0    492552   0% /sys/fs/cgroup
shm                     65536         0     65536   0% /dev/shm
/dev/sda2            47799020   5294492  40046724  12% /etc/resolv.conf
/dev/sda2            47799020   5294492  40046724  12% /etc/hostname
/dev/sda2            47799020   5294492  40046724  12% /etc/hosts
/dev/sda3            19091540   3958732  14139940  22% /data/nginx/html
tmpfs                  492552         0    492552   0% /proc/asound
```

```
tmpfs                   492552         0    492552   0% /proc/acpi
tmpfs                    65536         0     65536   0% /proc/kcore
tmpfs                    65536         0     65536   0% /proc/keys
tmpfs                    65536         0     65536   0% /proc/timer_list
tmpfs                    65536         0     65536   0% /proc/sched_debug
tmpfs                   492552         0    492552   0% /proc/scsi
tmpfs                   492552         0    492552   0% /sys/firmware
/ # cat /data/nginx/html/index.html
Test page on host
/ # echo Test page v2 on host > /data/nginx/html/index.html

#进入另一个容器看到数据变化
[root@ubuntu1804 ~]#docker exec -it 56a5460f584b  sh
/ # cat /data/nginx/html/index.html
Test page v2 on host

#访问应用
[root@ubuntu1804 ~]#curl 127.0.0.1:8001
Test page v2 on host
[root@ubuntu1804 ~]#curl 127.0.0.1:8002
Test page v2 on host
```

3.2.2.4 在宿主机修改数据

```
[root@ubuntu1804 ~]#echo Test page v3 on host > /data/testdir/index.html
[root@ubuntu1804 ~]#cat /data/testdir/index.html
Test page v3 on host
[root@ubuntu1804 ~]#curl 127.0.0.1:8001
Test page v3 on host
[root@ubuntu1804 ~]#curl 127.0.0.1:8002
Test page v3 on host

[root@ubuntu1804 ~]#docker exec -it e7b5bff6ce56 sh
/ # cat /data/nginx/html/index.html
Test page v3 on host
[root@ubuntu1804 ~]#docker exec -it 56a5460f584b  sh
/ # cat /data/nginx/html/index.html
Test page v3 on host
```

3.2.2.5 只读方法挂载数据卷

默认数据卷为可读可写，加ro选项，可以实现只读挂载，对于不希望容器修改的数据，比如：配置文件，脚本等，可以用此方式挂载

```
[root@ubuntu1804 ~]#docker run  -d -v /data/testdir:/data/nginx/html/:ro   -p
8004:80 nginx-alpine:1.16.1
727d3ecf65c5a79bd9a11033812dc01619c4c45bd25af5155f904016f5f0c45a

[root@ubuntu1804 ~]#docker exec -it 727d3ecf65c5a79bd9 sh
/ # cat /data/nginx/html/index.html
Test page v3 on host
/ # echo Test page v4 on host > /data/nginx/html/index.html
sh: can't create /data/nginx/html/index.html: Read-only file system
/ # cat /data/nginx/html/index.html
```

3.2.2.6 删除容器

删除容器后，宿主机的数据卷还存在，可继续给新的容器使用

```
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                  COMMAND                  CREATED
  STATUS              PORTS                                NAMES
e7b5bff6ce56        nginx-alpine:1.16.1    "nginx"                  9 minutes ago
  Up 9 minutes        443/tcp, 0.0.0.0:8002->80/tcp   hungry_robinson
56a5460f584b        nginx-alpine:1.16.1    "nginx"                  9 minutes ago
  Up 9 minutes        443/tcp, 0.0.0.0:8001->80/tcp   stupefied_dubinsky
[root@ubuntu1804 ~]#docker rm -f `docker ps -aq`
e7b5bff6ce56
56a5460f584b
[root@ubuntu1804 ~]#cat /data/testdir/index.html
Test page v3 on host

#新建的容器还可以继续使用原有的数据卷
[root@ubuntu1804 ~]#docker run  -d -v /data/testdir:/data/nginx/html/   -p
8003:80 nginx-alpine:1.16.1
ecd016506f6af3f4af61dbd869f8fce5f634ecdc0e3272f9e0402c981acd80a4
[root@ubuntu1804 ~]#curl 127.0.0.1:8003
Test page v3 on host
```

3.2.3 安战案例: MySQL使用的数据卷

```
[root@ubuntu1804 ~]#docker pull mysql:5.7.30
5.7.29: Pulling from library/mysql
Status: Downloaded newer image for mysql:5.7.30
docker.io/library/mysql:5.7.29
[root@ubuntu1804 ~]#docker images "mysql*"
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
mysql               5.7.29              b598110d0fff        2 weeks ago
435MB

[root@ubuntu1804 ~]#docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=123456
mysql:5.7.30
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                               NAMES
c21ca6f8a7fe        mysql:5.7.30        "docker-entrypoint.s…"   3 minutes ago
    Up 3 minutes        0.0.0.0:3306->3306/tcp, 33060/tcp   nifty_banach
```

```
[root@ubuntu1804 ~]#docker exec -it c21ca6f8a7fe bash
root@c21ca6f8a7fe:/# cat /etc/issue
Debian GNU/Linux 9 \n \l


root@c21ca6f8a7fe:/# cat /etc/mysql/my.cnf
......
!includedir /etc/mysql/conf.d/
!includedir /etc/mysql/mysql.conf.d/

root@c21ca6f8a7fe:/# cat /etc/mysql/mysql.conf.d/mysqld.cnf

[mysqld]
pid-file        = /var/run/mysqld/mysqld.pid
socket          = /var/run/mysqld/mysqld.sock
datadir         = /var/lib/mysql    #数据存放路径

root@c21ca6f8a7fe:/# pstree -p
mysqld(1)-+-{mysqld}(130)
          |-{mysqld}(131)
          |-{mysqld}(132)
          |-{mysqld}(133)
          |-{mysqld}(134)
          |-{mysqld}(135)
          |-{mysqld}(136)
          |-{mysqld}(137)
          |-{mysqld}(138)
          |-{mysqld}(139)
          |-{mysqld}(140)
          |-{mysqld}(141)
          |-{mysqld}(143)
          |-{mysqld}(144)
          |-{mysqld}(145)
          |-{mysqld}(146)
          |-{mysqld}(147)
          |-{mysqld}(148)
          |-{mysqld}(149)
          |-{mysqld}(150)
          |-{mysqld}(151)
          |-{mysqld}(152)
          |-{mysqld}(153)
          |-{mysqld}(154)
          |-{mysqld}(155)
          `-{mysqld}(156)

[root@ubuntu1804 ~]#mysql -uroot -p123456 -h127.0.0.1
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql> create database dockerdb;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dockerdb           |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.01 sec)

mysql>

#删除容器后，再创建新的容器，数据库信息丢失
[root@ubuntu1804 ~]#docker rm -f  c21ca6f8a7fe
c21ca6f8a7fe
[root@ubuntu1804 ~]#docker run -d  --name mysql -p 3306:3306 -e
MYSQL_ROOT_PASSWORD=123456  mysql:5.7.30
f52a50c7f80ee39d9a935a762eacb05db72dcfa5f0d02af8b4f23b5538080b67
[root@ubuntu1804 ~]#mysql -uroot -p123456 -h127.0.0.1
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

```
4 rows in set (0.00 sec)


#利用数据卷创建容器
[root@ubuntu1804 ~]#mkdir /data/mysql
[root@ubuntu1804 ~]#docker run -d  --name mysql -p 3306:3306 -v
/data/mysql/:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456  mysql:5.7.30
d64ef3f9a64491061132020306fd3e97e1aa361b0fb9f6f644f2a1e3f334119c
[root@ubuntu1804 ~]#mysql -uroot -p123456 -h127.0.0.1
mysql> create database dockerdb;
Query OK, 1 row affected (0.00 sec)

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| dockerdb           |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> exit

#删除容器，数据存放在挂载数据卷中，不会删除
[root@ubuntu1804 ~]#docker rm -fv mysql
mysql

[root@ubuntu1804 ~]#ls /data/mysql/
auto.cnf    ca.pem           client-key.pem  ib_buffer_pool  ib_logfile0  ibtmp1
performance_schema  public_key.pem   server-key.pem
ca-key.pem  client-cert.pem  dockerdb        ibdata1         ib_logfile1  mysql
  private_key.pem     server-cert.pem  sys

#重新创建新容器，之前数据还在
[root@ubuntu1804 ~]#docker run -d  --name mysql -p 3306:3306 -v
/data/mysql/:/var/lib/mysql -e MYSQL_ROOT_PASSWORD=123456  mysql:5.7.30
8d6cd0822cc61db30c0401992d2dbfa5c33b525f9aead44c165e0e247c37e5df
[root@ubuntu1804 ~]#mysql -uroot -p123456 -h127.0.0.1
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2019, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
```

```
| information_schema |
| dockerdb           |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql> exit

#指定多个数据卷，创建MySQL
[root@ubuntu1804 ~]#docker run --name mysql-test1 -v /data/mysql:/var/lib/mysql
-e MYSQL_ROOT_PASSWORD=123456 -e MYSQL_DATABASE=wordpress -e MYSQL_USER=wpuser -
e MYSQL_PASSWORD=123456 -d -p 3306:3306 mysql:5.7.30

[root@ubuntu1804 ~]#docker run --name mysql-test2 -v
/root/mysql/:/etc/mysql/conf.d -v /data/mysql2:/var/lib/mysql --env-
file=env.list -d -p 3307:3306 mysql:5.7.30

[root@ubuntu1804 ~]#cat mysql/mysql-test.cnf
[mysqld]
server-id=100
log-bin=mysql-bin

[root@ubuntu1804 ~]#cat env.list
MYSQL_ROOT_PASSWORD=123456
MYSQL_DATABASE=wordpress
MYSQL_USER=wpuser
MYSQL_PASSWORD=wppass
```

3.2.4 实战案例: 文件数据卷

文件挂载用于很少更改文件内容的场景，比如: nginx 的配置文件、tomcat的配置文件等。

3.2.4.1 准备相关文件

```
[root@ubuntu1804 ~]#mkdir /data/{bin,testapp,logs}
[root@ubuntu1804 ~]#echo testapp v1  > /data/testapp/index.html
[root@ubuntu1804 ~]#cat /data/testapp/index.html
testapp v1
[root@ubuntu1804 ~]#cp /data/dockerfile/web/tomcat/tomcat-base-8.5.50/apache-
tomcat-8.5.50/bin/catalina.sh  /data/bin/
[root@ubuntu1804 ~]#vim /data/bin/catalina.sh
#加下面tomcat的优化参数行
# -----------------------------------------------------------------------------
JAVA_OPTS="-server -Xms4g -Xmx4g -Xss512k -Xmn1g -
XX:CMSInitiatingOccupancyFraction=65 -XX:+UseFastAccessorMethods -
XX:+AggressiveOpts
 -XX:+UseBiasedLocking -XX:+DisableExplicitGC -XX:MaxTenuringThreshold=10 -
XX:NewSize=2048M -XX:MaxNewSize=2048M -XX:NewRatio=2 -XX:Pe
rmSize=128m -XX:MaxPermSize=512m -XX:CMSFullGCsBeforeCompaction=5 -
XX:+ExplicitGCInvokesConcurrent -XX:+UseConcMarkSweepGC -XX:+UsePar
NewGC -XX:+CMSParallelRemarkEnabled -XX:+UseCMSCompactAtFullCollection -
XX:LargePageSizeInBytes=128m -XX:+UseFastAccessorMethods"
```

```
# OS specific support.  $var _must_ be set to either true or false.

[root@ubuntu1804 ~]#chown 2019:2019 /data/bin/catalina.sh
[root@ubuntu1804 ~]#ll /data/bin/catalina.sh
-rwxr-x--- 1 2019 2019 24324 Jan 31 21:43 /data/bin/catalina.sh*
[root@ubuntu1804 ~]#chown 2019:2019 /data/logs/
```

3.2.4.2 引用文件数据卷启动容器

同时挂载可读可写方式的目录数据卷和只读方式的文件数据卷，实现三个数据卷的挂载，数据，日志和启动脚本

```
[root@ubuntu1804 ~]#docker run -d -v
/data/bin/catalina.sh:/apps/tomcat/bin/catalina.sh:ro -v
/data/testapp:/data/tomcat/webapps/testapp   -v /data/logs:/apps/tomcat/logs -p
8080:8080 tomcat-web:app1
55de76261c5e489de9a24d5533fe630d44aa7cce821627f6503a91c041c8f8d3
```

3.2.4.3 验证容器可以访问

```
[root@ubuntu1804 ~]#curl 127.0.0.1:8080/testapp/
testapp v1

[root@ubuntu1804 ~]#ls -l /data/logs/
total 36
drwxr-xr-x 2 2019 2019 4096 Jan 31 22:44 ./
drwxr-xr-x 7 root root 4096 Jan 31 22:43 ../
-rw-r----- 1 2019 2019 8336 Jan 31 22:44 catalina.2020-01-31.log
-rw-r----- 1 2019 2019 8808 Jan 31 22:44 catalina.out
-rw-r----- 1 2019 2019    0 Jan 31 22:44 host-manager.2020-01-31.log
-rw-r----- 1 2019 2019    0 Jan 31 22:44 localhost.2020-01-31.log
-rw-r----- 1 2019 2019   76 Jan 31 22:44 localhost_access_log.2020-01-31.txt
-rw-r----- 1 2019 2019    0 Jan 31 22:44 manager.2020-01-31.log
```

3.2.4.4 直接修改宿主机的数据

```
#宿主机修改目录数据卷
[root@ubuntu1804 ~]#echo testapp v2 > /data/testapp/index.html
[root@ubuntu1804 ~]#curl 127.0.0.1:8080/testapp/
testapp v2

[root@ubuntu1804 ~]#ll /data/bin/catalina.sh
-rwxr-x--- 1 2019 2019 24324 Jan 31 21:43 /data/bin/catalina.sh*
[root@ubuntu1804 ~]#echo >> /data/bin/catalina.sh
[root@ubuntu1804 ~]#ll /data/bin/catalina.sh
-rwxr-x--- 1 2019 2019 24325 Jan 31 22:21 /data/bin/catalina.sh*
```

3.2.4.5 进入容器修改数据

```
[root@ubuntu1804 ~]#docker exec -it 55de76261c5e bash
[root@55de76261c5e /]# netstat -ntl
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:8005          0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8009            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN

#文件数据卷上的文件为只读
[root@55de76261c5e /]#  echo >> /apps/tomcat/bin/catalina.sh
bash: /apps/tomcat/bin/catalina.sh: Read-only file system

#目录数据卷可读可写
[root@55de76261c5e /]# cat /data/tomcat/webapps/testapp/index.html
testapp v2
[root@55de76261c5e /]# echo testapp v3 > /data/tomcat/webapps/testapp/index.html
[root@55de76261c5e /]# cat /data/tomcat/webapps/testapp/index.html
testapp v3

[root@ubuntu1804 ~]#curl 127.0.0.1:8080/testapp/
testapp v3
```

3.2.4.6 查看容器中挂载和进程信息

```
[root@55de76261c5e /]# mount
......
/dev/sda3 on /apps/apache-tomcat-8.5.50/bin/catalina.sh type ext4
(ro,relatime,data=ordered)
/dev/sda3 on /data/tomcat/webapps/testapp type ext4 (rw,relatime,data=ordered)
......
[root@55de76261c5e /]# df
Filesystem     1K-blocks    Used Available Use% Mounted on
overlay         47799020 5295032  40046184  12% /
tmpfs              65536       0     65536   0% /dev
tmpfs             492552       0    492552   0% /sys/fs/cgroup
shm                65536       0     65536   0% /dev/shm
/dev/sda2       47799020 5295032  40046184  12% /etc/hosts
/dev/sda3       19091540 3974908  14123764  22% /data/tomcat/webapps/testapp
tmpfs             492552       0    492552   0% /proc/asound
tmpfs             492552       0    492552   0% /proc/acpi
tmpfs             492552       0    492552   0% /proc/scsi
tmpfs             492552       0    492552   0% /sys/firmware


[root@55de76261c5e /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.3  15136  3040 ?        Ss   22:09   0:00 /bin/bash
/apps/tomcat/bin/run_tomcat.sh
www          25  0.5 12.7 6253760 125268 ?      Sl   22:09   0:04
/usr/local/jdk/bin/java -Djava.util.logging.config.file=/apps/tomcat
root         26  0.0  0.4  85428  4468 ?        S    22:09   0:00 su - www -c
tail -f /etc/hosts
www          28  0.0  0.0   4416   716 ?        Ss   22:09   0:00 tail -f
/etc/hosts
root         85  0.0  0.4  15800  3980 pts/0    Ss   22:16   0:00 bash
root        109  0.0  0.3  55196  3696 pts/0    R+   22:25   0:00 ps aux
[root@55de76261c5e /]# ps aux|grep java
```

```
www          25  0.5 12.7 6253760 125268 ?      Sl   22:09   0:04
/usr/local/jdk/bin/java -
Djava.util.logging.config.file=/apps/tomcat/conf/logging.properties -
Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager -server -Xms4g
-Xmx4g -Xss512k -Xmn1g -XX:CMSInitiatingOccupancyFraction=65 -
XX:+UseFastAccessorMethods -XX:+AggressiveOpts -XX:+UseBiasedLocking -
XX:+DisableExplicitGC -XX:MaxTenuringThreshold=10 -XX:NewSize=2048M -
XX:MaxNewSize=2048M -XX:NewRatio=2 -XX:PermSize=128m -XX:MaxPermSize=512m -
XX:CMSFullGCsBeforeCompaction=5 -XX:+ExplicitGCInvokesConcurrent -
XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:+CMSParallelRemarkEnabled -
XX:+UseCMSCompactAtFullCollection -XX:LargePageSizeInBytes=128m -
XX:+UseFastAccessorMethods -Djdk.tls.ephemeralDHKeySize=2048 -
Djava.protocol.handler.pkgs=org.apache.catalina.webresources -
Dorg.apache.catalina.security.SecurityListener.UMASK=0027 -
Dignore.endorsed.dirs= -classpath
/apps/tomcat/bin/bootstrap.jar:/apps/tomcat/bin/tomcat-juli.jar -
Dcatalina.base=/apps/tomcat -Dcatalina.home=/apps/tomcat -
Djava.io.tmpdir=/apps/tomcat/temp org.apache.catalina.startup.Bootstrap start
root        111  0.0  0.2  12536  2320 pts/0    S+   22:25   0:00 grep --
color=auto java
```

3.2.5 实战案例: 匿名数据卷

```
[root@ubuntu1804 ~]#docker volume  ls
DRIVER              VOLUME NAME

[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES

#利用匿名数据卷创建容器
[root@ubuntu1804 ~]#docker run -d -p 80:80 --name nginx01 -v
/usr/share/nginx/html  nginx
914c622af7ebae68aab0b0075da39534fdb118cd5d4d7adf3054da94eb2a4952
[root@ubuntu1804 ~]#curl  127.0.0.1
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>
```

```
<p><em>Thank you for using nginx.</em></p>
</body>
</html>

#查看自动生成的匿名数据卷
[root@ubuntu1804 ~]#docker volume  ls
DRIVER              VOLUME NAME
local
2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d75502f8b40fe



#查看匿名数据卷的详细信息
[root@ubuntu1804 ~]#docker volume inspect
2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d75502f8b40fe
[
    {
        "CreatedAt": "2020-07-21T19:17:07+08:00",
        "Driver": "local",
        "Labels": null,
        "Mountpoint":
"/var/lib/docker/volumes/2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d755
02f8b40fe/_data",
        "Name":
"2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d75502f8b40fe",
        "Options": null,
        "Scope": "local"
    }
]

[root@ubuntu1804 ~]#docker inspect --format="{{.Mounts}}"  nginx01
[{volume 2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d75502f8b40fe
/var/lib/docker/volumes/2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d7550
2f8b40fe/_data /usr/share/nginx/html local  true }]


#查看匿名数据卷的文件
[root@ubuntu1804 ~]#ls
/var/lib/docker/volumes/2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d7550
2f8b40fe/_data
50x.html  index.html


#修改宿主机中匿名数据卷的文件
[root@ubuntu1804 ~]#echo Anonymous Volume >
/var/lib/docker/volumes/2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d7550
2f8b40fe/_data/index.html
[root@ubuntu1804 ~]#curl 127.0.0.1
Anonymous Volume

#删除容器不会删除匿名数据卷
[root@ubuntu1804 ~]#docker rm -f  nginx01
nginx01
[root@ubuntu1804 ~]#docker volume  ls
DRIVER              VOLUME NAME
local
2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d75502f8b40fe
```

```
[root@ubuntu1804 ~]#docker run -d -p 80:80 --name nginx01 -v
/usr/share/nginx/html  nginx
40a99ae066cc60aa0ad7d073e46101a15446509acbd7971e677f3cbe6733aa79

[root@ubuntu1804 ~]#cat
/var/lib/docker/volumes/2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d7550
2f8b40fe/_data/index.html
Anonymous Volume

#删除匿名数据卷
[root@ubuntu1804 ~]#docker volume rm
2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d75502f8b40fe
2a18aff09215e7830c6df56a80e0f8ace9b44f3384d512daf06d75502f8b40fe

[root@ubuntu1804 ~]#docker volume  ls
DRIVER              VOLUME NAME
```

3.2.6 实战案例: 命名数据卷

3.2.6.1 创建命名数据卷

```
[root@ubuntu1804 ~]#docker volume  create vol1
vol1
[root@ubuntu1804 ~]#docker volume ls
DRIVER              VOLUME NAME
local               vol1
[root@ubuntu1804 ~]#docker inspect vol1
[
    {
        "CreatedAt": "2020-07-21T18:36:45+08:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/vol1/_data",
        "Name": "vol1",
        "Options": {},
        "Scope": "local"
    }
]
```

3.2.6.2 使用命名数据卷创建容器

```
[root@ubuntu1804 ~]#docker run -d  -p 8001:80 --name nginx01 -v
vol1:/usr/share/nginx/html  nginx
2a2ad99de2984521cbce777eae8e482c3e6e55e35ab5f398a35c7986fa39cc99
[root@ubuntu1804 ~]#curl 127.0.0.1:8001
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
```

```
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>

#显示命名数据卷
[root@ubuntu1804 ~]#docker volume ls
DRIVER              VOLUME NAME
local               vol1

#查看命名数据卷详解信息
[root@ubuntu1804 ~]#docker volume inspect vol1
[
    {
        "CreatedAt": "2020-07-21T18:44:03+08:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/vol1/_data",
        "Name": "vol1",
        "Options": {},
        "Scope": "local"
    }
]

[root@ubuntu1804 ~]#docker inspect --format="{{.Mounts}}"  nginx01
[{volume vol1 /var/lib/docker/volumes/vol1/_data /usr/share/nginx/html local z
true }]


#查看命名数据卷的文件
[root@ubuntu1804 ~]#ls /var/lib/docker/volumes/vol1/_data
50x.html  index.html

#修改宿主机命名数据卷的文件
[root@ubuntu1804 ~]#echo nginx vol1 website >
/var/lib/docker/volumes/vol1/_data/index.html
[root@ubuntu1804 ~]#curl 127.0.0.1:8001
nginx vol1 website

#利用现在的命名数据卷再创建新容器,可以和原有容器共享同一个命名数据卷的数据
[root@ubuntu1804 ~]#docker run -d  -p 8002:80 --name nginx02 -v
vol1:/usr/share/nginx/html  nginx
fb8739438992b7797a3f23dc495d2fd43f513fd793114848171f376c4325675e
[root@ubuntu1804 ~]#curl 127.0.0.1:8002
nginx vol1 website
```

3.2.6.3 创建容器时自动创建命名数据卷

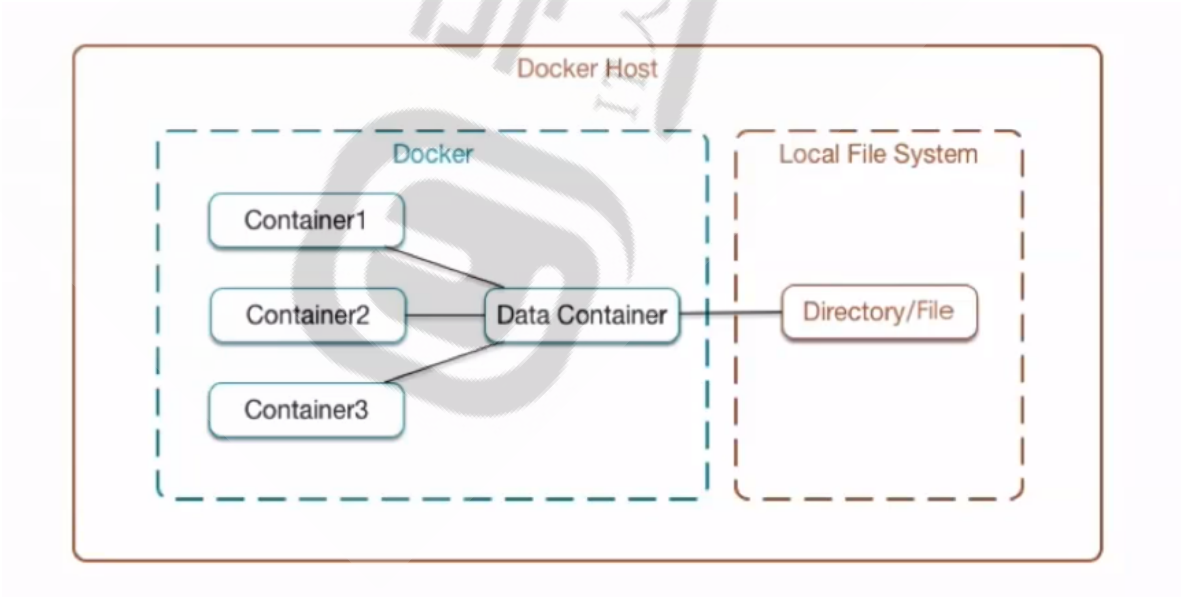
```
#创建容器自动创建命名数据卷
[root@ubuntu1804 ~]#docker run -d  -p 8003:80 --name nginx03 -v
vol2:/usr/share/nginx/html  nginx
5d7497c04907babcb2084f8f7a16874223e07965410fd4da14bc2fe993277aed

[root@ubuntu1804 ~]#docker volume ls
DRIVER              VOLUME NAME
local               portainer_data
local               vol1
local               vol2
```

3.2.6.4 删除数据卷

```
#删除指定的命名数据卷
[root@ubuntu1804 ~]#docker volume rm vol1

#清理全部不再使用的卷
[root@ubuntu1804 ~]#docker volume  prune -f
```

3.3 数据卷容器

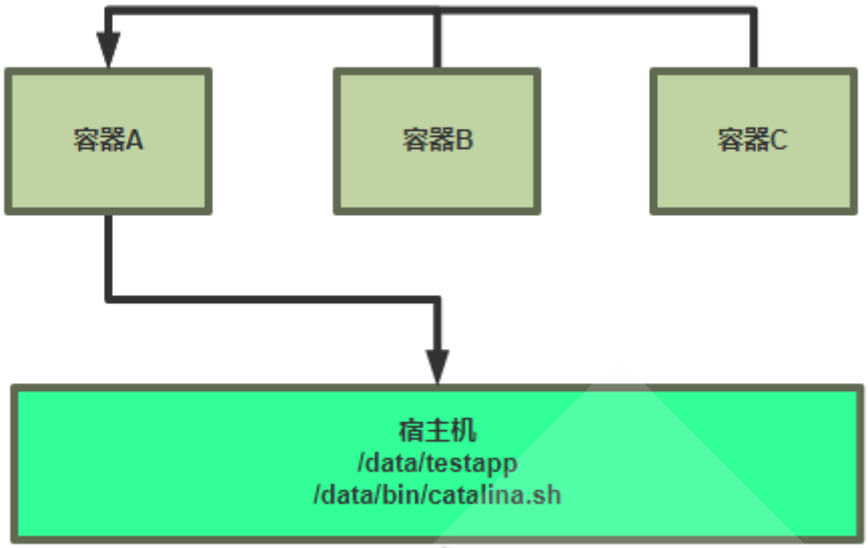
3.3.1 数据卷容器介绍



在Dockerfile中创建的是匿名数据卷,无法直接实现多个容器之间共享数据

数据卷容器最大的功能是可以让数据在多个docker容器之间共享

如下图所示: 即可以让B容器访问A容器的内容，而容器C也可以访问A容器的内容，即可以实现A，B，C三个容器之间的数据读写共享。



相当于先要创建一个后台运行的容器作为 Server，用于提供数据卷，这个卷可以为其他容器提供数据存储服务，其他使用此卷的容器作为client端，但此方法并不常使用

缺点：因为依赖一个 Server 的容器，所以此 Server 容器出了问题，其它 Client容器都会受影响

3.3.2 使用数据卷容器

启动容器时，指定使用数据卷容器

docker run 命令的以下选项可以实现数据卷容器，格式如下：

```
--volumes-from <数据卷容器>      Mount volumes from the specified container(s)
```

3.3.3 实战案例: 数据卷容器

3.3.3.1 创建一个数据卷容器 Server

先创建一个容器而无需启动也可以，并挂载宿主机的数据目录

范例：使用之前的镜像创建数据卷容器

```
#数据卷容器一般无需映射端口
[root@ubuntu1804 ~]#docker run -d --name volume-server -v
/data/bin/catalina.sh:/apps/tomcat/bin/catalina.sh:ro -v
/data/testapp:/data/tomcat/webapps/testapp   tomcat-web:app1
109a34c5a83b23bc57b16fc8b1e19104d3d522a8c62e1e98d8e69577b01739b5
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS               NAMES
109a34c5a83b        tomcat-web:app1     "/apps/tomcat/bin/ru…"   14 seconds ago
     Up 13 seconds       8009/tcp, 8080/tcp   volume-server
```

3.3.3.2 启动多个数据卷容器 Client

```
[root@ubuntu1804 ~]#docker run -d --name client1 --volumes-from volume-server -p
8081:8080  tomcat-web:app1
fe6ce0548dfee924cd39a8d86d5ed0e8ce9ea65323742f1336fa3b002c1b4a8c
[root@ubuntu1804 ~]#docker run -d --name client2 --volumes-from volume-server -p
8082:8080  tomcat-web:app1
10397838df9a489d7af211285Od4285bff5c2c262ea05cc9c5fb265af538f2c8

[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                                NAMES
10397838df9a        tomcat-web:app1     "/apps/tomcat/bin/ru…"   30 seconds ago
     Up 27 seconds       8009/tcp, 0.0.0.0:8082->8080/tcp   client2
fe6ce0548dfe        tomcat-web:app1     "/apps/tomcat/bin/ru…"   40 seconds ago
     Up 38 seconds       8009/tcp, 0.0.0.0:8081->8080/tcp   client1
109a34c5a83b        tomcat-web:app1     "/apps/tomcat/bin/ru…"   2 minutes ago
    Up 2 minutes        8009/tcp, 8080/tcp                   volume-server
[root@ubuntu1804 ~]#
```

3.3.3.3 验证访问

```
[root@ubuntu1804 ~]#curl 127.0.0.1:8081/testapp/
testapp v3
[root@ubuntu1804 ~]#curl 127.0.0.1:8082/testapp/
testapp v3
```

3.3.3.4 进入容器测试读写

读写权限依赖于源数据卷Server容器

```
#进入 Server 容器修改数据
[root@ubuntu1804 ~]#docker exec -it volume-server bash
[root@109a34c5a83b /]# cat /data/tomcat/webapps/testapp/index.html
testapp v3
[root@109a34c5a83b /]# echo testapp v4 > /data/tomcat/webapps/testapp/index.html
[root@109a34c5a83b /]#
[root@ubuntu1804 ~]#curl 127.0.0.1:8081/testapp/
testapp v4
[root@ubuntu1804 ~]#curl 127.0.0.1:8082/testapp/
testapp v4

#进入 Client 容器修改数据
[root@ubuntu1804 ~]#docker exec -it client1 bash
[root@fe6ce0548dfe /]# cat /data/tomcat/webapps/testapp/index.html
testapp v4
[root@fe6ce0548dfe /]# echo testapp v4 > /data/tomcat/webapps/testapp/index.html
[root@fe6ce0548dfe /]# cat /data/tomcat/webapps/testapp/index.html
testapp v5

[root@ubuntu1804 ~]#curl 127.0.0.1:8081/testapp/
testapp v5
[root@ubuntu1804 ~]#curl 127.0.0.1:8082/testapp/
testapp v5
```

3.3.3.5 在宿主机直接修改

```
[root@ubuntu1804 ~]#cat /data/testapp/index.html
testapp v5
[root@ubuntu1804 ~]#echo testapp v6 > /data/testapp/index.html
[root@ubuntu1804 ~]#cat /data/testapp/index.html
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8081/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8082/testapp/
testapp v6
[root@ubuntu1804 ~]#

[root@ubuntu1804 ~]#docker exec -it volume-server bash
[root@109a34c5a83b /]# cat  /data/tomcat/webapps/testapp/index.html
testapp v6
[root@109a34c5a83b /]#
```

3.3.3.6 关闭卷容器Server测试能否启动新容器

关闭卷容器Server，仍然可以创建新的client容器及访问旧的client容器

```
[root@ubuntu1804 ~]#docker stop volume-server
volume-server
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                          NAMES
10397838df9a        tomcat-web:app1     "/apps/tomcat/bin/ru…"   9 minutes ago
    Up 9 minutes        8009/tcp, 0.0.0.0:8082->8080/tcp   client2
fe6ce0548dfe        tomcat-web:app1     "/apps/tomcat/bin/ru…"   10 minutes ago
     Up 10 minutes       8009/tcp, 0.0.0.0:8081->8080/tcp   client1
[root@ubuntu1804 ~]#docker run -d --name client3 --volumes-from volume-server -p
8083:8080  tomcat-web:app1
458df991688c3dbc69d824f889616e0a7534ce18543c8559162f3609fbb26b53

[root@ubuntu1804 ~]#docker ps  -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS                         PORTS                                NAMES
458df991688c        tomcat-web:app1     "/apps/tomcat/bin/ru…"   About a minute
ago   Up About a minute                8009/tcp, 0.0.0.0:8083->8080/tcp
client3
10397838df9a        tomcat-web:app1     "/apps/tomcat/bin/ru…"   11 minutes ago
      Up 11 minutes                     8009/tcp, 0.0.0.0:8082->8080/tcp
client2
fe6ce0548dfe        tomcat-web:app1     "/apps/tomcat/bin/ru…"   11 minutes ago
      Up 11 minutes                     8009/tcp, 0.0.0.0:8081->8080/tcp
client1
109a34c5a83b        tomcat-web:app1     "/apps/tomcat/bin/ru…"   13 minutes ago
      Exited (137) About a minute ago
volume-server

[root@ubuntu1804 ~]#curl 127.0.0.1:8081/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8082/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8083/testapp/
```

```
testapp v6
```

3.3.3.7 删除源卷容器Server，访问client和创建新的client容器

删除数据卷容器后，旧的client 容器仍能访问，但无法再创建新的client容器

```
[root@ubuntu1804 ~]#docker rm -fv volume-server
volume-server
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED
    STATUS              PORTS                                NAMES
458df991688c        tomcat-web:app1     "/apps/tomcat/bin/ru…"   4 minutes ago
    Up 4 minutes        8009/tcp, 0.0.0.0:8083->8080/tcp   client3
10397838df9a        tomcat-web:app1     "/apps/tomcat/bin/ru…"   14 minutes ago
     Up 14 minutes       8009/tcp, 0.0.0.0:8082->8080/tcp   client2
fe6ce0548dfe        tomcat-web:app1     "/apps/tomcat/bin/ru…"   14 minutes ago
     Up 14 minutes       8009/tcp, 0.0.0.0:8081->8080/tcp   client1


[root@ubuntu1804 ~]#docker run -d --name client4 --volumes-from volume-server -p
8084:8080  tomcat-web:app1
Unable to find image 'tomcat-web:app1' locally
docker: Error response from daemon: pull access denied for tomcat-web,
repository does not exist or may require 'docker login': denied: requested
access to the resource is denied.
See 'docker run --help'.

[root@ubuntu1804 ~]#curl 127.0.0.1:8081/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8082/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8083/testapp/
testapp v6
```

3.3.3.8 重新创建容器卷 Server

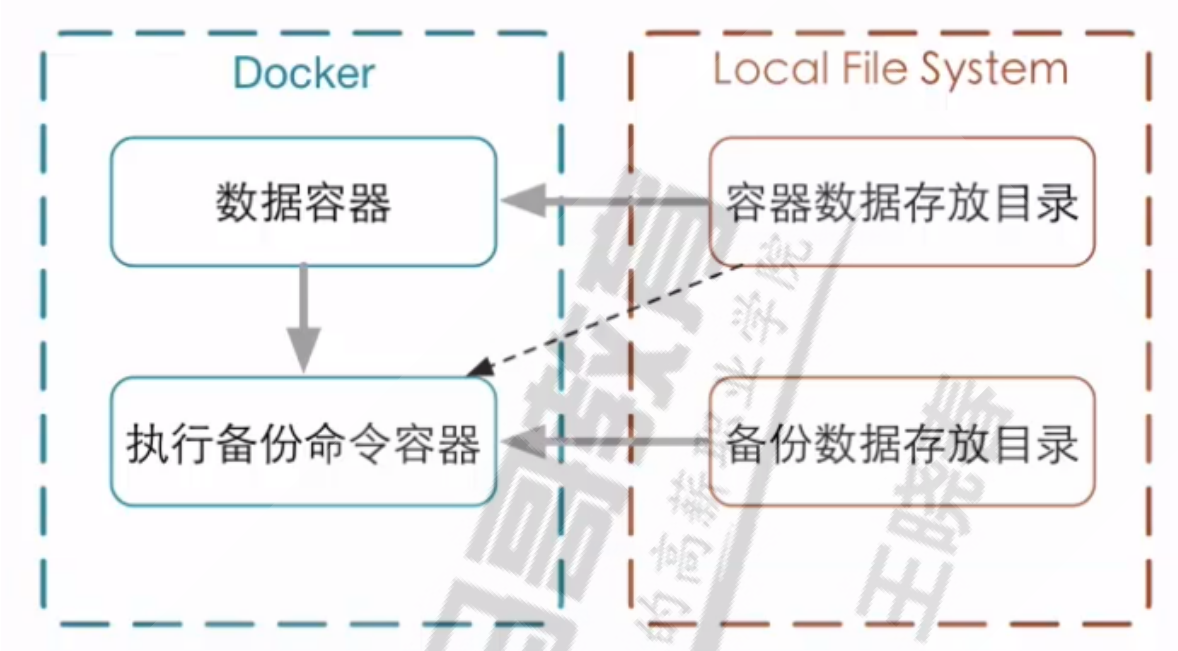
重新创建容器卷容器后，还可继续创建新client 容器

```
[root@ubuntu1804 ~]#docker run -d --name volume-server -v
/data/bin/catalina.sh:/apps/tomcat/bin/catalina.sh:ro -v
/data/testapp:/data/tomcat/webapps/testapp   tomcat-web:app1
ee0dacff6a4531dc8ecc1d416b449e087d8bf27dad2d6a6e515faf10455a1cc0

[root@ubuntu1804 ~]#docker run -d --name client4 --volumes-from volume-server -p
8084:8080  tomcat-web:app1
dcd002fe87887a0c5d1f62b24cd7665e42ee864c975d1a2953c383af0cb65a70

[root@ubuntu1804 ~]#curl 127.0.0.1:8084/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8081/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8082/testapp/
testapp v6
```

```
[root@ubuntu1804 ~]#curl 127.0.0.1:8083/testapp/
testapp v6
[root@ubuntu1804 ~]#curl 127.0.0.1:8084/testapp/
testapp v6
```

3.3.4 利用数据卷容器备份指定容器的数据卷实现

由于匿名数据卷在宿主机中的存储位置不确定,所以为了方便的备份匿名数据卷,可以利用数据卷容器实现数据卷的备份




```
#在执行备份命令容器上执行备份方式
docker run -it --rm --volumes-from [container name] -v $(pwd):/backup ubuntu
root@ca5bb2c1f877:/#tar cvf /backup/backup.tar [container data volume]

#说明
[container name]            #表示需要备份的容器
[container data volume]     #表示容器内的需要备份的数据卷对应的目录

#还原方式
docker run -it --rm --volumes-from [container name] -V $(pwd):/backup ubuntu
root@ca5bb2c1f877:/#tar xvf /backup/backup.tar -C [container data volume]
```

范例:

```
#创建需要备份的数据卷容器
[root@ubuntu1804 ~]#docker run -it -v /datavolume1  --name volume-server centos
bash
[root@88bbc22a3072 /]# ls
bin      dev  home  lib64       media  opt   root  sbin  sys  usr
datavolume1  etc  lib   lost+found  mnt    proc  run   srv   tmp  var
[root@88bbc22a3072 /]# touch /datavolume1/centos.txt
[root@88bbc22a3072 /]# exit
exit
[root@ubuntu1804 ~]#docker ps
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
#可以看到数据卷对应的宿主机目录
[root@ubuntu1804 ~]#docker inspect --format="{{.Mounts}}" volume-server
[{volume 21199be9bff7ce98926f673d2250d64d75aaa00c5d375a2528b2b47f0ec4cb93
/var/lib/docker/volumes/21199be9bff7ce98926f673d2250d64d75aaa00c5d375a2528b2b47f
0ec4cb93/_data /datavolume1 local  true }]

#利用数据卷容器创建执行备份操作容器
[root@ubuntu1804 ~]#docker run -it --rm --volumes-from volume-server -v
~/backup:/backup --name backup-server ubuntu
root@9ad8be4b5810:/# ls
backup  boot         dev  home  lib32  libx32  mnt  proc  run   srv  tmp  var
bin     datavolume1  etc  lib   lib64  media   opt  root  sbin  sys  usr
root@9ad8be4b5810:/# ls /backup/
root@9ad8be4b5810:/# ls  /datavolume1/
centos.txt
root@9ad8be4b5810:/# cd /datavolume1/
root@9ad8be4b5810:/datavolume1# tar cvf /backup/data.tar .
./
./centos.txt
root@9ad8be4b5810:/datavolume1# exit
exit
[root@ubuntu1804 ~]#ls ~/backup/
data.tar


#删除容器的数据
[root@ubuntu1804 ~]#docker start -i volume-server
[root@88bbc22a3072 /]# rm -rf /datavolume1/*
[root@88bbc22a3072 /]# ls /datavolume1/
[root@88bbc22a3072 /]# exit
exit

#进行还原
[root@ubuntu1804 ~]#docker run --rm --volumes-from volume-server -v
~/backup:/backup --name backup-server ubuntu tar xvf /backup/data.tar -C
/datavolume1/
./
./centos.txt

#验证是否还原
[root@ubuntu1804 ~]#docker start -i volume-server
[root@88bbc22a3072 /]# ls /datavolume1/
centos.txt
```

3.3.5 数据卷容器总结

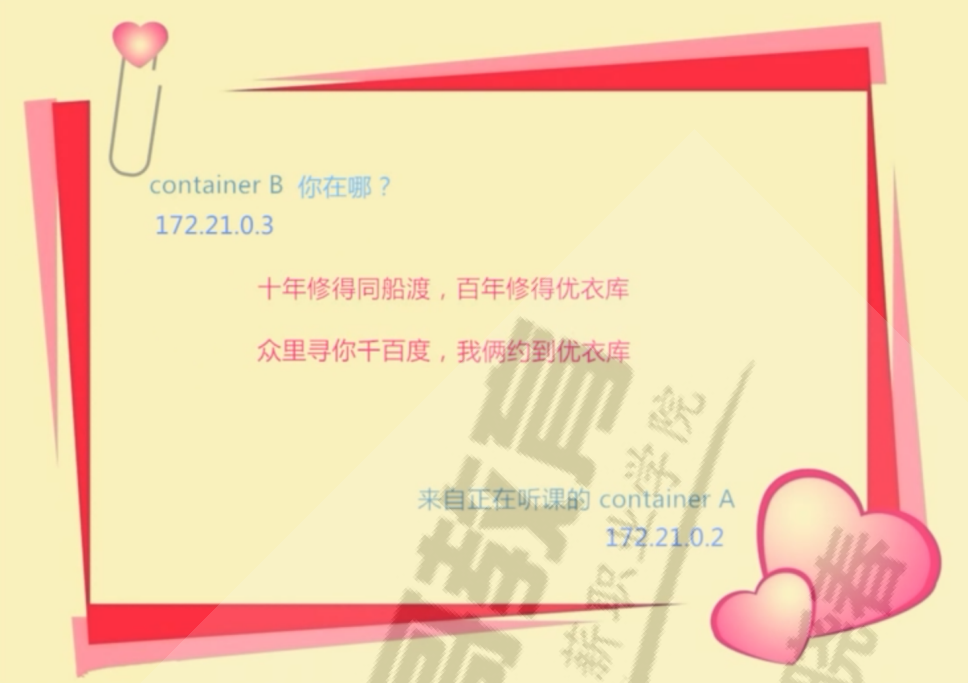
将提供卷的容器Server 删除，已经运行的容器Client依然可以使用挂载的卷，因为容器是通过挂载访问数据的，但是无法创建新的卷容器客户端，但是再把卷容器Server创建后即可正常创建卷容器Client，此方式可以用于线上共享数据目录等环境，因为即使数据卷容器被删除了，其他已经运行的容器依然可以挂载使用

由此可知, 数据卷容器的功能只是将数据挂载信息传递给了其它使用数据卷容器的容器,而数据卷容器本身并不提供数据存储功能

数据卷容器可以作为共享的方式为其他容器提供文件共享，类似于NFS共享，可以在生产中启动一个实例挂载本地的目录，然后其他的容器分别挂载此容器的目录，即可保证各容器之间的数据一致性

数据卷容器的 Server 和 Client 可以不使用同一个镜像生成

4 网络管理



docker容器创建后，必不可少的要和其它主机或容器进行网络通信

官方文档:

```
https://docs.docker.com/network/
```

4.1 Docker的默认的网络通信

4.1.1 Docker安装后默认的网络设置

Docker服务安装完成之后，默认在每个宿主机会生成一个名称为docker0的网卡其IP地址都是172.17.0.1/16

范例: 安装Docker的默认的网络配置

```
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
```

```
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:02:7f:a8:c6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2ff:fe7f:a8c6/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#brctl show
bridge name bridge id       STP enabled interfaces
docker0     8000.0242027fa8c6   no
```

范例: 安装Harbor的默认网络配置

```
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:01:f3:0c brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.102/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe01:f30c/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:f4:23:e8:29 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
4: br-9af624ecd23e: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP group default
    link/ether 02:42:e9:1c:1a:7b brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.1/16 brd 172.18.255.255 scope global br-9af624ecd23e
       valid_lft forever preferred_lft forever
    inet6 fe80::42:e9ff:fe1c:1a7b/64 scope link
       valid_lft forever preferred_lft forever
6: veth225895c@if5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether a6:f3:0f:ae:4b:43 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::a4f3:fff:feae:4b43/64 scope link
       valid_lft forever preferred_lft forever
8: veth244c237@if7: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether 72:12:35:11:e8:14 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::7012:35ff:fe11:e814/64 scope link
       valid_lft forever preferred_lft forever
10: veth81ab8cb@if9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether 5e:07:f2:eb:43:c2 brd ff:ff:ff:ff:ff:ff link-netnsid 2
    inet6 fe80::5c07:f2ff:feeb:43c2/64 scope link
```

```
       valid_lft forever preferred_lft forever
12: vethf8499d4@if11: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether 4e:df:12:c5:58:83 brd ff:ff:ff:ff:ff:ff link-netnsid 4
    inet6 fe80::4cdf:12ff:fec5:5883/64 scope link
       valid_lft forever preferred_lft forever
14: vethceabf74@if13: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether 06:c0:58:ea:51:2e brd ff:ff:ff:ff:ff:ff link-netnsid 5
    inet6 fe80::4c0:58ff:feea:512e/64 scope link
       valid_lft forever preferred_lft forever
16: veth47c5069@if15: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether c6:6f:aa:51:be:38 brd ff:ff:ff:ff:ff:ff link-netnsid 3
    inet6 fe80::c46f:aaff:fe51:be38/64 scope link
       valid_lft forever preferred_lft forever
18: veth83fde4a@if17: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether 32:74:1e:e2:81:50 brd ff:ff:ff:ff:ff:ff link-netnsid 6
    inet6 fe80::3074:1eff:fee2:8150/64 scope link
       valid_lft forever preferred_lft forever
20: veth2c51f87@if19: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether ca:b7:c9:da:87:92 brd ff:ff:ff:ff:ff:ff link-netnsid 7
    inet6 fe80::c8b7:c9ff:feda:8792/64 scope link
       valid_lft forever preferred_lft forever
22: veth0f4a931@if21: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether fa:29:a4:4d:b1:c2 brd ff:ff:ff:ff:ff:ff link-netnsid 8
    inet6 fe80::f829:a4ff:fe4d:b1c2/64 scope link
       valid_lft forever preferred_lft forever
24: veth55b6555@if23: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master br-9af624ecd23e state UP group default
    link/ether aa:87:c4:2c:de:7c brd ff:ff:ff:ff:ff:ff link-netnsid 9
    inet6 fe80::a887:c4ff:fe2c:de7c/64 scope link
       valid_lft forever preferred_lft forever


[root@ubuntu1804 ~]#apt -y install bridge-utils
[root@ubuntu1804 ~]#brctl show
bridge name bridge id       STP enabled interfaces
br-9af624ecd23e     8000.0242e91c1a7b   no      veth0f4a931
                            veth225895c
                            veth244c237
                            veth2c51f87
                            veth47c5069
                            veth55b6555
                            veth81ab8cb
                            veth83fde4a
                            vethceabf74
                            vethf8499d4
docker0     8000.0242f423e829   no
```

4.1.2 创建容器后的网络配置

每次新建容器后

- 宿主机多了一个虚拟网卡，和容器的网卡组合成一个网卡，比如：137: veth8ca6d43@if136，而在容器内的网卡名为136，可以看出和宿主机的网卡之间的关联
- 容器会自动获取一个172.17.0.0/16网段的随机地址，默认从172.17.0.2开始，第二次容器为172.17.0.3，以此类推
- 容器获取的地址并不固定,每次容器重启,可能会发生地址变化

4.1.2.1 创建第一个容器后的网络状态

范例：创建容器，容器自动获取IP地址

```
[root@ubuntu1804 ~]#docker run -it --rm  alpine:3.11 sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
136: eth0@if137: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  6b8d9f3a653e

[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
6b8d9f3a653e        alpine:3.11         "sh"                13 seconds ago
Up 12 seconds                           pensive_chandrasekhar
```

范例：新建第一个容器，宿主机的网卡多了一个新网卡

```
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
```

```
    link/ether 02:42:02:7f:a8:c6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2ff:fe7f:a8c6/64 scope link
       valid_lft forever preferred_lft forever
137: veth8ca6d43@if136: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether fa:96:37:77:a9:a9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::f896:37ff:fe77:a9a9/64 scope link
       valid_lft forever preferred_lft forever
```

范例: 查看新建容器后桥接状态

```
[root@ubuntu1804 ~]#brctl show
bridge name bridge id       STP enabled interfaces
docker0     8000.0242027fa8c6   no      veth8ca6d43
```

4.1.2.2 创建第二个容器后面的网络状态

范例: 再次创建第二个容器

```
[root@ubuntu1804 ~]#docker run -it --rm  alpine:3.11 sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
140: eth0@if141: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
 / # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.3  ab3ea580804a
/ # ping ab3ea580804a
PING ab3ea580804a (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.037 ms
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.132 ms
^C
--- ab3ea580804a ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.037/0.084/0.132 ms
/ # ping 6b8d9f3a653e
ping: bad address '6b8d9f3a653e'
/ #


[root@ubuntu1804 ~]#docker ps
```

```
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
ab3ea580804a        alpine:3.11         "sh"                9 seconds ago
Up 7 seconds                            vigilant_jones
6b8d9f3a653e        alpine:3.11         "sh"                13 seconds ago
Up 12 seconds                           pensive_chandrasekhar
```

范例: 新建第二个容器后宿主机又多了一个虚拟网卡

```
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:02:7f:a8:c6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2ff:fe7f:a8c6/64 scope link
       valid_lft forever preferred_lft forever
137: veth8ca6d43@if136: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether fa:96:37:77:a9:a9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::f896:37ff:fe77:a9a9/64 scope link
       valid_lft forever preferred_lft forever
141: vethf599a47@if140: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether 96:e7:52:fe:67:54 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::94e7:52ff:fefe:6754/64 scope link
       valid_lft forever preferred_lft forever
```

范例: 查看新建第二个容器后桥接状态

```
[root@ubuntu1804 ~]#brctl  show
bridge name bridge id       STP enabled interfaces
docker0     8000.0242027fa8c6   no      veth8ca6d43
                                        vethf599a47
```

4.1.3 容器间的通信

4.1.3.1 同一个宿主机的不同容器可相互通信

默认情况下

- 同一个宿主机的不同容器之间可以相互通信

```
dockerd   --icc    Enable inter-container communication (default true)
--icc=false  #此配置可以禁止同一个宿主机的容器之间通信
```

- 不同宿主机之间的容器IP地址重复，默认不能相互通信

范例: 同一个宿主机的容器之间访问

```
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:02:7f:a8:c6 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:2ff:fe7f:a8c6/64 scope link
       valid_lft forever preferred_lft forever
137: veth8ca6d43@if136: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether fa:96:37:77:a9:a9 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::f896:37ff:fe77:a9a9/64 scope link
       valid_lft forever preferred_lft forever
141: vethf599a47@if140: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
master docker0 state UP group default
    link/ether 96:e7:52:fe:67:54 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::94e7:52ff:fefe:6754/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#docker run -it --rm  alpine:3.11 sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
136: eth0@if137: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.452 ms
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.190 ms
^C
```

```
--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.190/0.321/0.452 ms
/ # ping 172.17.0.1
PING 172.17.0.1 (172.17.0.1): 56 data bytes
64 bytes from 172.17.0.1: seq=0 ttl=64 time=0.139 ms
64 bytes from 172.17.0.1: seq=1 ttl=64 time=0.183 ms
^C
--- 172.17.0.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.139/0.161/0.183 ms
/ #
```

4.1.3.2 禁止同一个宿主机的不同容器间通信

范例: 同一个宿主机不同容器间禁止通信

```
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
--icc=false
[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker

#创建两个容器,测试无法通信
[root@ubuntu1804 ~]#docker run -it --name test1 --rm alpine  sh
/ # hostname -i
172.17.0.2
[root@ubuntu1804 ~]#docker run -it --name test2 --rm alpine  sh
/ # hostname -i
172.17.0.3
/ # ping 172.17.0.2
```

范例: 在第二个宿主机上创建容器，跨宿主机的容器之间默认不能通信

```
[root@ubuntu1804 ~]#docker pull alpine
Using default tag: latest
latest: Pulling from library/alpine
c9b1b535fdd9: Pull complete
Digest: sha256:ab00606a42621fb68f2ed6ad3c88be54397f981a7b70a79db3d1172b11c4367d
Status: Downloaded newer image for alpine:latest
docker.io/library/alpine:latest
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
cab76bbb3db2        alpine              "sh"                About a minute ago
Up About a minute                       jolly_rosalind
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:6b:54:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.101/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6b:54d3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:1d:73:8b:71 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@ubuntu1804 ~]#docker run -it --rm  alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: eth0@if5: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping -c 1 172.17.0.3
PING 172.17.0.3 (172.17.0.3): 56 data bytes

--- 172.17.0.3 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

4.1.4 修改默认网络设置

新建容器默认使用docker0的网络配置,可以修改默认指向自定义的网桥网络

范例: 用自定义的网桥代替默认的docker0

```
#查看默认网络
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:40:27:06 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe40:2706/64 scope link
       valid_lft forever preferred_lft forever
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
```

```
    link/ether 02:42:35:ba:e7:ce brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:35ff:feba:e7ce/64 scope link
       valid_lft forever preferred_lft forever

[root@ubuntu1804 ~]#apt -y install bridge-utils
[root@ubuntu1804 ~]#brctl addbr br0
[root@ubuntu1804 ~]#ip a a 192.168.100.1/24 dev br0
[root@ubuntu1804 ~]#brctl  show
bridge name bridge id       STP enabled interfaces
br0     8000.000000000000   no
docker0     8000.024235bae7ce   no
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:40:27:06 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe40:2706/64 scope link
       valid_lft forever preferred_lft forever
5: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:35:ba:e7:ce brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:35ff:feba:e7ce/64 scope link
       valid_lft forever preferred_lft forever
18: br0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
group default qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.1/24 scope global br0
       valid_lft forever preferred_lft forever
    inet6 fe80::9cf2:e0ff:fe4e:96b3/64 scope link
       valid_lft forever preferred_lft forever


[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
-b br0

[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#ps -ef |grep dockerd
root      15773      1  0 11:12 ?        00:00:00 /usr/bin/dockerd -H fd:// --
containerd=/run/containerd/containerd.sock -b br0
root      15960   2300  0 11:14 pts/0    00:00:00 grep --color=auto dockerd

[root@ubuntu1804 ~]#docker run  --rm alpine hostname -i
192.168.100.2
```

4.2 容器名称互联

新建容器时，docker会自动分配容器名称，容器ID和IP地址，导致容器名称，容器ID和IP都不固定，那么如何区分不同的容器，实现和确定目标容器的通信呢？解决方案是给容器起个固定的名称，容器之间通过固定名称实现确定目标的通信

有两种固定名称:

- 容器名称
- 容器名称的别名

注意: 两种方式都最少需要两个容器才能实现

4.2.1 通过容器名称互联

4.2.1.1 容器名称介绍

即在同一个宿主机上的容器之间可以通过自定义的容器名称相互访问，比如: 一个业务前端静态页面是使用nginx，动态页面使用的是tomcat，另外还需要负载均衡调度器，如: haproxy 对请求调度至nginx和tomcat的容器，由于容器在启动的时候其内部IP地址是DHCP 随机分配的，而给容器起个固定的名称，则是相对比较固定的，因此比较适用于此场景

注意: 如果被引用的容器地址变化,必须重启当前容器才能生效

4.2.1.2 容器名称实现

docker run 创建容器，可使用--link选项实现容器名称的引用

```
--link list                                  #Add link to another container
格式:
docker run --name <容器名称>                    #先创建指定名称的容器
docker run --link <目标通信的容器ID或容器名称>     #再创建容器时引用上面容器的名称
```

4.2.1.3 实战案例1: 使用容器名称进行容器间通信

1. **先创建第一个指定容器名称的容器**

```
[root@ubuntu1804 ~]#docker run -it --name server1 --rm  alpine:3.11 sh
/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  cdb5173003f5
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
150: eth0@if151: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
```

```
       valid_lft forever preferred_lft forever
/ # ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.038 ms
^C
--- 172.17.0.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.038/0.038/0.038 ms
/ # ping server1
ping: bad address 'server1'
/ # ping cdb5173003f5
PING cdb5173003f5 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.040 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.128 ms
^C
--- cdb5173003f5 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.040/0.084/0.128 ms
/ #
```

2. **新建第二个容器时引用第一个容器的名称**

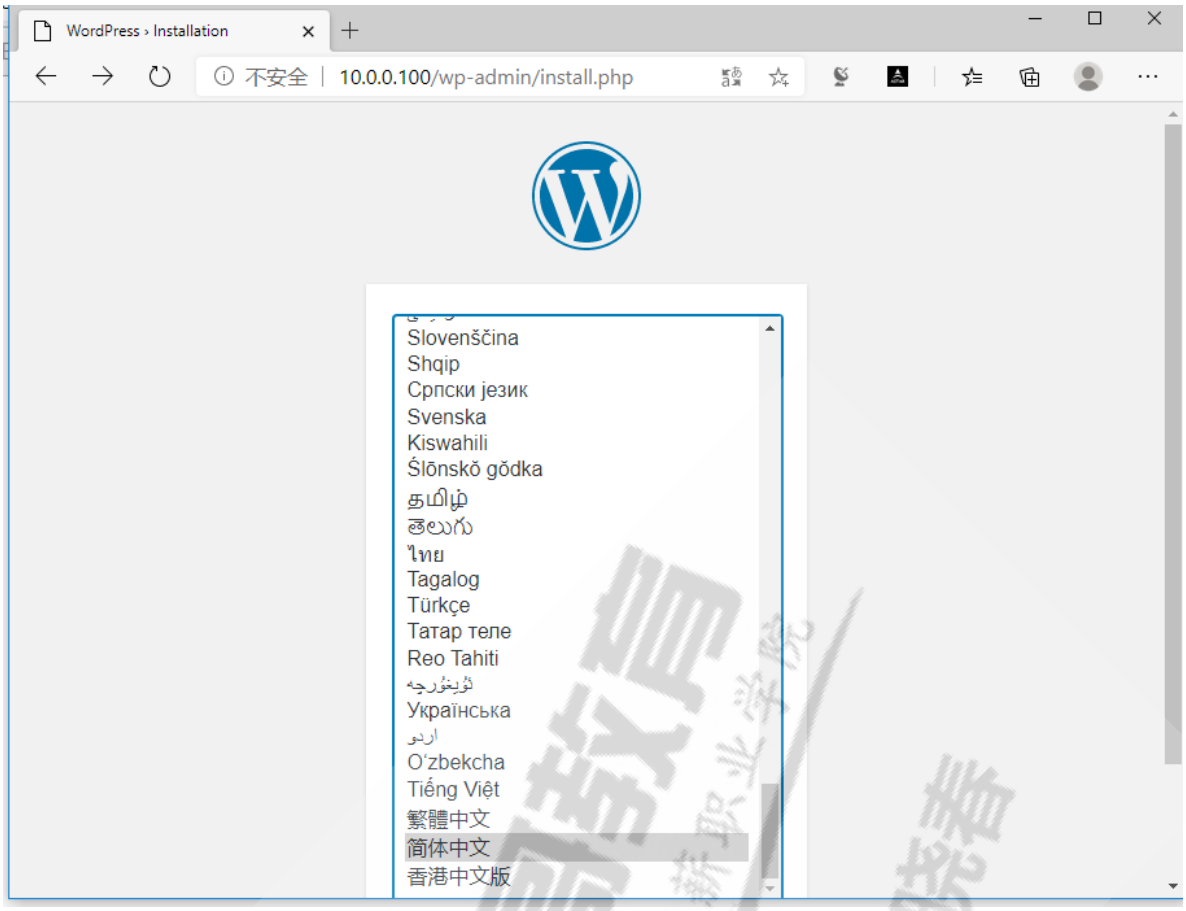
   会自动将第一个主机的名称加入/etc/hosts文件,从而可以利用第一个容器名称进行访问

```
[root@ubuntu1804 ~]#docker run -it  --rm --name server2  --link server1
alpine:3.11 sh
/ # env
HOSTNAME=395d8c3392ee
SHLVL=1
HOME=/root
TERM=xterm
SERVER1_NAME=/server2/server1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/

/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  server1 cdb5173003f5
172.17.0.3  7ca466320980
/ # ping server1
PING server1 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.111 ms
^C
--- server1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.111/0.111/0.111 ms
/ # ping server2
ping: bad address 'server2'
/ # ping 7ca466320980
PING 7ca466320980 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.116 ms
```

```
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.069 ms
^C
--- 7ca466320980 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.069/0.092/0.116 ms
/ # ping cdb5173003f5
PING cdb5173003f5 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.072 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.184 ms
^C
--- cdb5173003f5 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.072/0.128/0.184 ms
/ #

[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
    STATUS              PORTS               NAMES
7ca466320980        alpine:3.11         "sh"                24 seconds ago
   Up 23 seconds                           server2
cdb5173003f5        alpine:3.11         "sh"                7 minutes ago
    Up 7 minutes                            server1
```

4.2.1.4 实战案例2: 实现 wordpress 和 MySQL 两个容器互连

```
[root@centos7 ~]#tree lamp_docker/
lamp_docker/
├── env_mysql.list
├── env_wordpress.list
└── mysql
    └── mysql_test.cnf

1 directory, 3 files
[root@centos7 ~]#cat lamp_docker/env_mysql.list
MYSQL_ROOT_PASSWORD=123456
MYSQL_DATABASE=wordpress
MYSQL_USER=wpuser
MYSQL_PASSWORD=wppass
[root@centos7 ~]#cat lamp_docker/env_wordpress.list
WORDPRESS_DB_HOST=mysql:3306
WORDPRESS_DB_NAME=wordpress
WORDPRESS_DB_USER=wpuser
WORDPRESS_DB_PASSWORD=wppass
WORDPRESS_TABLE_PREFIX=wp_
[root@centos7 ~]#cat lamp_docker/mysql/mysql_test.cnf
[mysqld]
server-id=100
log-bin=mysql-bin


[root@centos7 ~]#docker run --name mysql -v
/root/lamp_docker/mysql/:/etc/mysql/conf.d -v /data/mysql:/var/lib/mysql --env-
file=/root/lamp_docker/env_mysql.list -d -p 3306:3306 mysql:5.7.30

[root@centos7 ~]#docker run -d   --name wordpress  --link mysql  --env-
file=/root/lamp_docker/env_wordpress.list  -p 80:80 wordpress
```



4.2.2 通过自定义容器别名互联

4.2.2.1 容器别名介绍

自定义的容器名称可能后期会发生变化，那么一旦名称发生变化，容器内程序之间也必须要随之发生变化，比如:程序通过固定的容器名称进行服务调用，但是容器名称发生变化之后再使用之前的名称肯定是无法成功调用，每次都进行更改的话又比较麻烦，因此可以使用自定义别名的方式解决，即容器名称可以随意更改，只要不更改别名即可

4.2.2.2 容器别名实现

命令格式:

```
docker run --name <容器名称>                                    #先创建指定名称的容器
docker run -d --name 容器名称 --link <目标容器名称>:<容器别名>    #给上面创建的容器起别名,来创建新容器
```

4.2.2.3 实战案例: 使用容器别名

范例: 创建第三个容器，引用前面创建的容器，并起别名

```
[root@ubuntu1804 ~]#docker run -it  --rm --name server3  --link server1:server1-alias  alpine:3.11 sh
/ # env
HOSTNAME=395d8c3392ee
SHLVL=1
HOME=/root
TERM=xterm
SERVER1-ALIAS_NAME=/server3/server1-alias
```

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/

/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  server1-alias cdb5173003f5 server1
172.17.0.4  d9622c6831f7
/ # ping server1
PING server1 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.101 ms
^C
--- server1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.101/0.101/0.101 ms
/ # ping server1-alias
PING server1-alias (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.073 ms
^C
--- server1-alias ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.073/0.073/0.073 ms
/ #
```

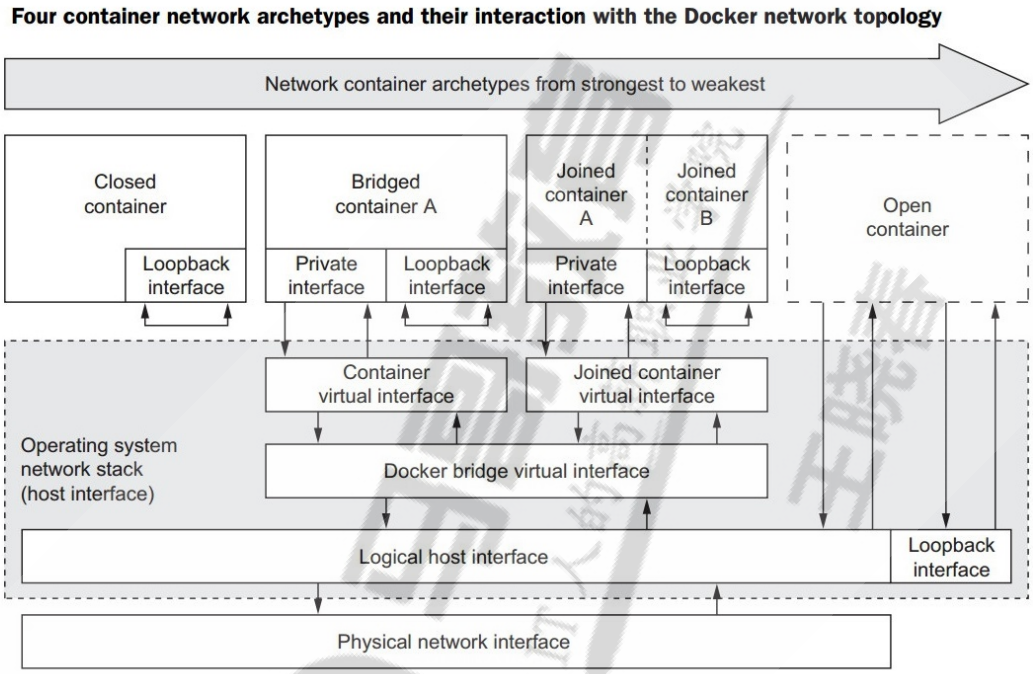
范例: 创建第四个容器，引用前面创建的容器，并起多个别名

```
[root@ubuntu1804 ~]#docker run -it  --rm --name server4  --link
server1:"server1-alias1 server1-alias2"  alpine:3.11 sh
/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  server1-alias1 server1-alias2 cdb5173003f5 server1
172.17.0.5  db3d2f084c05
/ # ping server1
PING server1 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.107 ms
^C
--- server1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.107/0.107/0.107 ms
/ # ping server1-alias1
PING server1-alias1 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.126 ms
^C
--- server1-alias1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.126/0.126/0.126 ms
/ # ping server1-alias2
```

```
PING server1-alias2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.124 ms
^C
--- server1-alias2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.124/0.124/0.124 ms
/ #
```

4.3 docker 网络连接模式

4.3.1 网络模式介绍



Docker 的网络支持5种网络模式:

- none
- bridge
- host
- container
- network-name

范例: 查看默认的网络模式

```
[root@ubuntu1804 ~]#docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
fe08e6d23c4c        bridge              bridge              local
cb64aa83626c        host                host                local
10619d45dcd4        none                null                local
```

4.3.2 网络模式指定

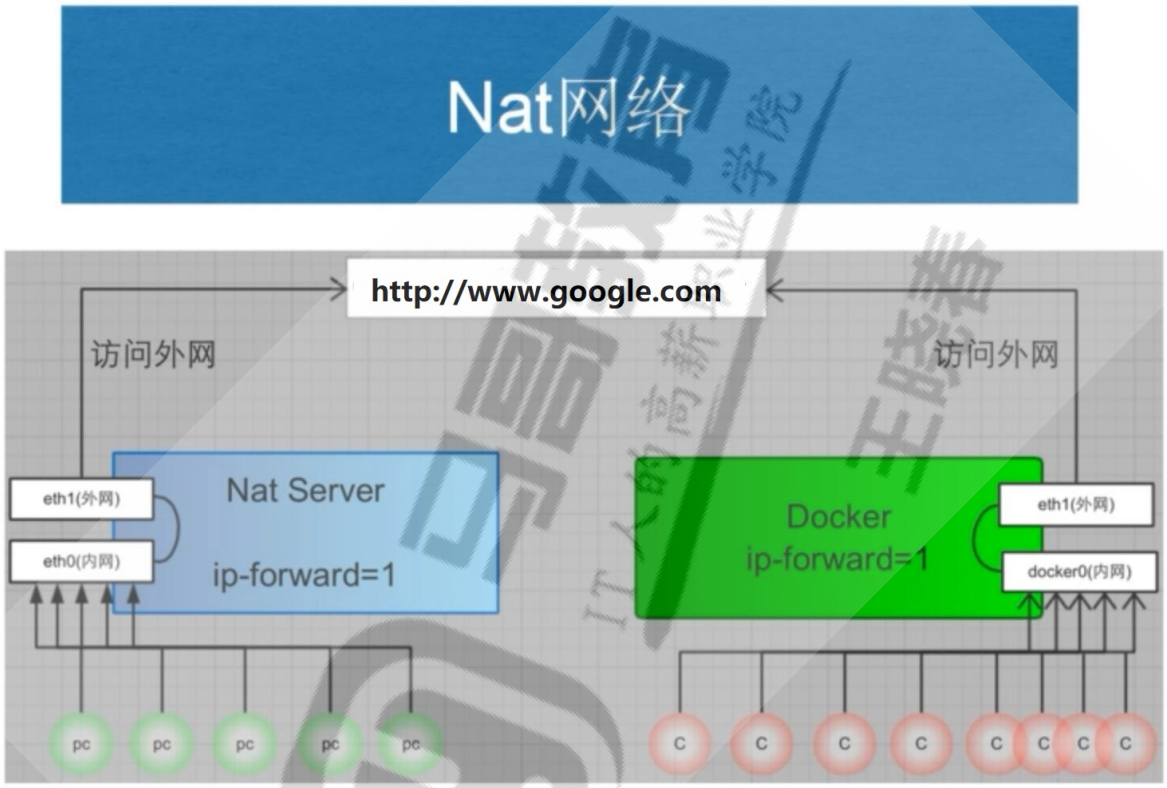
默认新建的容器使用Bridge模式，创建容器时，docker run 命令使用以下选项指定网络模式

格式

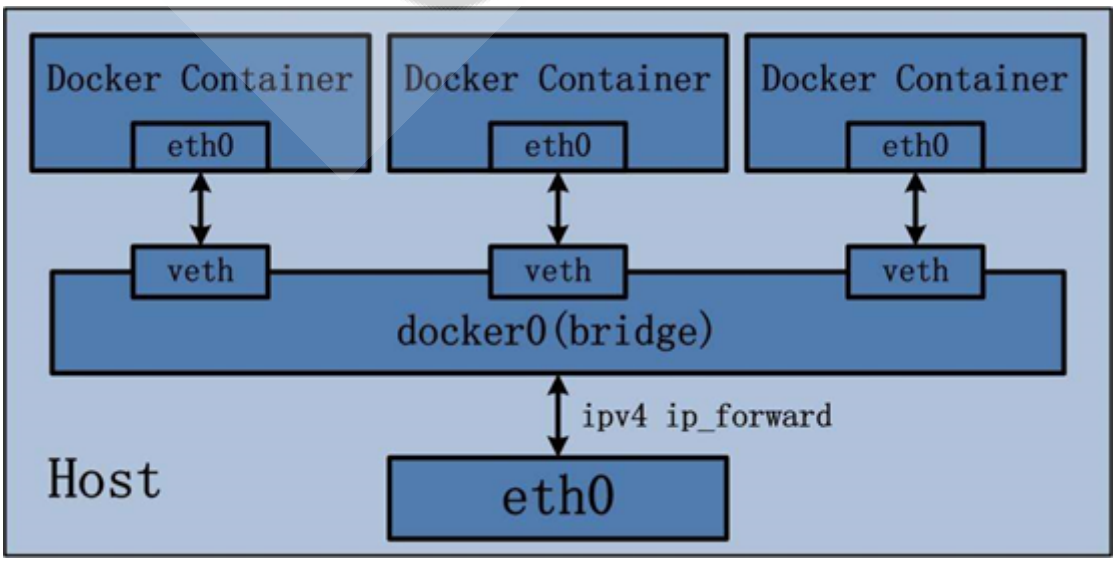
```
docker run --network <mode>
docker run --net=<mode>

<mode>:  可是以下值
none
bridge
host
container:<容器名或容器ID>
<自定义网络名称>
```

4.2.3 bridge网络模式

4.2.3.1 bridge 网络模式架构



本模式是docker的默认模式，即不指定任何模式就是bridge模式，也是使用比较多的模式，此模式创建的容器会为每一个容器分配自己的网络 IP 等信息，并将容器连接到一个虚拟网桥与外界通信



可以和外部网络之间进行通信，通过SNAT访问外网，使用DNAT可以让容器被外部主机访问，所以此模式也称为NAT模式

此模式宿主机需要启动ip_forward功能

bridge网络模式特点

- 网络资源隔离：不同宿主机的容器无法直接通信，各自使用独立网络
- 无需手动配置：容器默认自动获取172.17.0.0/16的IP地址，此地址可以修改
- 可访问外网：利用宿主机的物理网卡，SNAT连接外网
- 外部主机无法直接访问容器：可以通过配置DNAT接受外网的访问
- 低性能较低：因为可通过NAT，网络转换带来更的损耗
- 端口管理繁琐：每个容器必须手动指定唯一的端口，容器产生端口冲容

4.2.3.2 bridge 模式的默认设置

范例：查看bridge模式信息

```
[root@ubuntu1804 ~]#docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id":
"fe08e6d23c4c9de00bdab479446f136c09537a1551aa62ff2c95f8cfcabd6357",
        "Created": "2020-01-31T16:11:32.718471804+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "cdb5173003f52033c7c8183994cf763d2a64ff39c431431402fd8dedf4727393":
{
                "Name": "server1",
                "EndpointID":
"6977fb6f74b75014513c34296f1e23ff0197f81f3209bbf7fcd39ba8e9f54c0d",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
```

```
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
[root@ubuntu1804 ~]#
```

范例: 宿主机的网络状态

```
#安装docker后,默认启用ip_forward
[root@ubuntu1804 ~]#cat /proc/sys/net/ipv4/ip_forward
1
[root@ubuntu1804 ~]#iptables -vnL -t nat
Chain PREROUTING (policy ACCEPT 245 packets, 29850 bytes)
 pkts bytes target     prot opt in     out     source               destination

   11   636 DOCKER     all  --  *      *       0.0.0.0/0            0.0.0.0/0
         ADDRTYPE match dst-type LOCAL

Chain INPUT (policy ACCEPT 103 packets, 20705 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain OUTPUT (policy ACCEPT 144 packets, 10324 bytes)
 pkts bytes target     prot opt in     out     source               destination

    0     0 DOCKER     all  --  *      *       0.0.0.0/0           !127.0.0.0/8
         ADDRTYPE match dst-type LOCAL

Chain POSTROUTING (policy ACCEPT 158 packets, 11500 bytes)
 pkts bytes target     prot opt in     out     source               destination

  125  7831 MASQUERADE  all  --  *      !docker0  172.17.0.0/16        0.0.0.0/0


Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination

    2   168 RETURN     all  --  docker0 *       0.0.0.0/0            0.0.0.0/0
```

范例: 通过宿主机的物理网卡利用SNAT访问外部网络

```
#在另一台主机上建立httpd服务器
[root@centos7 ~]#systemctl is-active httpd
active

#启动容器，默认是bridge网络模式
[root@ubuntu1804 ~]#docker run -it  --rm  alpine:3.11 sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
```

```
       valid_lft forever preferred_lft forever
166: eth0@if167: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
#可能访问其它宿主机
/ # ping 10.0.0.7
PING 10.0.0.7 (10.0.0.7): 56 data bytes
64 bytes from 10.0.0.7: seq=0 ttl=63 time=0.764 ms
64 bytes from 10.0.0.7: seq=1 ttl=63 time=1.147 ms
^C
--- 10.0.0.7 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.764/0.955/1.147 ms
/ # ping www.baidu.com
PING www.baidu.com (61.135.169.125): 56 data bytes
64 bytes from 61.135.169.125: seq=0 ttl=127 time=5.182 ms
^C
--- www.baidu.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 5.182/5.182/5.182 ms

/ # traceroute 10.0.0.7
traceroute to 10.0.0.7 (10.0.0.7), 30 hops max, 46 byte packets
 1  172.17.0.1 (172.17.0.1)  0.008 ms  0.008 ms  0.007 ms
 2  10.0.0.7 (10.0.0.7)  0.255 ms  0.510 ms  0.798 ms
/ # wget -qO - 10.0.0.7
Website on 10.0.0.7
/ # route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.17.0.1      0.0.0.0         UG    0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0

[root@centos7 ~]#curl 127.0.0.1
Website on 10.0.0.7

[root@centos7 ~]#tail  /var/log/httpd/access_log
127.0.0.1 - - [01/Feb/2020:19:31:16 +0800] "GET / HTTP/1.1" 200 20 "-"
"curl/7.29.0"
10.0.0.100 - - [01/Feb/2020:19:31:21 +0800] "GET / HTTP/1.1" 200 20 "-" "Wget"
```

4.2.3.3 修改默认的 bridge 模式网络配置

范例: 修改bridge模式默认的网段方法1

```
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
```

```
    link/ether 00:0c:29:6b:54:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6b:54d3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:e0:ef:72:05 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:e0ff:feef:7205/64 scope link
       valid_lft forever preferred_lft forever

[root@ubuntu1804 ~]#docker run -it --rm  alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
4: eth0@if5: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ #exit

#修改桥接地址
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
 --bip=10.100.0.1/24
[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:6b:54:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.101/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6b:54d3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:e0:ef:72:05 brd ff:ff:ff:ff:ff:ff
    inet 10.100.0.1/24 brd 10.100.0.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:e0ff:feef:7205/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#docker run -it --rm  alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
179: eth0@if180: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:0a:64:00:02 brd ff:ff:ff:ff:ff:ff
    inet 10.100.0.2/24 brd 10.100.0.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # exit
[root@ubuntu1804 ~]#docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id":
"ace11446c233d1fef534d9c734bf3ab5524afdbe76934a1a0e64803d03d54f98",
        "Created": "2020-02-02T13:23:58.037630754+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "10.100.0.0/24",
                    "Gateway": "10.100.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
[root@ubuntu1804 ~]#
```

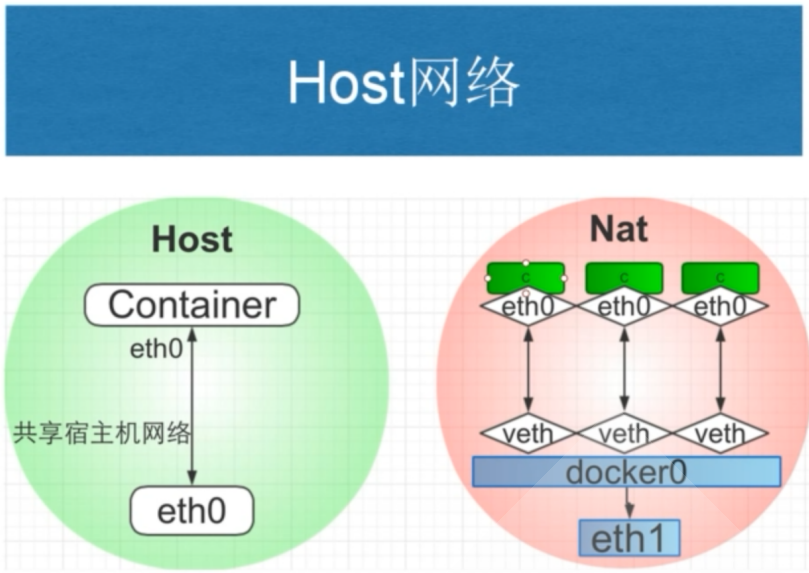
范例：修改bridge网络配置方法2

```
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
{
  "hosts": ["tcp://0.0.0.0:2375", "fd://"],
  "bip": "192.168.100.100/24",          #分配docker0网卡的IP,24是容器IP的netmask
  "fixed-cidr": "192.168.100.128/26", #分配容器IP范围,26不是容器IP的子网掩码,只表示地址
范围
```

```
  "fixed-cidr-v6": "2001:db8::/64",
  "mtu": 1500,
  "default-gateway": "192.168.100.200",  #网关必须和bip在同一个网段
  "default-gateway-v6": "2001:db8:abcd::89",
  "dns": [ "1.1.1.1", "8.8.8.8"]

}
[root@ubuntu1804 ~]#systemctl restart docker

[root@ubuntu1804 ~]#ip a show docker0
3: docker0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
group default
    link/ether 02:42:23:be:97:75 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.100/24 brd 192.168.100.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:23ff:febe:9775/64 scope link
       valid_lft forever preferred_lft forever


[root@ubuntu1804 ~]#docker run -it --name b1 busybox
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
36: eth0@if37: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 02:42:c0:a8:64:80 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.128/24 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # cat /etc/resolv.conf
search magedu.com magedu.org
nameserver 1.1.1.1
nameserver 8.8.8.8
/ # route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.100.200 0.0.0.0         UG    0      0        0 eth0
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0


[root@ubuntu1804 ~]#docker network inspect bridge
[
    {
        "Name": "bridge",
        "Id":
"381bc2df514b0901e2a7570708aa93a3af05f298f27d4d077b52a8b324fad66c",
        "Created": "2020-07-27T21:58:31.419420569+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "192.168.100.0/24",
                    "IPRange": "192.168.100.128/26",
                    "Gateway": "192.168.100.100",
```

```
                    "AuxiliaryAddresses": {
                        "DefaultGatewayIPv4": "192.168.100.200"
                    }
                },
                {
                    "Subnet": "2001:db8::/64",
                    "AuxiliaryAddresses": {
                        "DefaultGatewayIPv6": "2001:db8:abcd::89"
                    }
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "2f16c9f5efc1eefe766f6ae6ba7fcfa3434e8f4876ecdcf48c3343acd9e45b2d":
{
                "Name": "b1",
                "EndpointID":
"0a0fdf3d786310dca53e04f0734b9f0eeaa79aac147c7a3c69ac8d04444570f3",
                "MacAddress": "02:42:c0:a8:64:80",
                "IPv4Address": "192.168.100.128/24",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```


4.2.4 Host 模式



如果指定host模式启动的容器，那么新创建的容器不会创建自己的虚拟网卡，而是直接使用宿主机的网卡和IP地址，因此在容器里面查看到的IP信息就是宿主机的信息，访问容器的时候直接使用宿主机IP+容器端口即可，不过容器内除网络以外的其它资源，如：文件系统、系统进程等仍然和宿主机保持隔离

此模式由于直接使用宿主机的网络无需转换，网络性能最高，但是各容器内使用的端口不能相同，适用于运行容器端口比较固定的业务

Host 网络模式特点:

- 使用参数 --network host 指定
- 共享宿主机网络
- 网络性能无损耗
- 网络故障排除相对简单
- 各容器网络无隔离
- 网络资源无法分别统计
- 端口管理困难：容易产生端口冲突
- 不支持端口映射

范例:

```
#查看宿主机的网络设置
[root@ubuntu1804 ~]#ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:2ff:fe7f:a8c6  prefixlen 64  scopeid 0x20<link>
        ether 02:42:02:7f:a8:c6  txqueuelen 0  (Ethernet)
        RX packets 63072  bytes 152573158 (152.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 56611  bytes 310696704 (310.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.100  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::20c:29ff:fe34:df91  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:34:df:91  txqueuelen 1000  (Ethernet)
        RX packets 2029082  bytes 1200597401 (1.2 GB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7272209  bytes 11576969391 (11.5 GB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

```
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 3533  bytes 320128 (320.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3533  bytes 320128 (320.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@ubuntu1804 ~]#route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0

#打开容器前确认宿主机的80/tcp端口没有打开
[root@ubuntu1804 ~]#ss -ntl|grep :80

#创建host模式的容器
[root@ubuntu1804 ~]#docker run -d --network host --name web1 nginx-centos7-
base:1.6.1
41fb5b8e41db26e63579a424df643d1f02e272dc75e76c11f4e313a443187ed1

#创建容器后，宿主机的80/tcp端口打开
[root@ubuntu1804 ~]#ss -ntlp|grep :80
LISTEN   0         128                 0.0.0.0:80               0.0.0.0:*
users:(("nginx",pid=43762,fd=6),("nginx",pid=43737,fd=6))

#进入容器
[root@ubuntu1804 ~]#docker exec -it web1 bash

#进入容器后仍显示宿主机的主机名提示符信息
[root@ubuntu1804 /]# hostname
ubuntu1804.magedu.org
[root@ubuntu1804 /]# ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        inet6 fe80::42:2ff:fe7f:a8c6  prefixlen 64  scopeid 0x20<link>
        ether 02:42:02:7f:a8:c6  txqueuelen 0  (Ethernet)
        RX packets 63072  bytes 152573158 (145.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 56611  bytes 310696704 (296.3 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.100  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::20c:29ff:fe34:df91  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:34:df:91  txqueuelen 1000  (Ethernet)
        RX packets 2028984  bytes 1200589212 (1.1 GiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7272137  bytes 11576960933 (10.7 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
```

```
        RX packets 3533  bytes 320128 (312.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 3533  bytes 320128 (312.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
[root@ubuntu1804 /]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0

#从容器访问远程主机
[root@ubuntu1804 /]# curl 10.0.0.7
Website on 10.0.0.7


#查看远程主机的访问日志
[root@centos7 ~]#tail -n1 /var/log/httpd/access_log
10.0.0.100 - - [01/Feb/2020:19:58:06 +0800] "GET / HTTP/1.1" 200 20 "-"
"curl/7.29.0"

#远程主机可以访问容器的web服务
[root@centos7 ~]#curl 10.0.0.100/app/
Test Page in app
```

范例：host模式下端口映射无法实现

```
[root@ubuntu1804 ~]#ss -ntl|grep :81
[root@ubuntu1804 ~]#docker run -d --network host --name web2 -p 81:80  nginx-
centos7-base:1.6.1
WARNING: Published ports are discarded when using host network mode
6b6a910d79d94b188f719bc6ad00c274acd76a4a2929212157cd49b5219d44ae
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                       COMMAND                  CREATED
             STATUS                      PORTS               NAMES
6b6a910d79d9        nginx-centos7-base:1.6.1    "/apps/nginx/sbin/ng…"   6
seconds ago         Exited (1) 2 seconds ago                        web2
b27c0fd28b40        nginx-centos7-base:1.6.1    "/apps/nginx/sbin/ng…"   About a
minute ago   Up About a minute                               web1
```

范例：对比前面host模式的容器和bridge模式的端口映射

```
[root@ubuntu1804 ~]#docker port web1
[root@ubuntu1804 ~]#docker port web2
[root@ubuntu1804 ~]#docker run -d --network bridge -p 8001:80 --name web3 nginx-
centos7-base:1.6.1
4095372b9a561704eac98ccef8041a80a2cdc2aa7b57d2798dec1a8dcb00c377
[root@ubuntu1804 ~]#docker port web3
80/tcp -> 0.0.0.0:8001
```

4.2.5 none 模式

在使用none 模式后，Docker 容器不会进行任何网络配置，没有网卡、没有IP也没有路由，因此默认无法与外界通信，需要手动添加网卡配置IP等，所以极少使用

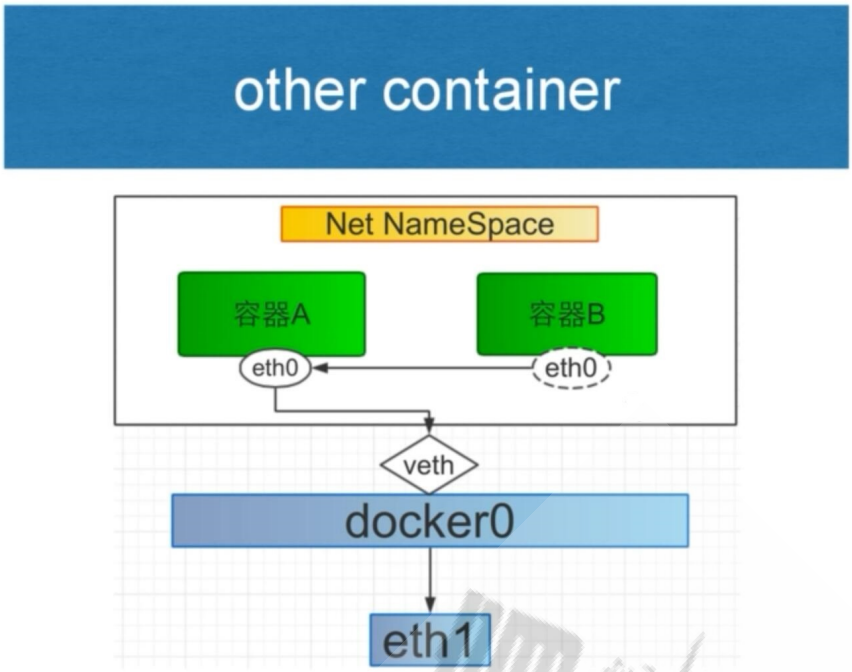
none模式特点

- 使用参数 --network none 指定
- 默认无网络功能，无法和外部通信

范例: 启动none模式的容器

```
[root@ubuntu1804 ~]#docker run -d --network none -p 8001:80 --name web1-none
nginx-centos7-base:1.6.1
5207dcbd0aeea88548819267d3751135e337035475cf3cd63a5e1be6599c0208
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                         COMMAND                  CREATED
             STATUS              PORTS               NAMES
5207dcbd0aee        nginx-centos7-base:1.6.1   "/apps/nginx/sbin/ng…"   About a
minute ago   Up About a minute                       web1-none

[root@ubuntu1804 ~]#docker port web1-none
[root@ubuntu1804 ~]#docker exec -it web1-none bash
[root@5207dcbd0aee /]# ifconfig -a
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@5207dcbd0aee /]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
[root@5207dcbd0aee /]# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN
[root@5207dcbd0aee /]# ping www.baidu.com
ping: www.baidu.com: Name or service not known
[root@5207dcbd0aee /]# ping 172.17.0.1
connect: Network is unreachable
[root@5207dcbd0aee /]#
```

4.2.6 Container 模式



使用此模式创建的容器需指定和一个已经存在的容器共享一个网络，而不是和宿主机共享网，新创建的容器不会创建自己的网卡也不会配置自己的IP，而是和一个被指定的已经存在的容器共享IP和端口范围，因此这个容器的端口不能和被指定容器的端口冲突，除了网络之外的文件系统、进程信息等仍然保持相互隔离，两个容器的进程可以通过lo网卡进行通信

Container 模式特点

- 使用参数 --network container:名称或ID 指定
- 与宿主机网络空间隔离
- 空器间共享网络空间
- 适合频繁的容器间的网络通信
- 直接使用对方的网络，较少使用

范例:

```
#创建第一个容器
[root@ubuntu1804 ~]#docker run -it --name server1 -p 80:80 alpine:3.11 sh
/ # ifconfig
eth0      Link encap:Ethernet  HWaddr 02:42:AC:11:00:02
          inet addr:172.17.0.2  Bcast:172.17.255.255  Mask:255.255.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:766 (766.0 B)  TX bytes:0 (0.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ # netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
```

```
/ #


#在另一个终端执行下面操作
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS                NAMES
4d342fac169f        alpine:3.11         "sh"                29 seconds ago
Up 28 seconds       0.0.0.0:80->80/tcp   server1
[root@ubuntu1804 ~]#docker port server1
80/tcp -> 0.0.0.0:80

#无法访问web服务
[root@ubuntu1804 ~]#curl 127.0.0.1/app/
curl: (52) Empty reply from server

#创建第二个容器，基于第一个容器的container的网络模式
[root@ubuntu1804 ~]#docker run -d --name server2 --network container:server1
nginx-centos7-base:1.6.1
7db90f38590ade11e1c833a8b2175810c71b3f222753c5177bb8b05952f08a7b
#可以访问web服务
[root@ubuntu1804 ~]#curl 127.0.0.1/app/
Test Page in app

[root@ubuntu1804 ~]#docker exec -it server2 bash
#和第一个容器共享相同的网络
[root@4d342fac169f /]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.2  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:02  txqueuelen 0  (Ethernet)
        RX packets 29  bytes 2231 (2.1 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 12  bytes 1366 (1.3 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 10  bytes 860 (860.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 860 (860.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@4d342fac169f /]# netstat -ntl
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN

#可访问外网
[root@4d342fac169f /]# ping www.baidu.com
PING www.a.shifen.com (61.135.169.121) 56(84) bytes of data.
64 bytes from 61.135.169.121 (61.135.169.121): icmp_seq=1 ttl=127 time=3.99 ms
64 bytes from 61.135.169.121 (61.135.169.121): icmp_seq=2 ttl=127 time=5.03 ms
^C
--- www.a.shifen.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 3.999/4.514/5.030/0.519 ms
[root@4d342fac169f /]#
```

范例: 第一个容器使用host网络模式,第二个容器与之共享网络

```
[root@ubuntu1804 ~]#docker run -d --name c1  --network host  nginx-
centos7.8:v5.0-1.18.0
5a60804f3917d82dfe32db140411cf475f20acce0fe4674d94e4557e1003d8e0
[root@ubuntu1804 ~]#docker run -it  --name c2 --network container:c1
centos7.8:v1.0
[root@ubuntu1804 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:63:8b:ac brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe63:8bac/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:24:86:98:fb brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:24ff:fe86:98fb/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#docker exec -it c1 bash
[root@ubuntu1804 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:63:8b:ac brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe63:8bac/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:24:86:98:fb brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:24ff:fe86:98fb/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 /]#
```

范例:第一个容器使用none网络模式,第二个容器与之共享网络

```
[root@ubuntu1804 ~]#docker run -d --name c1  --network none  nginx-
centos7.8:v5.0-1.18.0
caf5b57299c8359f21f30b8894c5f8496ff39b44ead6a732056000689cb0c91c
[root@ubuntu1804 ~]#docker run -it  --name c2 --network container:c1
centos7.8:v1.0
[root@caf5b57299c8 /]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
[root@caf5b57299c8 /]#
```

4.3.7 自定义网络模式

除了以上的网络模式，也可以自定义网络，使用自定义的网段地址，网关等信息

注意: 自定义网络内的容器可以直接通过容器名进行相互的访问,而无需使用 --link

可以使用自定义网络模式,实现不同集群应用的独立网络管理,而互不影响,而且在网一个网络内,可以直接利用容器名相互访问非常便利

4.3.7.1 自定义网络实现

```
[root@ubuntu1804 ~]#docker network  --help

Usage:  docker network COMMAND

Manage networks

Commands:
  connect     Connect a container to a network
  create      Create a network
  disconnect  Disconnect a container from a network
  inspect     Display detailed information on one or more networks
  ls          List networks
  prune       Remove all unused networks
  rm          Remove one or more networks
```

创建自定义网络:

```
docker network create -d <mode> --subnet <CIDR> --gateway <网关> <自定义网络名称>

#注意mode不支持host和none
```

查看自定义网络信息

```
docker network inspect <自定义网络名称或网络ID>
```

引用自定议网络

```
docker run --network  <自定义网络名称> <镜像名称>
```

删除自定义网络

```
doccker network rm <自定义网络名称或网络ID>
```

4.3.7.2 实战案例: 自定义网络

4.3.7.2.1 创建自定义的网络

```
[root@ubuntu1804 ~]#docker  network create -d bridge --subnet 172.27.0.0/16 --
gateway 172.27.0.1 test-net
c90dee3b7937e007ed31a8d016a9e54c0174d0d26487b154db0aff04d9016d5b
[root@ubuntu1804 ~]#docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
cabde0b33c94        bridge              bridge              local
cb64aa83626c        host                host                local
10619d45dcd4        none                null                local
c90dee3b7937        test-net            bridge              local

[root@ubuntu1804 ~]#docker inspect test-net
[
    {
        "Name": "test-net",
        "Id":
"00ab0f2d29e82d387755e1bea19532dc279fa134a565e496d308ec62f7edf434",
        "Created": "2020-07-22T09:59:09.431393706+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.27.0.0/16",
                    "Gateway": "172.27.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]

[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
```

```
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:34:df:91 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.100/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe34:df91/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:9b:31:73:2b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:9bff:fe31:732b/64 scope link
       valid_lft forever preferred_lft forever
#新添加了一个虚拟网卡
14: br-c90dee3b7937: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN group default
    link/ether 02:42:58:7c:f0:93 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.1/16 brd 172.27.255.255 scope global br-c90dee3b7937
       valid_lft forever preferred_lft forever
    inet6 fe80::42:58ff:fe7c:f093/64 scope link
       valid_lft forever preferred_lft forever

#新加了一个网桥
[root@ubuntu1804 ~]#brctl show
bridge name bridge id       STP enabled interfaces
br-00ab0f2d29e8     8000.024245e647ec   no
docker0     8000.0242cfd26f0a   no

[root@ubuntu1804 ~]#route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
172.17.0.0      0.0.0.0         255.255.0.0     U     0      0        0 docker0
172.27.0.0      0.0.0.0         255.255.0.0     U     0      0        0 br-
c90dee3b7937
```

4.3.7.2.2 利用自定义的网络创建容器

```
[root@ubuntu1804 ~]#docker run -it --rm --network test-net alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
15: eth0@if16: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.2/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # / # route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         172.27.0.1      0.0.0.0         UG    0      0        0 eth0
```

```
172.27.0.0      0.0.0.0         255.255.0.0     U     0      0        0 eth0

/ # cat /etc/resolv.conf
search magedu.com magedu.org
nameserver 127.0.0.11
options ndots:0

/ # ping -c1 www.baidu.com
PING www.baidu.com (111.206.223.172): 56 data bytes
64 bytes from 111.206.223.172: seq=0 ttl=127 time=5.053 ms



#再开一个新终端窗口查看网络
[root@ubuntu1804 ~]#docker inspect test-net
[
    {
        "Name": "test-net",
        "Id":
"00ab0f2d29e82d387755e1bea19532dc279fa134a565e496d308ec62f7edf434",
        "Created": "2020-07-22T09:59:09.431393706+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.27.0.0/16",
                    "Gateway": "172.27.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        #出现此网络中容器的网络信息
        "Containers": {
            "89e54ed71c111ac7b41a62ce20191707cf53a3a234ba3e25ac11c1a4a6bed7ef":
{
                "Name": "frosty_ellis",
                "EndpointID":
"cf72bf192df73a8b290d8b18dd8507fef64a1f9480d4d65f74c23258d20dbafb",
                "MacAddress": "02:42:ac:1b:00:02",
                "IPv4Address": "172.27.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

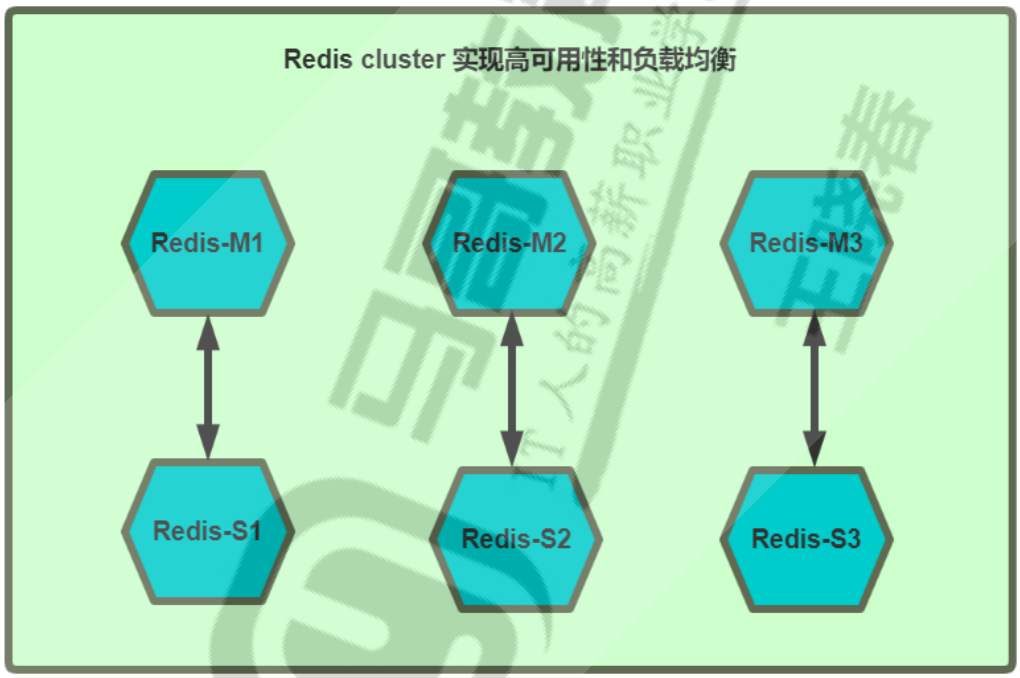
4.3.7.3 实战案例: 自定义网络中的容器之间通信

```
[root@ubuntu1804 ~]#docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
c2f770f19400        bridge              bridge              local
220d4008a6a0        host                host                local
adb6f338ff6d        none                null                local
00ab0f2d29e8        test-net            bridge              local

[root@ubuntu1804 ~]#docker run -it --rm --network test-net --name test1  alpine
sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.2/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.27.0.2  c3446876a38b

#等后面步骤中容器test2创建好可以再访问
/ # ping -c1 test2
PING test2 (172.27.0.3): 56 data bytes
64 bytes from 172.27.0.3: seq=0 ttl=64 time=0.072 ms

--- test2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.072/0.072/0.072 ms



[root@ubuntu1804 ~]#docker run -it --rm --network test-net --name test2 alpine
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
25: eth0@if26: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.3/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
```

```
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.27.0.3  305fd14a4b70

/ # ping -c1 test1
PING test1 (172.27.0.2): 56 data bytes
64 bytes from 172.27.0.2: seq=0 ttl=64 time=0.074 ms

--- test1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.074/0.074/0.074 ms
```

结论: 自定义网络中的容器之间可以直接利用容器名进行通信

4.3.7.4 实战案例: 利用自定义网络实现 Redis Cluster



4.3.7.4.1 创建自定义网络

```
[root@ubuntu1804 ~]#docker network create net-redis --subnet 172.18.0.0/16
09b9dded99787835dccc029e16fa2782292d22c3e258f60a1db15d44e7a3bd93
[root@ubuntu1804 ~]#docker inspect net-redis
[
    {
        "Name": "net-redis",
        "Id":
"09b9dded99787835dccc029e16fa2782292d22c3e258f60a1db15d44e7a3bd93",
        "Created": "2020-07-22T14:16:43.295465692+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
```

```
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

4.3.7.4.2 创建6个redis容器配置

```
# 通过脚本创建六个redis容器配置
[root@ubuntu1804 ~]#for port in {1..6};do
    mkdir -p /data/redis/node-${port}/conf
    cat  >> /data/redis/node-${port}/conf/redis.conf << EOF
port 6379
bind 0.0.0.0
masterauth 123456
requirepass 123456
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
cluster-announce-ip 172.18.0.1${port}
cluster-announce-port 6379
cluster-announce-bus-port 16379
appendonly yes
EOF
done

[root@ubuntu1804 ~]#tree /data/redis/
/data/redis/
├── node-1
│   └── conf
│       └── redis.conf
├── node-2
│   └── conf
│       └── redis.conf
├── node-3
│   └── conf
│       └── redis.conf
├── node-4
│   └── conf
│       └── redis.conf
├── node-5
│   └── conf
│       └── redis.conf
```

```
└── node-6
    └── conf
        └── redis.conf

12 directories, 6 files
[root@ubuntu1804 ~]#cat /data/redis/node-1/conf/redis.conf
port 6379
bind 0.0.0.0
masterauth 123456
requirepass 123456
cluster-enabled yes
cluster-config-file nodes.conf
cluster-node-timeout 5000
cluster-announce-ip 172.18.0.11
cluster-announce-port 6379
cluster-announce-bus-port 16379
appendonly yes
```

4.3.7.4.3 创建6个 redis 容器

```
# 通过脚本运行六个redis容器
[root@ubuntu1804 ~]#for port in {1..6};do
    docker run -p 637${port}:6379 -p 1667${port}:16379 --name redis-${port} \
    -v /data/redis/node-${port}/data:/data \
    -v /data/redis/node-${port}/conf/redis.conf:/etc/redis/redis.conf \
    -d --net net-redis --ip 172.18.0.1${port} redis:5.0.9-alpine3.11 redis-
server /etc/redis/redis.conf
done


[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                  COMMAND                  CREATED
          STATUS              PORTS
NAMES
2525235efae6        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   23 seconds
ago       Up 21 seconds       0.0.0.0:6376->6379/tcp, 0.0.0.0:16676->16379/tcp
redis-6
bd89dbb445ae        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   24 seconds
ago       Up 22 seconds       0.0.0.0:6375->6379/tcp, 0.0.0.0:16675->16379/tcp
redis-5
51cb6244d34d        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   26 seconds
ago       Up 23 seconds       0.0.0.0:6374->6379/tcp, 0.0.0.0:16674->16379/tcp
redis-4
1a49a47eb72d        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   27 seconds
ago       Up 25 seconds       0.0.0.0:6373->6379/tcp, 0.0.0.0:16673->16379/tcp
redis-3
a03e957680f0        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   28 seconds
ago       Up 26 seconds       0.0.0.0:6372->6379/tcp, 0.0.0.0:16672->16379/tcp
redis-2
b5332c0cba81        redis:5.0.9-alpine3.11   "docker-entrypoint.s…"   31 seconds
ago       Up 28 seconds       0.0.0.0:6371->6379/tcp, 0.0.0.0:16671->16379/tcp
redis-1
```

4.3.7.4.4 创建 redis cluster

```
#连接redis cluster
```

```
[root@ubuntu1804 ~]#docker exec -it redis-1 /bin/sh
/data # redis-cli -a 123456
Warning: Using a password with '-a' or '-u' option on the command line interface
may not be safe.
127.0.0.1:6379> exit
#不支持 { } 扩展
/data # echo {1..10}
{1..10}
/data # echo $-
smi

# 创建集群
/data # redis-cli -a 123456 --cluster create 172.18.0.11:6379 172.18.0.12:6379
172.18.0.13:6379 172.18.0.14:6379 172.18.0.15:6379 172.18.0.16:6379 --cluster-
replicas 1
Warning: Using a password with '-a' or '-u' option on the command line interface
may not be safe.
>>> Performing hash slots allocation on 6 nodes...
Master[0] -> Slots 0 - 5460
Master[1] -> Slots 5461 - 10922
Master[2] -> Slots 10923 - 16383
Adding replica 172.18.0.15:6379 to 172.18.0.11:6379
Adding replica 172.18.0.16:6379 to 172.18.0.12:6379
Adding replica 172.18.0.14:6379 to 172.18.0.13:6379
M: 0f9bd0d24495f826702a030703896f7690bebdee 172.18.0.11:6379
   slots:[0-5460] (5461 slots) master
M: 9b6ab0b8f75516d6acd9d566d0d349f1fdd29540 172.18.0.12:6379
   slots:[5461-10922] (5462 slots) master
M: 599f69b43a3579ec064b1854680c77997c809470 172.18.0.13:6379
   slots:[10923-16383] (5461 slots) master
S: c64dfd1bd6c964a3c6425a28f6ab0f0e1bcea1ba 172.18.0.14:6379
   replicates 599f69b43a3579ec064b1854680c77997c809470
S: 2f69287f52ec7243a0b894491814d2afe28a46d2 172.18.0.15:6379
   replicates 0f9bd0d24495f826702a030703896f7690bebdee
S: 06295ce4884948858cf60243629a595afa461b21 172.18.0.16:6379
   replicates 9b6ab0b8f75516d6acd9d566d0d349f1fdd29540
Can I set the above configuration? (type 'yes' to accept): #输入yes
>>> Nodes configuration updated
>>> Assign a different config epoch to each node
>>> Sending CLUSTER MEET messages to join the cluster
Waiting for the cluster to join
....
>>> Performing Cluster Check (using node 172.18.0.11:6379)
M: 0f9bd0d24495f826702a030703896f7690bebdee 172.18.0.11:6379
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
S: 2f69287f52ec7243a0b894491814d2afe28a46d2 172.18.0.15:6379
   slots: (0 slots) slave
   replicates 0f9bd0d24495f826702a030703896f7690bebdee
S: c64dfd1bd6c964a3c6425a28f6ab0f0e1bcea1ba 172.18.0.14:6379
   slots: (0 slots) slave
   replicates 599f69b43a3579ec064b1854680c77997c809470
M: 9b6ab0b8f75516d6acd9d566d0d349f1fdd29540 172.18.0.12:6379
   slots:[5461-10922] (5462 slots) master
   1 additional replica(s)
M: 599f69b43a3579ec064b1854680c77997c809470 172.18.0.13:6379
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
```

```
S: 06295ce4884948858cf60243629a595afa461b21 172.18.0.16:6379
   slots: (0 slots) slave
   replicates 9b6ab0b8f75516d6acd9d566d0d349f1fdd29540
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.
```

4.3.7.4.5 测试访问 redis cluster

```
#连接redis cluster
/data # redis-cli  -a 123456 -c
Warning: Using a password with '-a' or '-u' option on the command line interface
may not be safe.
127.0.0.1:6379>  cluster info
cluster_state:ok
cluster_slots_assigned:16384
cluster_slots_ok:16384
cluster_slots_pfail:0
cluster_slots_fail:0
cluster_known_nodes:6
cluster_size:3
cluster_current_epoch:6
cluster_my_epoch:1
cluster_stats_messages_ping_sent:267
cluster_stats_messages_pong_sent:278
cluster_stats_messages_sent:545
cluster_stats_messages_ping_received:273
cluster_stats_messages_pong_received:267
cluster_stats_messages_meet_received:5
cluster_stats_messages_received:545

127.0.0.1:6379> cluster nodes
#看到172.18.0.{11,12,13}为master,172.18.0.{14,15,16}为slave
#以下为master/slave关系
#172.18.0.11<--->172.18.0.15
#172.18.0.12<--->172.18.0.16
#172.18.0.13<--->172.18.0.14
2f69287f52ec7243a0b894491814d2afe28a46d2 172.18.0.15:6379@16379 slave
0f9bd0d24495f826702a030703896f7690bebdee 0 1595404269581 5 connected
0f9bd0d24495f826702a030703896f7690bebdee 172.18.0.11:6379@16379 myself,master -
0 1595404269000 1 connected 0-5460
c64dfd1bd6c964a3c6425a28f6ab0f0e1bcea1ba 172.18.0.14:6379@16379 slave
599f69b43a3579ec064b1854680c77997c809470 0 1595404268000 4 connected
9b6ab0b8f75516d6acd9d566d0d349f1fdd29540 172.18.0.12:6379@16379 master - 0
1595404268976 2 connected 5461-10922
599f69b43a3579ec064b1854680c77997c809470 172.18.0.13:6379@16379 master - 0
1595404269481 3 connected 10923-16383
06295ce4884948858cf60243629a595afa461b21 172.18.0.16:6379@16379 slave
9b6ab0b8f75516d6acd9d566d0d349f1fdd29540 0 1595404268000 6 connected

#添加key到redis-2上
127.0.0.1:6379> set name wang
-> Redirected to slot [5798] located at 172.18.0.12:6379
OK
#添加key到redis-1上
```

```
172.18.0.12:6379> set title cto
-> Redirected to slot [2217] located at 172.18.0.11:6379
OK

172.18.0.11:6379> get name
-> Redirected to slot [5798] located at 172.18.0.12:6379
"wang"
172.18.0.12:6379> get title
-> Redirected to slot [2217] located at 172.18.0.11:6379
"cto"
```

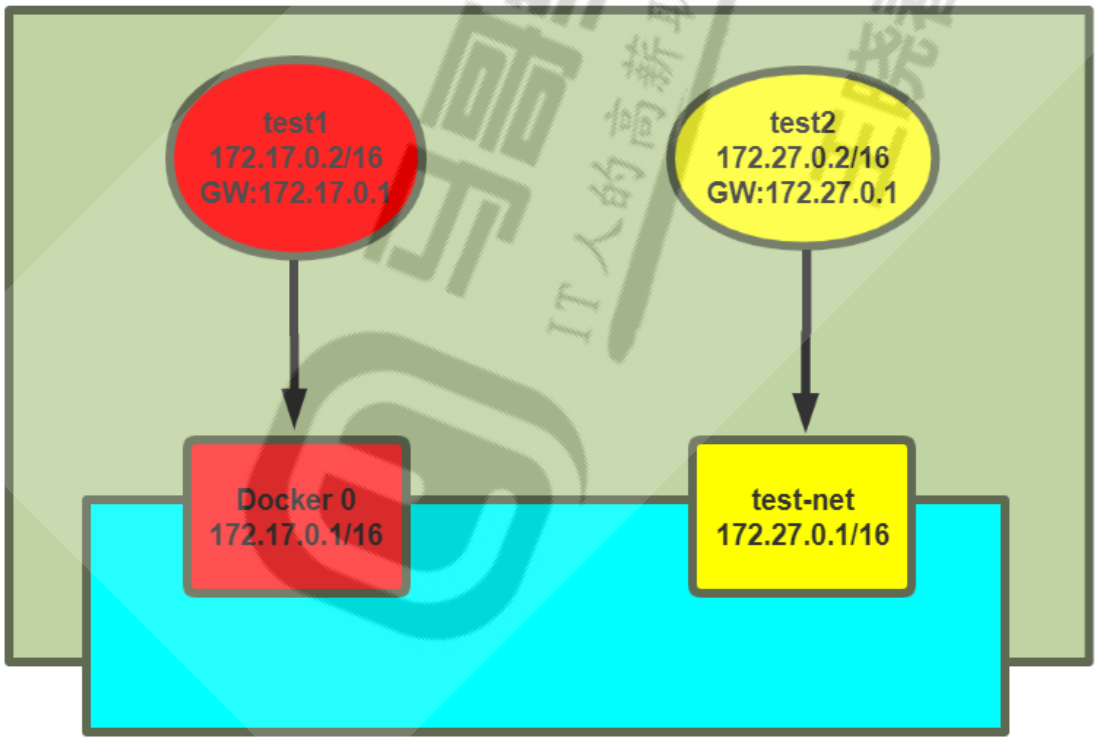
4.3.7.4.5 测试故障实现 redis cluster 高可用性

```
#模拟redis-2故障
[root@ubuntu1804 ~]#docker stop  redis-2
redis-2

#再次查看cluster状态,可以看到redis-2出错
[root@ubuntu1804 ~]#docker exec -it redis-1 /bin/sh
/data # redis-cli -a 123456 --cluster check 127.0.0.1:6379
Warning: Using a password with '-a' or '-u' option on the command line interface
may not be safe.
Could not connect to Redis at 172.18.0.12:6379: Host is unreachable
#查看到 172.18.0.16提升为新的master
172.18.0.16:6379 (06295ce4...) -> 1 keys | 5462 slots | 0 slaves.
172.18.0.13:6379 (599f69b4...) -> 0 keys | 5461 slots | 1 slaves.
172.18.0.15:6379 (2f69287f...) -> 1 keys | 5461 slots | 1 slaves.
[OK] 2 keys in 3 masters.
0.00 keys per slot on average.
>>> Performing Cluster Check (using node 127.0.0.1:6379)
S: 0f9bd0d24495f826702a030703896f7690bebdee 127.0.0.1:6379
   slots: (0 slots) slave
   replicates 2f69287f52ec7243a0b894491814d2afe28a46d2
M: 06295ce4884948858cf60243629a595afa461b21 172.18.0.16:6379
   slots:[5461-10922] (5462 slots) master
M: 599f69b43a3579ec064b1854680c77997c809470 172.18.0.13:6379
   slots:[10923-16383] (5461 slots) master
   1 additional replica(s)
M: 2f69287f52ec7243a0b894491814d2afe28a46d2 172.18.0.15:6379
   slots:[0-5460] (5461 slots) master
   1 additional replica(s)
S: c64dfd1bd6c964a3c6425a28f6ab0f0e1bcea1ba 172.18.0.14:6379
   slots: (0 slots) slave
   replicates 599f69b43a3579ec064b1854680c77997c809470
[OK] All nodes agree about slots configuration.
>>> Check for open slots...
>>> Check slots coverage...
[OK] All 16384 slots covered.


/data # redis-cli  -a 123456 -c
Warning: Using a password with '-a' or '-u' option on the command line interface
may not be safe.
127.0.0.1:6379>  cluster nodes
06295ce4884948858cf60243629a595afa461b21 172.18.0.16:6379@16379 master - 0
1595406573128 8 connected 5461-10922
```

```
599f69b43a3579ec064b1854680c77997c809470 172.18.0.13:6379@16379 master - 0
1595406572623 3 connected 10923-16383
0f9bd0d24495f826702a030703896f7690bebdee 172.18.0.11:6379@16379 myself,slave
2f69287f52ec7243a0b894491814d2afe28a46d2 0 1595406571000 1 connected
2f69287f52ec7243a0b894491814d2afe28a46d2 172.18.0.15:6379@16379 master - 0
1595406571614 7 connected 0-5460
9b6ab0b8f75516d6acd9d566d0d349f1fdd29540 172.18.0.12:6379@16379 master,fail -
1595404533839 1595404532528 2 connected
c64dfd1bd6c964a3c6425a28f6ab0f0e1bcea1ba 172.18.0.14:6379@16379 slave
599f69b43a3579ec064b1854680c77997c809470 0 1595406572118 4 connected
127.0.0.1:6379> get name
-> Redirected to slot [5798] located at 172.18.0.16:6379
"wang"
172.18.0.16:6379> get title
-> Redirected to slot [2217] located at 172.18.0.15:6379
"cto"
```

4.3.8 同一个宿主机之间不同网络的容器通信

开两个容器，一个使用自定义网络容器，一个使用默认brideg网络的容器,默认因iptables规则导致无法通信




```
[root@ubuntu1804 ~]#docker run -it --rm  --name test1 alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

```
/ # ping 172.27.0.2     #无法ping通自定义网络容器
PING 172.27.0.2 (172.27.0.2): 56 data bytes

[root@ubuntu1804 ~]#docker run -it --rm --network test-net --name test2 alpine
sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
21: eth0@if22: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.2/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping 172.17.0.2     #无法ping 通默认的网络容器
PING 172.27.0.2 (172.17.0.2): 56 data bytes
```

4.3.8.1 实战案例 1: 修改iptables实现同一宿主机上的不同网络的容器间通信

```
#确认开启ip_forward
[root@ubuntu1804 ~]#cat /proc/sys/net/ipv4/ip_forward
1

#默认网络和自定义网络是两个不同的网桥
[root@ubuntu1804 ~]#brctl  show
bridge name bridge id       STP enabled interfaces
br-c90dee3b7937     8000.0242587cf093   no      veth984a5b4
docker0     8000.02429b31732b   no      veth1a20128

[root@ubuntu1804 ~]#iptables -vnL
Chain INPUT (policy ACCEPT 1241 packets, 87490 bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

  859 72156 DOCKER-USER  all  --  *      *       0.0.0.0/0            0.0.0.0/0

  859 72156 DOCKER-ISOLATION-STAGE-1  all  --  *      *       0.0.0.0/0
   0.0.0.0/0
    0     0 ACCEPT     all  --  *      br-c90dee3b7937  0.0.0.0/0
 0.0.0.0/0           ctstate RELATED,ESTABLISHED
    0     0 DOCKER     all  --  *      br-c90dee3b7937  0.0.0.0/0
 0.0.0.0/0
    0     0 ACCEPT     all  --  br-c90dee3b7937 !br-c90dee3b7937  0.0.0.0/0
       0.0.0.0/0
    0     0 ACCEPT     all  --  br-c90dee3b7937 br-c90dee3b7937  0.0.0.0/0
     0.0.0.0/0
    0     0 ACCEPT     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0
         ctstate RELATED,ESTABLISHED
    0     0 DOCKER     all  --  *      docker0  0.0.0.0/0            0.0.0.0/0

    0     0 ACCEPT     all  --  docker0 !docker0  0.0.0.0/0            0.0.0.0/0
```

```
    0     0 ACCEPT     all  --  docker0 docker0  0.0.0.0/0            0.0.0.0/0


Chain OUTPUT (policy ACCEPT 1456 packets, 209K bytes)
 pkts bytes target     prot opt in     out     source               destination


Chain DOCKER (2 references)
 pkts bytes target     prot opt in     out     source               destination


Chain DOCKER-ISOLATION-STAGE-1 (1 references)
 pkts bytes target     prot opt in     out     source               destination

  289 24276 DOCKER-ISOLATION-STAGE-2  all  --  br-c90dee3b7937 !br-c90dee3b7937
 0.0.0.0/0            0.0.0.0/0
  570 47880 DOCKER-ISOLATION-STAGE-2  all  --  docker0 !docker0  0.0.0.0/0
       0.0.0.0/0
    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0


Chain DOCKER-ISOLATION-STAGE-2 (2 references)
 pkts bytes target     prot opt in     out     source               destination

  570 47880 DROP       all  --  *      br-c90dee3b7937  0.0.0.0/0
 0.0.0.0/0
  289 24276 DROP       all  --  *      docker0  0.0.0.0/0            0.0.0.0/0

    0     0 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0


Chain DOCKER-USER (1 references)
 pkts bytes target     prot opt in     out     source               destination

  859 72156 RETURN     all  --  *      *       0.0.0.0/0            0.0.0.0/0

[root@ubuntu1804 ~]#

[root@ubuntu1804 ~]#iptables-save
# Generated by iptables-save v1.6.1 on Sun Feb  2 14:33:19 2020
*filter
:INPUT ACCEPT [1283:90246]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [1489:217126]
:DOCKER - [0:0]
:DOCKER-ISOLATION-STAGE-1 - [0:0]
:DOCKER-ISOLATION-STAGE-2 - [0:0]
:DOCKER-USER - [0:0]
-A FORWARD -j DOCKER-USER
-A FORWARD -j DOCKER-ISOLATION-STAGE-1
-A FORWARD -o br-c90dee3b7937 -m conntrack --ctstate RELATED,ESTABLISHED -j
ACCEPT
-A FORWARD -o br-c90dee3b7937 -j DOCKER
-A FORWARD -i br-c90dee3b7937 ! -o br-c90dee3b7937 -j ACCEPT
-A FORWARD -i br-c90dee3b7937 -o br-c90dee3b7937 -j ACCEPT
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
```

```
-A FORWARD -i docker0 -o docker0 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-1 -i br-c90dee3b7937 ! -o br-c90dee3b7937 -j DOCKER-
ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -i docker0 ! -o docker0 -j DOCKER-ISOLATION-STAGE-2
-A DOCKER-ISOLATION-STAGE-1 -j RETURN
-A DOCKER-ISOLATION-STAGE-2 -o br-c90dee3b7937 -j DROP #注意此行规则
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j DROP   #注意此行规则
-A DOCKER-ISOLATION-STAGE-2 -j RETURN
-A DOCKER-USER -j RETURN
COMMIT
# Completed on Sun Feb  2 14:33:19 2020
# Generated by iptables-save v1.6.1 on Sun Feb  2 14:33:19 2020
*nat
:PREROUTING ACCEPT [887:75032]
:INPUT ACCEPT [6:1028]
:OUTPUT ACCEPT [19:1444]
:POSTROUTING ACCEPT [19:1444]
:DOCKER - [0:0]
-A PREROUTING -m addrtype --dst-type LOCAL -j DOCKER
-A OUTPUT ! -d 127.0.0.0/8 -m addrtype --dst-type LOCAL -j DOCKER
-A POSTROUTING -s 172.27.0.0/16 ! -o br-c90dee3b7937 -j MASQUERADE
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
-A DOCKER -i br-c90dee3b7937 -j RETURN
-A DOCKER -i docker0 -j RETURN
COMMIT
# Completed on Sun Feb  2 14:33:19 2020

[root@ubuntu1804 ~]#iptables-save > iptables.rule
[root@ubuntu1804 ~]#vim iptables.rule
#修改下面两行的规则
-A DOCKER-ISOLATION-STAGE-2 -o br-c90dee3b7937 -j ACCEPT
-A DOCKER-ISOLATION-STAGE-2 -o docker0 -j ACCEPT
#或者执行下面命令
[root@ubuntu1804 ~]#iptables -I DOCKER-ISOLATION-STAGE-2  -j ACCEPT

[root@ubuntu1804 ~]#iptables-restore < iptables.rule

#再次两个容器之间可以相互通信
/ # ping 172.27.0.2
PING 172.27.0.2 (172.27.0.2): 56 data bytes
64 bytes from 172.27.0.2: seq=896 ttl=63 time=0.502 ms
64 bytes from 172.27.0.2: seq=897 ttl=63 time=0.467 ms
64 bytes from 172.27.0.2: seq=898 ttl=63 time=0.227 ms

/ # ping 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=63 time=0.163 ms
64 bytes from 172.17.0.2: seq=1 ttl=63 time=0.232 ms
```

4.3.8.2 实战案例 2: 通过解决docker network connect 实现同一个宿主机不同网络的容器间通信

可以使用docker netowrk connect命令实现同一个宿主机不同网络的容器间相互通信

```
#将CONTAINER连入指定的NETWORK中,使此CONTAINER可以与NETWORK中的其它容器进行通信
docker network connect [OPTIONS] NETWORK CONTAINER
```

```
Connect a container to a network
Options:
      --alias strings           Add network-scoped alias for the container
      --driver-opt strings      driver options for the network
      --ip string               IPv4 address (e.g., 172.30.100.104)
      --ip6 string              IPv6 address (e.g., 2001:db8::33)
      --link list               Add link to another container
      --link-local-ip strings   Add a link-local address for the container

#将CONTAINER与指定的NETWORK断开连接,使此CONTAINER可以与CONTAINER中的其它容器进行无法通信
docker network disconnect [OPTIONS] NETWORK CONTAINER

Disconnect a container from a network

Options:
  -f, --force   Force the container to disconnect from a network
```

4.3.8.2.1 上面案例中test1和test2的容器间默认无法通信

```
#每个网络中有属于此网络的容器信息
[root@ubuntu1804 ~]#docker network  inspect bridge
[
    {
        "Name": "bridge",
        "Id":
"c2f770f19400aa482054a92f2ff6ce54cae2ed45a15c7d98e0959c64dfefd58d",
        "Created": "2020-07-22T09:23:20.265208248+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9bc707b1a810c4bab39a4c0ed3ff5867cc45b21fe8ae6737f2a9d0163ed2c7a9":
{
                "Name": "test1",
                "EndpointID":
"475bba6925c426158b3c523e07b6773c884d404d82e6c19d5e4a41f54f8856c2",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
```

```
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]

#每个网络中有属于此网络的容器信息
[root@ubuntu1804 ~]#docker network  inspect test-net
[
    {
        "Name": "test-net",
        "Id":
"00ab0f2d29e82d387755e1bea19532dc279fa134a565e496d308ec62f7edf434",
        "Created": "2020-07-22T09:59:09.431393706+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.27.0.0/16",
                    "Gateway": "172.27.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "c3446876a38b3d7e70ca35429051dea7373643a95689d22f252faedc31f3c427":
{
                "Name": "test2",
                "EndpointID":
"13fb11baeca7e90abdc9183334315e95df4a55367d3add1472d741a556cb662c",
                "MacAddress": "02:42:ac:1b:00:02",
                "IPv4Address": "172.27.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

4.3.8.2.2 让默认网络中容器test1可以连通自定义网络test-net的容器test2

```
[root@ubuntu1804 ~]#docker network connect test-net test1
[root@ubuntu1804 ~]#docker network  inspect test-net
[
    {
        "Name": "test-net",
        "Id":
"00ab0f2d29e82d387755e1bea19532dc279fa134a565e496d308ec62f7edf434",
        "Created": "2020-07-22T09:59:09.431393706+08:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.27.0.0/16",
                    "Gateway": "172.27.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9bc707b1a810c4bab39a4c0ed3ff5867cc45b21fe8ae6737f2a9d0163ed2c7a9":
{
                "Name": "test1",
                "EndpointID":
"600891a1f0727f0fddcb9c123540d02963a30a54d011554e0dfd1c108ecabdd2",
                "MacAddress": "02:42:ac:1b:00:03",
                "IPv4Address": "172.27.0.3/16",
                "IPv6Address": ""
            },
            "c3446876a38b3d7e70ca35429051dea7373643a95689d22f252faedc31f3c427":
{
                "Name": "test2",
                "EndpointID":
"13fb11baeca7e90abdc9183334315e95df4a55367d3add1472d741a556cb662c",
                "MacAddress": "02:42:ac:1b:00:02",
                "IPv4Address": "172.27.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
```

```
]

#在test1容器中可以看到新添加了一个网卡,并且分配了test-net网络的IP信息
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
27: eth0@if28: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
29: eth1@if30: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.3/16 brd 172.27.255.255 scope global eth1
       valid_lft forever preferred_lft forever
#test1可以连接test2容器
/ # ping -c1 172.27.0.2
PING 172.27.0.2 (172.27.0.2): 56 data bytes
64 bytes from 172.27.0.2: seq=0 ttl=64 time=0.100 ms

--- 172.27.0.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.100/0.100/0.100 ms


#在test2容器中没有变化,仍然无法连接test1
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.2/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping -c1 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
^C
--- 172.17.0.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

4.3.8.2.3 让自定义网络中容器test2可以连通默认网络的容器test1

```
#将自定义网络中的容器test2也加入到默认网络中,使之和默认网络中的容器test1通信
[root@ubuntu1804 ~]#docker network connect bridge test2
[root@ubuntu1804 ~]#docker network  inspect bridge
[
    {
        "Name": "bridge",
        "Id":
"c2f770f19400aa482054a92f2ff6ce54cae2ed45a15c7d98e0959c64dfefd58d",
        "Created": "2020-07-22T09:23:20.265208248+08:00",
        "Scope": "local",
```

```
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "9bc707b1a810c4bab39a4c0ed3ff5867cc45b21fe8ae6737f2a9d0163ed2c7a9":
{
                "Name": "test1",
                "EndpointID":
"475bba6925c426158b3c523e07b6773c884d404d82e6c19d5e4a41f54f8856c2",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            },
            "c3446876a38b3d7e70ca35429051dea7373643a95689d22f252faedc31f3c427":
{
                "Name": "test2",
                "EndpointID":
"a049010b37dd5a40c1ff8e8d0b327b70727316dd86b2c69c05231bfd6c985af6",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```


```
#确认自定义网络的容器test2中添加了新网卡,并设置默认网络的IP信息
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
```

```
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.2/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
31: eth1@if32: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth1
       valid_lft forever preferred_lft forever


#test2可以连接test1容器
/ # ping -c1 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.128 ms

--- 172.17.0.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.128/0.128/0.128 ms


#在test1中可以利用test2容器名通信
/ # ping -c1 test2
PING test2 (172.27.0.2): 56 data bytes
64 bytes from 172.27.0.2: seq=0 ttl=64 time=0.076 ms

#在test2中可以利test1容器名通信
/ # ping -c1 test1
PING test1 (172.27.0.3): 56 data bytes
64 bytes from 172.27.0.3: seq=0 ttl=64 time=0.075 ms
```

4.3.8.2.4 断开不同网络中容器的通信

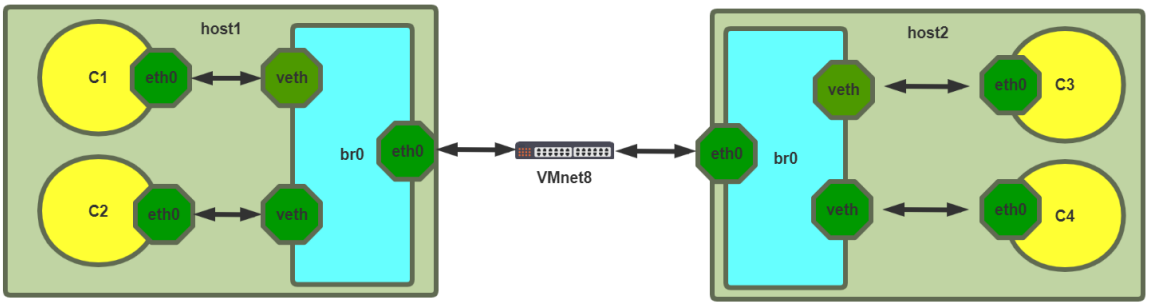
```
#将test1 断开和网络test-net中其它容器的通信
[root@ubuntu1804 ~]#docker network  disconnect test-net test1

#在容器test1中无法和test2通信
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
27: eth0@if28: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping  -c1 172.27.0.2
PING 172.27.0.2 (172.27.0.2): 56 data bytes

--- 172.27.0.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss

#在容器test2中仍能和test1通信
/ # ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.2/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
31: eth1@if32: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:00:03 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.3/16 brd 172.17.255.255 scope global eth1
       valid_lft forever preferred_lft forever
/ # ping -c1 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.085 ms



#将test2 断开和默认网络中其它容器的通信
[root@ubuntu1804 ~]#docker network  disconnect bridge test2


#在容器test2中无法和test1通信
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
23: eth0@if24: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:1b:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.27.0.2/16 brd 172.27.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping -c1 172.17.0.2
PING 172.17.0.2 (172.17.0.2): 56 data bytes

--- 172.17.0.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

4.4 实现跨宿主机的容器之间网络互联

同一个宿主机之间的各个容器之间是可以直接通信的，但是如果访问到另外一台宿主机的容器呢?

4.4.1 方式1: 利用桥接实现跨宿主机的容器间互联




```
#分别将两个宿主机都执行下面操作
[root@ubuntu1804 ~]#apt -y install bridge-utils
[root@ubuntu1804 ~]#brctl addif docker0 eth0

#在两个宿主机上各启动一个容器,需要确保IP不同,相互测试访问
#第一个宿主机的容器
[root@ubuntu1804 ~]#docker run -it --name b1 busybox
/ # hostname -i
172.17.0.2
/ # httpd -h /data/html/ -f -v
[::ffff:172.17.0.3]:42488:response:200

#第二个宿主机的容器
[root@ubuntu1804 ~]#docker run -it --name b2 busybox
/ # hostname -i
172.17.0.3
/#wget-q0 - http://172.17.0.2
httpd website in busybox
```

4.4.2 方式2: 利用NAT实现跨主机的容器间互联

4.4.2.1 docker跨主机互联实现说明

跨主机互联是说A宿主机的容器可以访问B主机上的容器，但是前提是保证各宿主机之间的网络是可以相互通信的，然后各容器才可以通过宿主机访问到对方的容器

实现原理：是在宿主机做一个网络路由就可以实现A宿主机的容器访问B主机的容器的目的

注意：此方式只适合小型网络环境，复杂的网络或者大型的网络可以使用google开源的k8s进行互联

4.4.2.2 修改各宿主机网段

Docker默认网段是172.17.0.x/24,而且每个宿主机都是一样的，因此要做路由的前提就是各个主机的网络不能一致

4.4.2.2.1 第一个宿主机A上更改网段

```
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
[root@ubuntu1804 ~]#cat /etc/docker/daemon.json
{
   "bip": "192.168.100.1/24",
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}

[root@ubuntu1804 ~]# systemctl restart docker
[root@ubuntu1804 ~]#ip a
```

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:6b:54:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.101/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6b:54d3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:e0:ef:72:05 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.1/24 brd 192.168.100.255 scope global docker0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:e0ff:feef:7205/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 docker0
```

4.4.2.2.2 第二个宿主机B更改网段

```
[root@ubuntu1804 ~]#vim /etc/docker/daemon.json
{
  "bip": "192.168.200.1/24",
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}

[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:01:f3:0c brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.102/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe01:f30c/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:e8:c0:a4:d8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.1/24 brd 192.168.200.255 scope global docker0
       valid_lft forever preferred_lft forever
```

```
    inet6 fe80::42:e8ff:fec0:a4d8/64 scope link
       valid_lft forever preferred_lft forever
[root@ubuntu1804 ~]#route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.200.0   0.0.0.0         255.255.255.0   U     0      0        0 docker0
```

4.4.2.3 在两个宿主机分别启动一个容器

第一个宿主机启动容器server1

```
[root@ubuntu1804 ~]#docker run -it --name server1 --rm  alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
16: eth0@if17: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:c0:a8:64:02 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.2/24 brd 192.168.100.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.100.1   0.0.0.0         UG    0      0        0 eth0
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

第二个宿主机启动容器server2

```
[root@ubuntu1804 ~]#docker run -it --name server2 --rm  alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
8: eth0@if9: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:c0:a8:c8:02 brd ff:ff:ff:ff:ff:ff
    inet 192.168.200.2/24 brd 192.168.200.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         192.168.200.1   0.0.0.0         UG    0      0        0 eth0
192.168.200.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

从第一个宿主机的容器server1无法和第二个宿主机的server2相互访问

```
[root@ubuntu1804 ~]#docker run -it --name server1 --rm  alpine sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
```

```
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
14: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:0a:64:00:02 brd ff:ff:ff:ff:ff:ff
    inet 10.100.0.2/16 brd 10.100.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping -c1 192.168.200.2
PING 192.168.200.2 (192.168.200.2): 56 data bytes

--- 192.168.200.2 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
```

4.4.2.4 添加静态路由和iptables规则

在各宿主机添加静态路由，网关指向对方宿主机的IP

4.4.2.4.1 在第一台宿主机添加静态路由和iptables规则

```
[root@ubuntu1804 ~]#route add -net 192.168.200.0/24 gw 10.0.0.102
[root@ubuntu1804 ~]#iptables -A FORWARD -s 10.0.0.0/24 -j ACCEPT
```

4.4.2.4.2 在第二台宿主机添加静态路由和iptables规则

```
[root@ubuntu1804 ~]#route add -net 192.168.100.0/24 gw 10.0.0.101
[root@ubuntu1804 ~]#iptables -A FORWARD -s 10.0.0.0/24 -j ACCEPT
```

4.4.2.5 测试跨宿主机之间容器互联

宿主机A的容器server1访问宿主机B容器server2，同时在宿主机B上tcpdump抓包观察

```
/ # ping -c1 192.168.200.2
PING 192.168.200.2 (192.168.200.2): 56 data bytes
64 bytes from 192.168.200.2: seq=0 ttl=62 time=1.022 ms

--- 192.168.200.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.022/1.022/1.022 ms


#宿主机B的抓包可以观察到
[root@ubuntu1804 ~]#tcpdump  -i eth0 -nn icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
16:57:37.912925 IP 10.0.0.101 > 192.168.200.2: ICMP echo request, id 2560, seq
0, length 64
16:57:37.913208 IP 192.168.200.2 > 10.0.0.101: ICMP echo reply, id 2560, seq 0,
length 64
```

宿主机B的容器server2访问宿主机B容器server1，同时在宿主机A上tcpdump抓包观察

```
/ # ping -c1 192.168.100.2
PING 192.168.100.2 (192.168.100.2): 56 data bytes
64 bytes from 192.168.100.2: seq=0 ttl=62 time=1.041 ms

--- 192.168.100.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.041/1.041/1.041 ms

#宿主机A的抓包可以观察到
[root@ubuntu1804 ~]#tcpdump -i eth0 -nn icmp
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
16:59:11.775784 IP 10.0.0.102 > 192.168.100.2: ICMP echo request, id 2560, seq
0, length 64
16:59:11.776113 IP 192.168.100.2 > 10.0.0.102: ICMP echo reply, id 2560, seq 0,
length 64
```

4.4.2.6 创建第三个容器测试

```
#在第二个宿主机B上启动第一个提供web服务的nginx容器server3
#注意无需打开端口映射
[root@ubuntu1804 ~]#docker run  -d --name server3 centos7-nginx:1.6.1
69fc554fd00e4f7880c139283b64f2701feafb91047b217906b188c1f461b699
[root@ubuntu1804 ~]#docker exec -it server3 bash
[root@69fc554fd00e /]# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.200.3  netmask 255.255.255.0  broadcast 192.168.200.255
        ether 02:42:c0:a8:c8:03  txqueuelen 0  (Ethernet)
        RX packets 8  bytes 656 (656.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[root@69fc554fd00e /]#

#从server1中访问server3的页面可以成功
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
14: eth0@if15: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:0a:64:00:02 brd ff:ff:ff:ff:ff:ff
    inet 10.100.0.2/16 brd 10.100.255.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # wget -qO - http://192.168.200.3/app
```

```
Test Page in app

/ #

#从server3容器观察访问日志，可以看到来自于第一个宿主机，而非server1容器
[root@69fc554fd00e /]# tail -f /apps/nginx/logs/access.log
10.0.0.101 - - [02/Feb/2020:09:02:00 +0000] "GET /app HTTP/1.1" 301 169 "-"
"Wget"


#用tcpdump抓包80/tcp的包，可以观察到以下内容
[root@ubuntu1804 ~]#tcpdump  -i eth0 -nn port 80
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
17:03:35.885627 IP 10.0.0.101.43578 > 192.168.200.3.80: Flags [S], seq
3672256868, win 29200, options [mss 1460,sackOK,TS val 4161963574 ecr
0,nop,wscale 7], length 0
17:03:35.885768 IP 192.168.200.3.80 > 10.0.0.101.43578: Flags [S.], seq
2298407060, ack 3672256869, win 28960, options [mss 1460,sackOK,TS val
3131173298 ecr 4161963574,nop,wscale 7], length 0
17:03:35.886312 IP 10.0.0.101.43578 > 192.168.200.3.80: Flags [.], ack 1, win
229, options [nop,nop,TS val 4161963575 ecr 3131173298], length 0
17:03:35.886507 IP 10.0.0.101.43578 > 192.168.200.3.80: Flags [P.], seq 1:80,
ack 1, win 229, options [nop,nop,TS val 4161963575 ecr 3131173298], length 79:
HTTP: GET /app HTTP/1.1
17:03:35.886541 IP 192.168.200.3.80 > 10.0.0.101.43578: Flags [.], ack 80, win
227, options [nop,nop,TS val 3131173299 ecr 4161963575], length 0
17:03:35.887179 IP 192.168.200.3.80 > 10.0.0.101.43578: Flags [P.], seq 1:365,
ack 80, win 227, options [nop,nop,TS val 3131173299 ecr 4161963575], length 364:
HTTP: HTTP/1.1 301 Moved Permanently
17:03:35.887222 IP 192.168.200.3.80 > 10.0.0.101.43578: Flags [F.], seq 365, ack
80, win 227, options [nop,nop,TS val 3131173299 ecr 4161963575], length 0
17:03:35.890139 IP 10.0.0.101.43580 > 192.168.200.3.80: Flags [.], ack
1660534352, win 229, options [nop,nop,TS val 4161963579 ecr 3131173301], length
0
17:03:35.890297 IP 10.0.0.101.43580 > 192.168.200.3.80: Flags [P.], seq 0:80,
ack 1, win 229, options [nop,nop,TS val 4161963579 ecr 3131173301], length 80:
HTTP: GET /app/ HTTP/1.1
17:03:35.890327 IP 192.168.200.3.80 > 10.0.0.101.43580: Flags [.], ack 80, win
227, opons [nop,nop,TS val 3131173303 ecr 4161963579], length 0
```

4.4.3 方式3: 利用Open vSwitch实现跨主机的容器间互联

4.4.3.1 Open vSwitch介绍

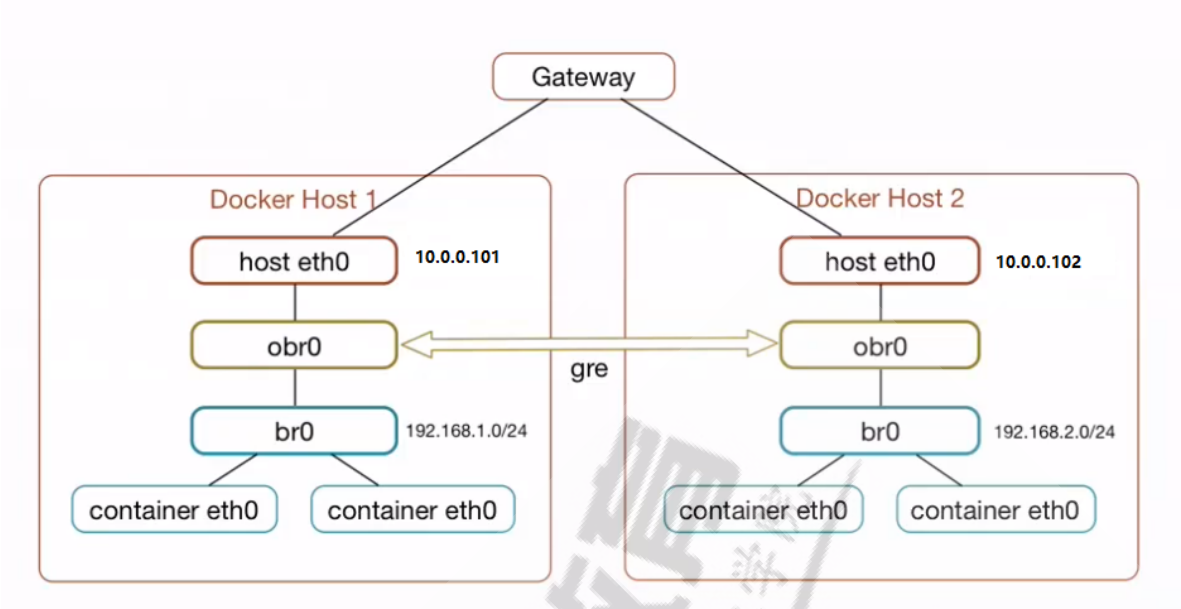
Open vSwitch，即Open Virtual Switch开放虚拟交换机，简称OVS, 是在开源的Apache2.0许可下的产品级质量的多层虚拟交换机。由Nicira Networks开发，主要实现代码为可移植的C代码。它的目的是让大规模网络自动化可以通过编程扩展，同时仍然支持标准的管理接口和协议(例如NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag) ,即Open vSwitch通过软件的方式实现了交换机功能

跟传统的物理交换机相比，虚拟交换机同样具备众多优点，一是配置更加灵活。一台普通的服务器可以配置出数十台甚至上百台虚拟交换机，且端口数目可以灵活选择。例如，VMware的ESXi一台服务器可以仿真出248台虚拟交换机，且每台交换机预设虚拟端口即可达56个；二是成本更加低廉，通过虚拟交换往往可以获得昂贵的普通交换机才能达到的性能，例如微软的Hyper-V平台，虚拟机与虚拟交换机之

间的联机速度轻易可达10Gbps。

官网: http://www.openvswitch.org/

使用Open vSwitch实现跨主机容器连接一原理



什么是GRE隧道?

> GRE:通用路由协议封装
> 隧道技术(Tunneling) 是一种通过使用互联网络的基础设施在网络之间传递数据的方式。使用隧道传递的数据(或负载)可以是不同协议的数据帧或包。隧道协议将其它协议的数据帧或包重新封装然后通过隧道发送。新的帧头提供路由信息,以便通过互联网传递被封装的负载数据。

4.3.3.2 利用Open vSwitch实现docker跨主机网络

实现目标: 将两台主机的容器利用Open vSwitch连接起来，实现互联互通

4.3.3.2.1 环境准备

主机名	操作系统	宿主机IP	Docker0 IP	容器 IP
ovs1	ubuntu 18.04	10.0.0.101/24	192.168.1.1/24	192.168.1.0/24
ovs2	ubuntu 18.04	10.0.0.102/24	192.168.2.1/24	192.168.2.0/24

4.3.3.2.2 修改两台主机的docker0分别使用不同的网段

```
#配置第一台主机
[root@ovs1 ~]#vim /etc/docker/daemon.json
{
  "bip": "192.168.1.1/24",
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}
[root@ovs1 ~]#systemctl restart docker
[root@ovs1 ~]#ip add show  docker0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:dc:29:03:6c brd ff:ff:ff:ff:ff:ff
```

```
    inet 192.168.1.1/24 brd 192.168.1.255 scope global docker0
       valid_lft forever preferred_lft forever


#配置第二台主机
[root@ovs2 ~]#vim /etc/docker/daemon.json
{
  "bip": "192.168.2.1/24",
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}
[root@ovs2 ~]#systemctl restart docker
[root@ovs2 ~]#ip add show  docker0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:e2:38:84:83 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/24 brd 192.168.2.255 scope global docker0
       valid_lft forever preferred_lft forever
```

4.3.3.2.3 在两个宿主机安装openvswitch-switch和bridge-utils和确认版本

```
#在第一个主机安装包
[root@ovs1 ~]#apt -y install openvswitch-switch bridge-utils
[root@ovs1 ~]#ps -e | grep ovs
  6766 ?        00:00:00 ovsdb-server
  6826 ?        00:00:00 ovs-vswitchd

#查看ovs版本信息和ovs支持的OpenFlow协议的版本
[root@ovs1 ~]#ovs-appctl --version
ovs-appctl (Open vSwitch) 2.9.5
[root@ovs1 ~]#ovs-ofctl --version
ovs-ofctl (Open vSwitch) 2.9.5
OpenFlow versions 0x1:0x5

#查看网桥
[root@ovs1 ~]#brctl  show
bridge name bridge id       STP enabled interfaces
docker0     8000.0242dc29036c   no

#在第二个主机安装包
[root@ovs2 ~]#apt -y install openvswitch-switch bridge-utils
[root@ovs2 ~]#ps -e | grep ovs
  6618 ?        00:00:00 ovsdb-server
  6680 ?        00:00:00 ovs-vswitchd

#查看ovs版本信息和ovs支持的OpenFlow协议的版本
[root@ovs2 ~]#ovs-appctl --version
ovs-appctl (Open vSwitch) 2.9.5
[root@ovs2 ~]#ovs-ofctl --version
ovs-ofctl (Open vSwitch) 2.9.5
OpenFlow versions 0x1:0x5

[root@ovs2 ~]#brctl  show
bridge name bridge id       STP enabled interfaces
docker0     8000.0242e2388483   no
```

4.3.3.2.4 在两个宿主机都创建obr0网桥并激活

```
[root@ovs1 ~]#ovs-vsctl add-br obr0
[root@ovs1 ~]#ip link set dev obr0 up
[root@ovs1 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:6b:54:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.101/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6b:54d3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:dc:29:03:6c brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global docker0
       valid_lft forever preferred_lft forever
4: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
default qlen 1000
    link/ether ce:ff:6f:7f:4b:11 brd ff:ff:ff:ff:ff:ff
5: obr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/ether f2:2b:d7:d8:a1:4d brd ff:ff:ff:ff:ff:ff
    inet6 fe80::f02b:d7ff:fed8:a14d/64 scope link
       valid_lft forever preferred_lft forever

#第二台主机重复上面
[root@ovs2 ~]#ovs-vsctl add-br obr0
[root@ovs2 ~]#ip link set dev obr0 up
[root@ovs2 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:01:f3:0c brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.102/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe01:f30c/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:e2:38:84:83 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/24 brd 192.168.2.255 scope global docker0
       valid_lft forever preferred_lft forever
4: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
default qlen 1000
    link/ether d6:29:ca:3a:9d:99 brd ff:ff:ff:ff:ff:ff
```

```
5: obr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/ether 82:4f:05:e3:5d:42 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::804f:5ff:fee3:5d42/64 scope link
       valid_lft forever preferred_lft forever
```

4.3.3.2.5 在两个宿主机创建gre隧道(remote_ip为peer宿主机ip)

注意: 如果有多台docker主机需要构建网络创建多个gre隧道

```
#一条命令实现,remote_ip指向另一台宿主机的IP
[root@ovs1 ~]#ovs-vsctl add-port obr0 gre0 -- set Interface gre0 type=gre
options:remote_ip=10.0.0.102
#或者两条命令实现
[root@ovs1 ~]#ovs-vsctl add-port obr0 gre0
[root@ovs1 ~]#ovs-vsctl set Interface gre0 type=gre options:remote_ip=10.0.0.102

[root@ovs1 ~]#ovs-vsctl list-ports obr0
gre0

[root@ovs1 ~]#ovs-vsctl show
84cbdad7-4731-4c2e-b7d7-eecb4a56d27b
    Bridge "obr0"
        Port "gre0"
            Interface "gre0"
                type: gre
                options: {remote_ip="10.0.0.102"}
        Port "obr0"
            Interface "obr0"
                type: internal
    ovs_version: "2.9.5"

[root@ovs1 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:6b:54:d3 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.101/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe6b:54d3/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:dc:29:03:6c brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global docker0
       valid_lft forever preferred_lft forever
4: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
default qlen 1000
    link/ether ce:ff:6f:7f:4b:11 brd ff:ff:ff:ff:ff:ff
5: obr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
group default qlen 1000
```

```
    link/ether f2:2b:d7:d8:a1:4d brd ff:ff:ff:ff:ff:ff
    inet6 fe80::f02b:d7ff:fed8:a14d/64 scope link
       valid_lft forever preferred_lft forever
6: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN group default qlen 1000
    link/gre 0.0.0.0 brd 0.0.0.0
7: gretap0@NONE: <BROADCAST,MULTICAST> mtu 1462 qdisc noop state DOWN group
default qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
8: erspan0@NONE: <BROADCAST,MULTICAST> mtu 1450 qdisc noop state DOWN group
default qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
9: gre_sys@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65000 qdisc fq_codel
master ovs-system state UNKNOWN group default qlen 1000
    link/ether ce:d2:c1:4e:be:c6 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::ccd2:c1ff:fe4e:bec6/64 scope link
       valid_lft forever preferred_lft forever

#配置第二个宿主机
[root@ovs2 ~]#ovs-vsctl add-port obr0 gre0 -- set Interface gre0 type=gre
options:remote_ip=10.0.0.101
[root@ovs2 ~]#ovs-vsctl list-ports obr0
gre0

[root@ovs2 ~]#ovs-vsctl show
e6a3aab3-e224-4834-85fc-2516b33a67e2
    Bridge "obr0"
        Port "gre0"
            Interface "gre0"
                type: gre
                options: {remote_ip="10.0.0.101"}
        Port "obr0"
            Interface "obr0"
                type: internal
    ovs_version: "2.9.5"

[root@ovs2 ~]#ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 00:0c:29:01:f3:0c brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.102/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fe01:f30c/64 scope link
       valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:e2:38:84:83 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/24 brd 192.168.2.255 scope global docker0
       valid_lft forever preferred_lft forever
4: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group
default qlen 1000
    link/ether d6:29:ca:3a:9d:99 brd ff:ff:ff:ff:ff:ff
```

```
5: obr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
group default qlen 1000
    link/ether 82:4f:05:e3:5d:42 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::804f:5ff:fee3:5d42/64 scope link
       valid_lft forever preferred_lft forever
6: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN group default qlen 1000
    link/gre 0.0.0.0 brd 0.0.0.0
7: gretap0@NONE: <BROADCAST,MULTICAST> mtu 1462 qdisc noop state DOWN group
default qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
8: erspan0@NONE: <BROADCAST,MULTICAST> mtu 1450 qdisc noop state DOWN group
default qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
10: gre_sys@NONE: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65000 qdisc fq_codel
master ovs-system state UNKNOWN group default qlen 1000
    link/ether 0a:98:48:d9:5f:83 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::898:48ff:fed9:5f83/64 scope link
       valid_lft forever preferred_lft forever
```

4.3.3.2.6 在两个宿主机将obr0作为接口加入docker0网桥

```
#第一台宿主机执行
[root@ovs1 ~]#brctl addif docker0 obr0
[root@ovs1 ~]#brctl show
bridge name bridge id       STP enabled interfaces
docker0     8000.0242dc29036c   no      obr0

#第二台宿主机执行同样操作
[root@ovs2 ~]#brctl addif docker0 obr0
[root@ovs2 ~]#brctl show
bridge name bridge id       STP enabled interfaces
docker0     8000.0242e2388483   no      obr0
```

4.3.3.2.7 在两个宿主机添加静态路由(网段地址为 peer Docker网段)

```
#ovs1 添加 peer docker net
[root@ovs1 ~]#ip route add 192.168.2.0/24 dev docker0
[root@ovs1 ~]#route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 docker0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 docker0

#ovs2 添加 peer docker net
[root@ovs2 ~]#ip route add 192.168.1.0/24 dev docker0
[root@ovs2 ~]#route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
0.0.0.0         10.0.0.2        0.0.0.0         UG    0      0        0 eth0
10.0.0.0        0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 docker0
192.168.2.0     0.0.0.0         255.255.255.0   U     0      0        0 docker0
```

4.3.3.2.8 在两个宿主机测试跨主机的容器之间的连通性

```
[root@ovs1 ~]#docker run -it alpine /bin/sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN qlen 1000
    link/gre 0.0.0.0 brd 0.0.0.0
3: gretap0@NONE: <BROADCAST,MULTICAST> mtu 1462 qdisc noop state DOWN qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
4: erspan0@NONE: <BROADCAST,MULTICAST> mtu 1450 qdisc noop state DOWN qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
10: eth0@if11: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:01:02 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.2/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping -c 3 192.168.2.2
PING 192.168.2.2 (192.168.2.2): 56 data bytes
64 bytes from 192.168.2.2: seq=0 ttl=63 time=4.459 ms
64 bytes from 192.168.2.2: seq=1 ttl=63 time=1.279 ms
64 bytes from 192.168.2.2: seq=2 ttl=63 time=0.517 ms

--- 192.168.2.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.517/2.085/4.459 ms


[root@ovs2 ~]#docker run -it alpine /bin/sh
/ # ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: gre0@NONE: <NOARP> mtu 1476 qdisc noop state DOWN qlen 1000
    link/gre 0.0.0.0 brd 0.0.0.0
3: gretap0@NONE: <BROADCAST,MULTICAST> mtu 1462 qdisc noop state DOWN qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
4: erspan0@NONE: <BROADCAST,MULTICAST> mtu 1450 qdisc noop state DOWN qlen 1000
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
11: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
state UP
    link/ether 02:42:ac:11:02:02 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.2/24 brd 192.168.2.255 scope global eth0
       valid_lft forever preferred_lft forever
/ # ping -c 3 192.168.1.2
PING 192.168.1.2 (192.168.1.2): 56 data bytes
64 bytes from 192.168.1.2: seq=0 ttl=63 time=1.553 ms
64 bytes from 192.168.1.2: seq=1 ttl=63 time=1.136 ms
64 bytes from 192.168.1.2: seq=2 ttl=63 time=1.176 ms

--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 1.136/1.288/1.553 ms
/ #
```

在第二个主机上再打开一个nginx容器，从第一个主机的容器访问，观察来源的IP

```
[root@ovs2 ~]#docker pull nginx
[root@ovs2 ~]#docker run -d --name nginx nginx
d3c26005a7626628f7baf017481217b36e3d69dabfa6cc86fe125f9548e7333c
[root@ovs2 ~]#docker exec -it nginx hostname -I
192.168.2.2
[root@ovs2 ~]#docker logs -f nginx
192.168.1.2 - - [27/Feb/2020:09:57:18 +0000] "GET / HTTP/1.1" 200 612 "-" "Wget"
"-"


#从第一个主机的容器发起请求，可以查看到上面的访问日志输出
[root@ovs1 ~]#docker run -it alpine wget -qO -  http://192.168.2.2/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

4.3.3.2.9 在两个宿主机用脚本保存配置用于开机启动

```
#ovs1配置
[root@ovs1 ~]#cat > net.sh <<EOF
#!/bin/bash
ip link set dev obr0 up
brctl addif docker0 obr0
ip route add 192.168.2.0/24 dev docker0
EOF
[root@ovs1 ~]#chmod +x net.sh

#ovs2配置
[root@ovs2 ~]#cat > net.sh <<EOF
```

```
#!/bin/bash
ip link set dev obr0 up
brctl addif docker0 obr0
ip route add 192.168.1.0/24 dev docker0
EOF
[root@ovs1 ~]#chmod +x net.sh
```

4.4.4 方式4: 使用 weave 实现跨主机的容器间互联

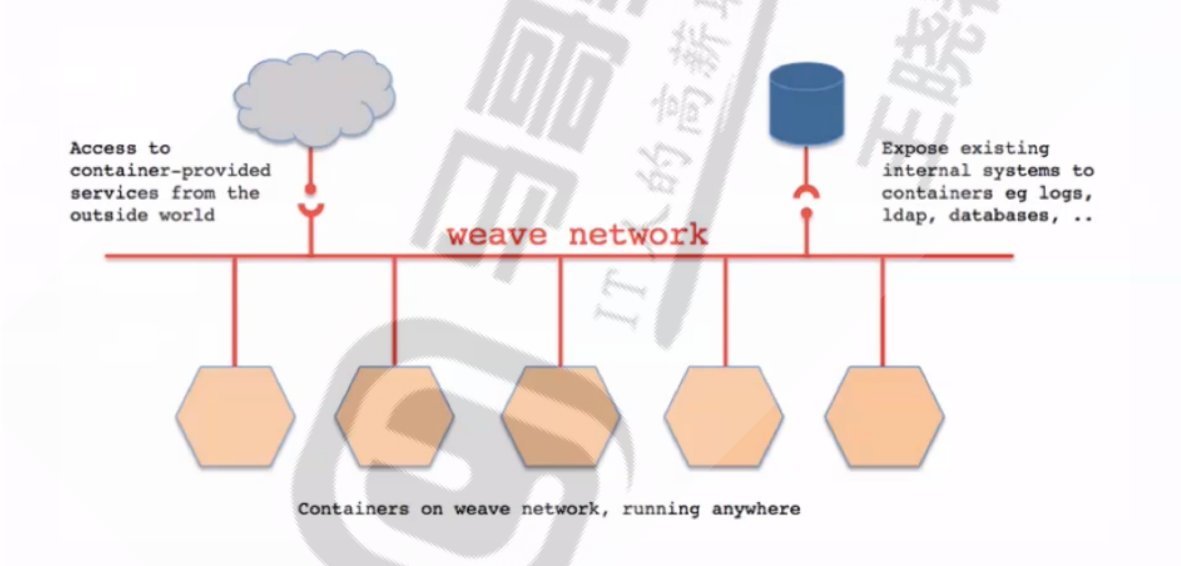
4.4.4.1 weave 介绍

weave 原意编织,意在建立一个虚拟的网络，用于将运行在不同主机的Docker容器连接起来

官网: http://weave.works
github链接: https://github.com/weaveworks/weave#readme

4.4.4.2 weave实现跨主机容器互联的流程

官方文档: https://www.weave.works/docs/net/latest/install/using-weave/



weave实现跨主机容器互联的流程

- 安装weave
- 启动weave `$weave launch`
- 连接不同主机
- 通过weave启动容器

4.4.4.3 实战案例: 通过weave 实现跨主机容器的互联

4.4.4.3.1 环境准备

主机名	操作系统	宿主机IP	Docker0 IP	容器
ubuntu1804	ubuntu 18.04	10.0.0.100/24	172.17.0.1/16	a1
centos7	centos 7.8	10.0.0.200/24	172.17.0.1/16	a2

4.4.4.3.2 安装weave

```
[root@ubuntu1804 ~]#wget -O /usr/bin/weave
https://raw.githubusercontent.com/zettio/weave/master/weave
--2020-07-23 17:05:08--
https://raw.githubusercontent.com/zettio/weave/master/weave
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
151.101.108.133
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|151.101.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 52484 (51K) [text/plain]
Saving to: '/usr/bin/weave'

/usr/bin/weave              100%[=========================================>]
 51.25K   174KB/s    in 0.3s

2020-07-23 17:05:15 (174 KB/s) - '/usr/bin/weave' saved [52484/52484]

[root@ubuntu1804 ~]#ll /usr/bin/weave
-rw-r--r-- 1 root root 52484 Jul 23 17:05 /usr/bin/weave

#在第二个宿主机重复一样的操作
[root@centos7 ~]#wget -O /usr/bin/weave
https://raw.githubusercontent.com/zettio/weave/master/weave[root@centos7 ~]#ll
/usr/bin/weave -h
-rw-r--r-- 1 root root 52K Jul 23 17:08 /usr/bin/weave
```

4.4.4.3.3 第一个宿主机启动weave

```
[root@ubuntu1804 ~]#chmod +x /usr/bin/weave
[root@ubuntu1804 ~]#weave launch
latest: Pulling from weaveworks/weave
21c83c524219: Pull complete
cfdfcbee9cb6: Pull complete
a56e93dd024c: Pull complete
fea445d5ce38: Pull complete
59db32ebe99d: Pull complete
Digest: sha256:743ae84161f97bba965d5565344b04ff40770b32b06e1f55c8dfb1250c880e88
Status: Downloaded newer image for weaveworks/weave:latest
docker.io/weaveworks/weave:latest
latest: Pulling from weaveworks/weavedb
72bf8a6af285: Pull complete
Digest: sha256:7badb003b9c0bf5c51bf801be2a4d5d371f0738818f9cbe60a508f54fd07de9a
Status: Downloaded newer image for weaveworks/weavedb:latest
docker.io/weaveworks/weavedb:latest
Unable to find image 'weaveworks/weaveexec:latest' locally
latest: Pulling from weaveworks/weaveexec
21c83c524219: Already exists
cfdfcbee9cb6: Already exists
a56e93dd024c: Already exists
fea445d5ce38: Already exists
59db32ebe99d: Already exists
12d4ba695a4d: Pull complete
298ee7d058fb: Pull complete
85e41461f1eb: Pull complete
f322a47dd011: Pull complete
```

```
Digest: sha256:e666e66bf10c9da5dce52b777e86f9fbb62157169614a80f2dbe324b335c5602
Status: Downloaded newer image for weaveworks/weaveexec:latest
0244d489c1dfe1b403a6e1cf228f260e84cafabea408d8cdabee1b0ca09c7519
```

4.4.4.3.4 启动第二宿主机的weave并连接第一个宿主机

```
#在第二个宿主机启动weave
[root@centos7 ~]#chmod +x /usr/bin/weave
[root@centos7 ~]#weave  launch 10.0.0.100
latest: Pulling from weaveworks/weave
21c83c524219: Pull complete
cfdfcbee9cb6: Pull complete
a56e93dd024c: Pull complete
fea445d5ce38: Pull complete
59db32ebe99d: Pull complete
Digest: sha256:743ae84161f97bba965d5565344b04ff40770b32b06e1f55c8dfb1250c880e88
Status: Downloaded newer image for weaveworks/weave:latest
docker.io/weaveworks/weave:latest
latest: Pulling from weaveworks/weavedb
72bf8a6af285: Pull complete
Digest: sha256:7badb003b9c0bf5c51bf801be2a4d5d371f0738818f9cbe60a508f54fd07de9a
Status: Downloaded newer image for weaveworks/weavedb:latest
docker.io/weaveworks/weavedb:latest
Unable to find image 'weaveworks/weaveexec:latest' locally
latest: Pulling from weaveworks/weaveexec
21c83c524219: Already exists
cfdfcbee9cb6: Already exists
a56e93dd024c: Already exists
fea445d5ce38: Already exists
59db32ebe99d: Already exists
12d4ba695a4d: Pull complete
298ee7d058fb: Pull complete
85e41461f1eb: Pull complete
f322a47dd011: Pull complete
Digest: sha256:e666e66bf10c9da5dce52b777e86f9fbb62157169614a80f2dbe324b335c5602
Status: Downloaded newer image for weaveworks/weaveexec:latest
ab2ebef1670374ae81c3031c9de96df4136e55749f914eaddcb8e5e5aee60c6c

#查看两个宿主机的连接
[root@centos7 ~]#ss -nt
State      Recv-Q Send-Q    LocalAddress:Port      Peer Address:Port
ESTAB      0      0         10.0.0.200:46521        10.0.0.100:6783
```

4.4.4.3.5 通过weave 启动容器

```
#第一个宿主机初始化环境
[root@ubuntu1804 ~]#eval $(weave env)
#第一个宿主机开启动容器
[root@ubuntu1804 ~]#docker run --name a1 -ti weaveworks/ubuntu
root@a1:/# hostname -i
10.32.0.1
root@a1:/# ping -c1 a2
PING a2 (10.44.0.0) 56(84) bytes of data.
64 bytes from a2.weave.local (10.44.0.0): icmp_seq=1 ttl=64 time=3.33 ms

#第二个宿主机初始化环境
[root@centos7 ~]#eval $(weave env)
```

```
#第二个宿主机开启动容器
[root@centos7 ~]#docker run --name a2 -ti weaveworks/ubuntu
root@a2:/# hostname -i
10.44.0.0
root@a2:/# ping a1 -c1
PING a1 (10.32.0.1) 56(84) bytes of data.
64 bytes from a1.weave.local (10.32.0.1): icmp_seq=1 ttl=64 time=0.474 ms
```

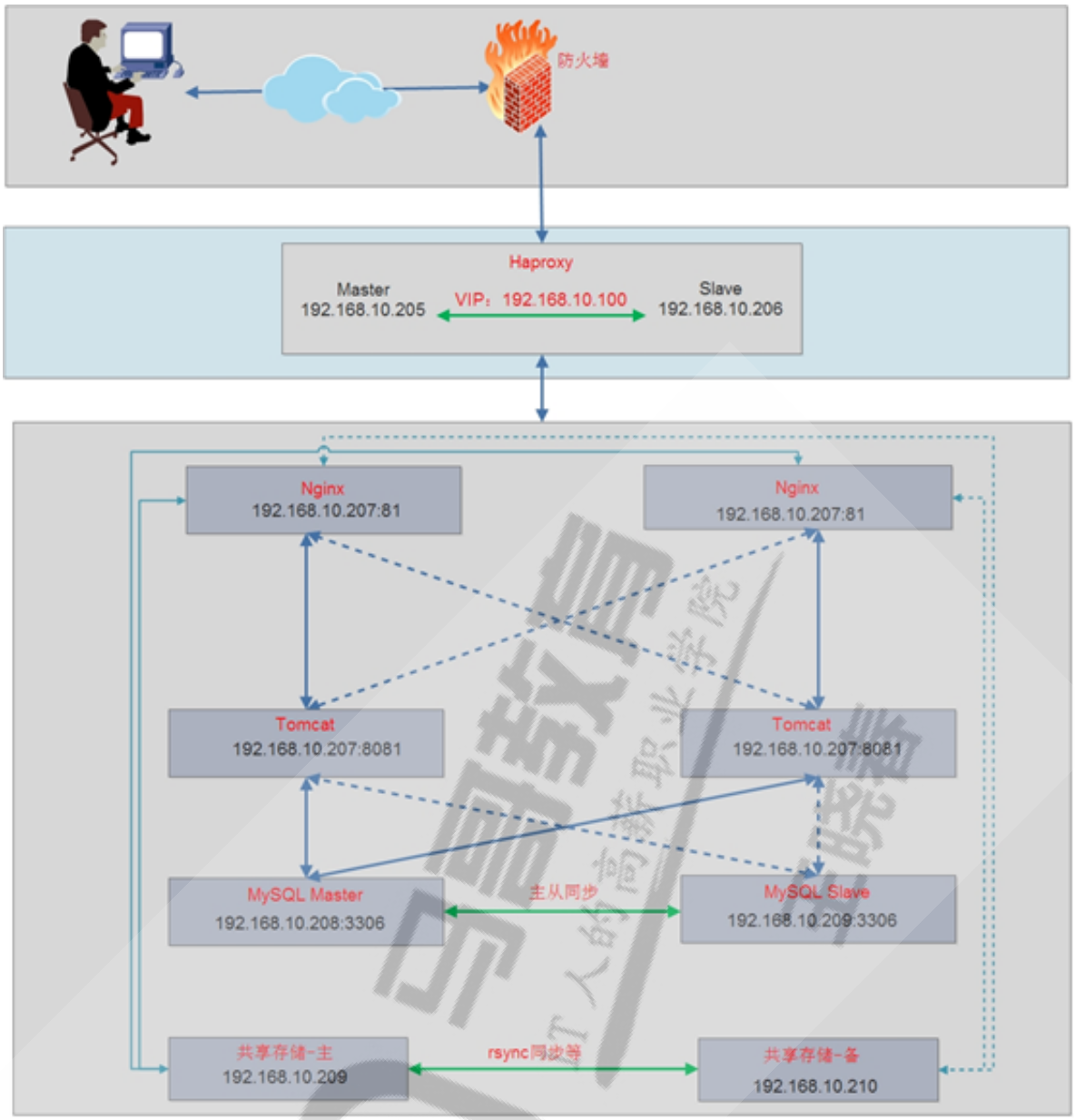
4.4.4.3.6 查看宿主机的容器

```
#自动生成了weave相关的容器
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                              COMMAND
CREATED             STATUS                          PORTS               NAMES
9374a414902a        weaveworks/ubuntu                  "/w/w /bin/bash"          5
minutes ago       Exited (0) 2 minutes ago                       a1
0244d489c1df        weaveworks/weave:latest            "/home/weave/weaver …"    37
minutes ago       Up 37 minutes                                  weave
7048529f3ac6        weaveworks/weaveexec:latest        "data-only"               37
minutes ago       Created                                        weavevolumes-
latest
7fa02af2b482        weaveworks/weavedb:latest          "data-only"               37
minutes ago       Created                                        weavedb


[root@centos7 ~]#docker ps -a
CONTAINER ID        IMAGE                              COMMAND
CREATED             STATUS                  PORTS               NAMES
be519cda22f7        weaveworks/ubuntu                  "/w/w /bin/bash"          3
minutes ago       Up 3 minutes                                   a2
ab2ebef16703        weaveworks/weave:latest            "/home/weave/weaver …"    31
minutes ago       Up 31 minutes                                  weave
f7b21ebd2b09        weaveworks/weaveexec:latest        "data-only"               31
minutes ago       Created                                        weavevolumes-latest
b7879a1e77b8        weaveworks/weavedb:latest          "data-only"               31
minutes ago       Created                                        weavedb
```

4.5 实战案例: 利用docker结合负载实现网络架构高可用

4.5.1 整体规划图

下图为一个小型的网络架构图，其中 nginx 使用docker 运行



环境准备

主机名	操作系统	宿主机IP	Docker0 IP	容器
docker-server1	ubuntu 18.04	192.168.10.205/24	172.17.0.1/16	nginx-web1
docker-server2	ubuntu 18.04	192.168.10.206/24	172.17.0.1/16	nginx-web2

4.5.2 安装并配置keepalived

4.5.2.1 Server1 安装并配置

```
[root@docker-server1 ~]# yum install keepalived -y
[root@docker-server1 ~]# cat /etc/keepalived/keepalived.conf
vrrp_instance MAKE_VIP_INT {
  state MASTER
  interface eth0
  virtual_router_id 1
  priority 100
```

```
  advert_int 1
  unicast_src_ip 192.168.10.205
  unicast_peer {
   192.168.10.206
  }

   authentication {
   auth_type PASS
   auth_pass 1111
  }

  virtual_ipaddress {
    192.168.10.100/24 dev eth0 label eth0:1
  }
}

[root@docker-server1~]# systemctl restart keepalived && systemctl enable
keepalived
```

4.5.2.2 Server2 安装并配置

```
[root@docker-server2 ~]# yum install keepalived -y
[root@docker-server2 ~]# cat /etc/keepalived/keepalived.conf
vrrp_instance MAKE_VIP_INT {
  state BACKUP
  interface eth0
  virtual_router_id 1
  priority 50
  advert_int 1
  unicast_src_ip 192.168.10.206
  unicast_peer {
    192.168.10.205
  }



  authentication {
    auth_type PASS
    auth_pass 1111
  }

  virtual_ipaddress {
    192.168.10.100/24 dev eth0 label eth0:1
  }
}

[root@docker-server2 ~]# systemctl restart keepalived && systemctl enable
keepalived
```

4.5.3 安装并配置haproxy

4.5.3.1 修改系统内核使其可以监听本地不存在的IP

```
[root@docker-server1 ~]# sysctl -w net.ipv4.ip_nonlocal_bind=1
[root@docker-server2 ~]# sysctl -w net.ipv4.ip_nonlocal_bind=1
```

4.5.3.2 Server1安装并配置haproxy

```
[root@docker-server1 ~]# yum install haproxy -y
[root@docker-server1 ~]# cat /etc/haproxy/haproxy.cfg
global
maxconn 100000
uid 99
gid 99
daemon
nbproc 1
log 127.0.0.1 local0 info
defaults
option http-keep-alive
#option forwardfor
maxconn 100000
mode tcp
timeout connect 500000ms
timeout client 500000ms
timeout server 500000ms
listen stats
 mode http
 bind 0.0.0.0:9999
 stats enable
 log global
 stats uri   /haproxy-status
 stats auth  haadmin:q1w2e3r4ys


#---------------------------------------------------------------------

frontend docker_nginx_web
  bind 192.168.10.100:80
  mode http
  default_backend docker_nginx_hosts

backend docker_nginx_hosts
  mode http
  #balance source
  balance roundrobin

server 192.168.10.205  192.168.10.205:81 check inter 2000 fall 3 rise 5
server 192.168.10.206  192.168.10.206:81 check inter 2000 fall 3 rise 5
```

4.5.3.3 Server2安装并配置haproxy

```
[root@docker-server2 ~]# yum install haproxy -y
[root@docker-server2 ~]# cat /etc/haproxy/haproxy.cfg
global
maxconn 100000
uid 99
gid 99
```

```
daemon
nbproc 1
log 127.0.0.1 local0 info

defaults
option http-keep-alive

#option forwardfor
maxconn 100000
mode tcp
timeout connect 500000ms
timeout client 500000ms
timeout server 500000ms

listen stats
 mode http
 bind 0.0.0.0:9999
 stats enable
 log global
 stats uri   /haproxy-status
 stats auth  haadmin:q1w2e3r4ys


#=============================================================

frontend docker_nginx_web
  bind 192.168.10.100:80
  mode http
  default_backend docker_nginx_hosts


backend docker_nginx_hosts

  mode http
  #balance source
  balance roundrobin
  server 192.168.10.205  192.168.10.205:81 check inter 2000 fall 3 rise 5
  server 192.168.10.206  192.168.10.206:81 check inter 2000 fall 3 rise 5
```

4.5.3.4 各服务器别分启动haproxy

```
[root@docker-server1 ~]# systemctl enable haproxy
Created symlink from /etc/systemd/system/multi-user.target.wants/haproxy.service
to /usr/lib/systemd/system/haproxy.service.
[root@docker-server1 ~]# systemctl restart haproxy
[root@docker-server2 ~]# systemctl enable haproxy
Created symlink from /etc/systemd/system/multi-user.target.wants/haproxy.service
to /usr/lib/systemd/system/haproxy.service.

[root@docker-server2 ~]# systemctl restart haproxy
```

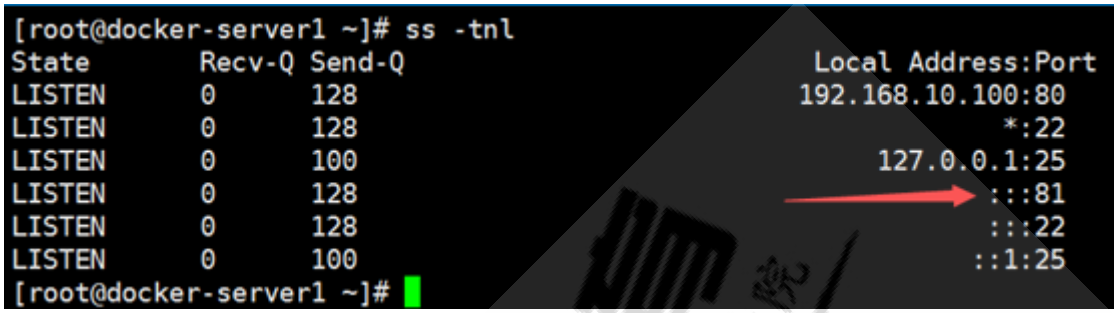
4.5.4 服务器启动nginx容器并验证

4.5.4.1 Server1 启动Nginx 容器

从本地Nginx 镜像启动一个容器，并指定端口，默认协议是tcp方式

```
[root@docker-server1 ~]# docker rm -f `docker ps -a -q`  #先删除之前所有的容器
[root@docker-server1 ~]# docker run --name nginx-web1 -d -p 81:80 nginx-
1.10.3:v1 nginx
5410e4042f731d2abe100519269f9241a7db2b3a188c6747b28423b5a584d020
```

4.5.4.2 验证端口



4.5.4.3 验证web访问

test nginx page

4.5.4.3 Server2 启动nginx 容器

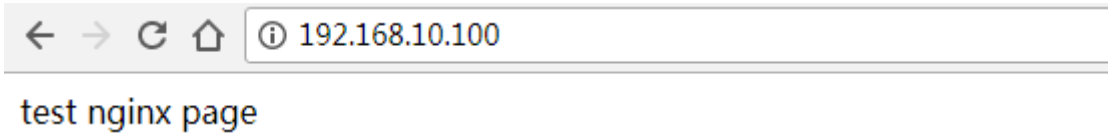
```
[root@docker-server2 ~]# docker run --name nginx-web1 -d -p 81:80 nginx-
1.10.3:v1 nginx
84f2376242e38d7c8ba7fabf3134ac0610ab26358de0100b151df6a231a2b56a
```

4.5.4.4 验证端口

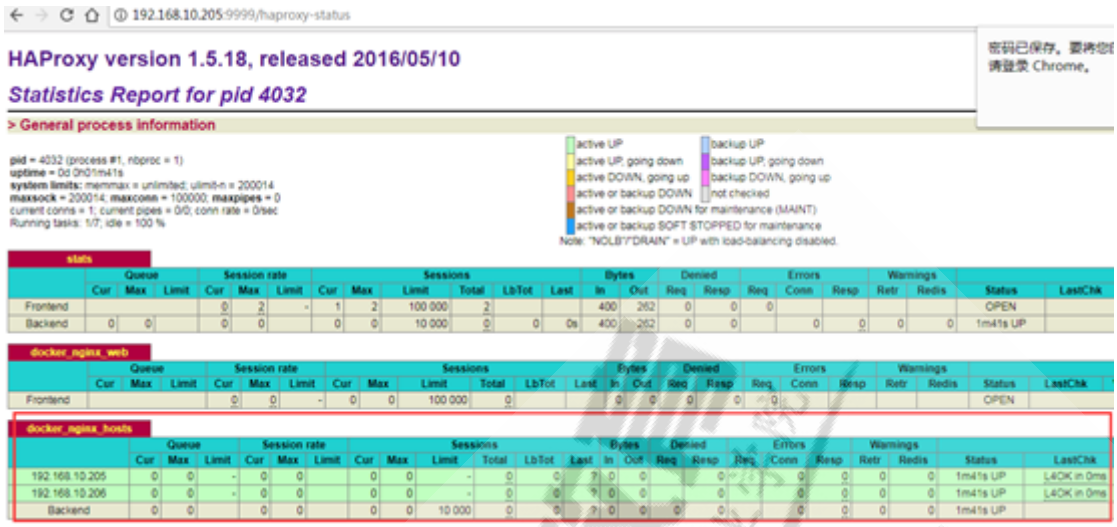
4.5.4.5 验证web访问

test nginx page

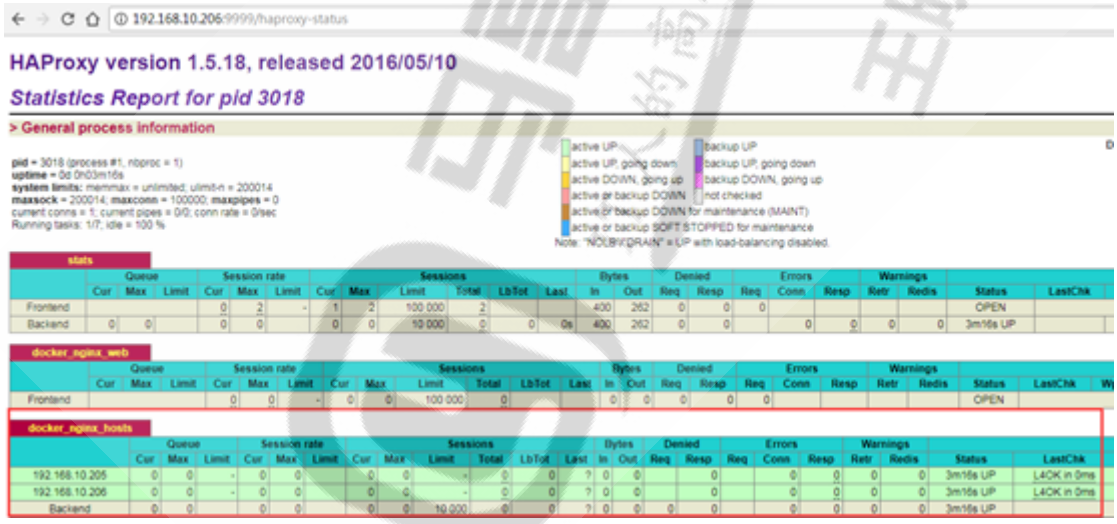
4.5.4.6 访问VIP



4.5.4.7 Server1 haproxy状态页面



4.5.4.8 Server2 haproxy状态页面



5 Docker 仓库管理

Docker仓库，类似于yum仓库，是用来保存镜像的仓库。为了方便的管理和使用docker镜像,可以将镜像集中保存至Docker仓库中，将制作好的镜像push到仓库集中保存，在需要镜像时，从仓库中pull镜像即可。

Docker 仓库分为公有云仓库和私有云仓库

公有云仓库：由互联网公司对外公开的仓库

- 官方
- 阿里云等第三方仓库

私有云仓库：组织内部搭建的仓库，一般只为组织内部使用，常使用下面软件搭建仓库

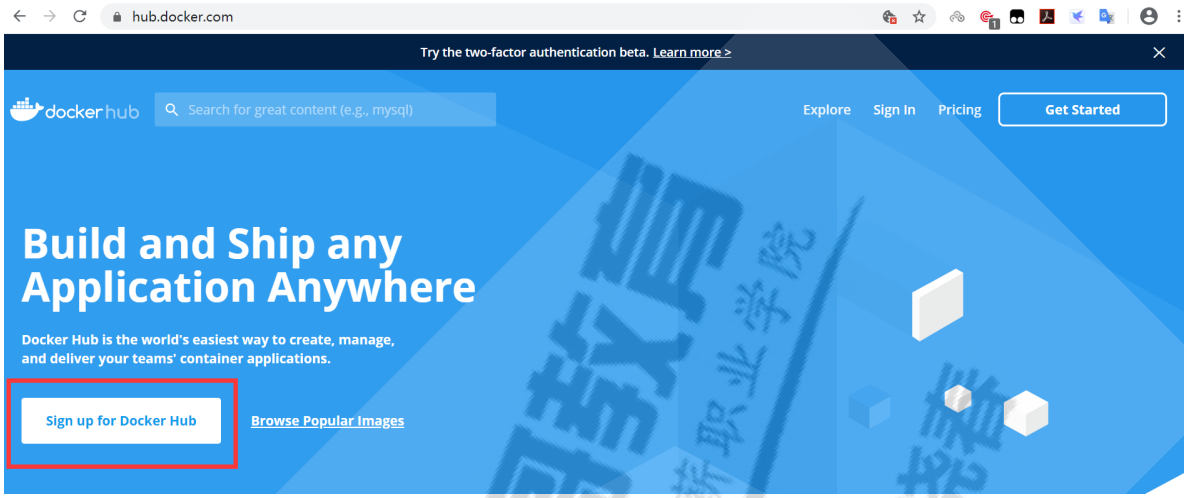
- docker registory
- docker harbor

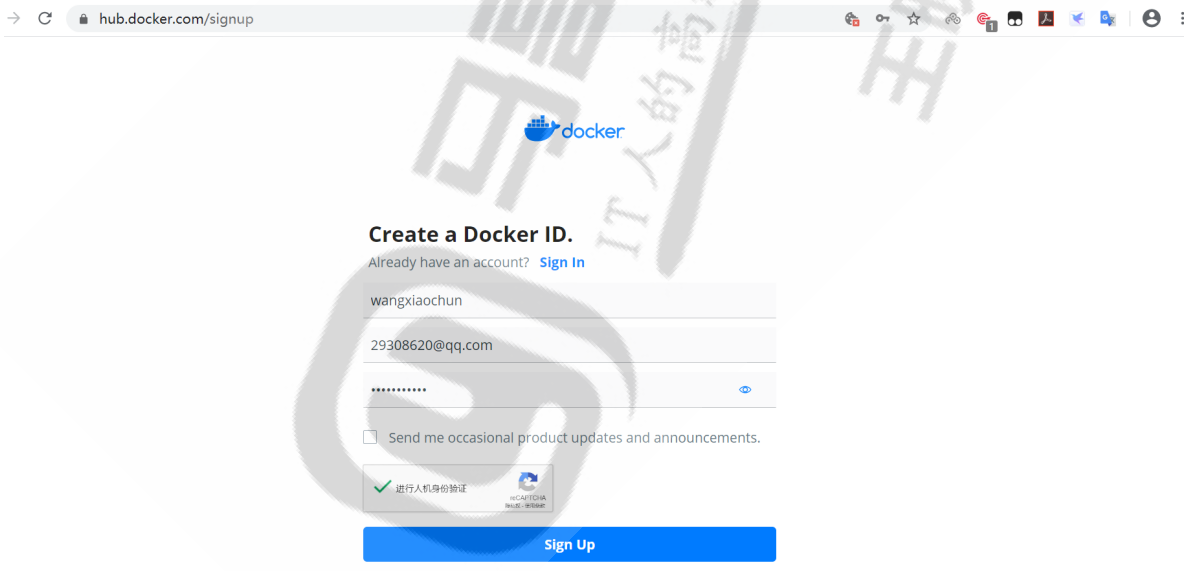
5.1 官方 docker 仓库

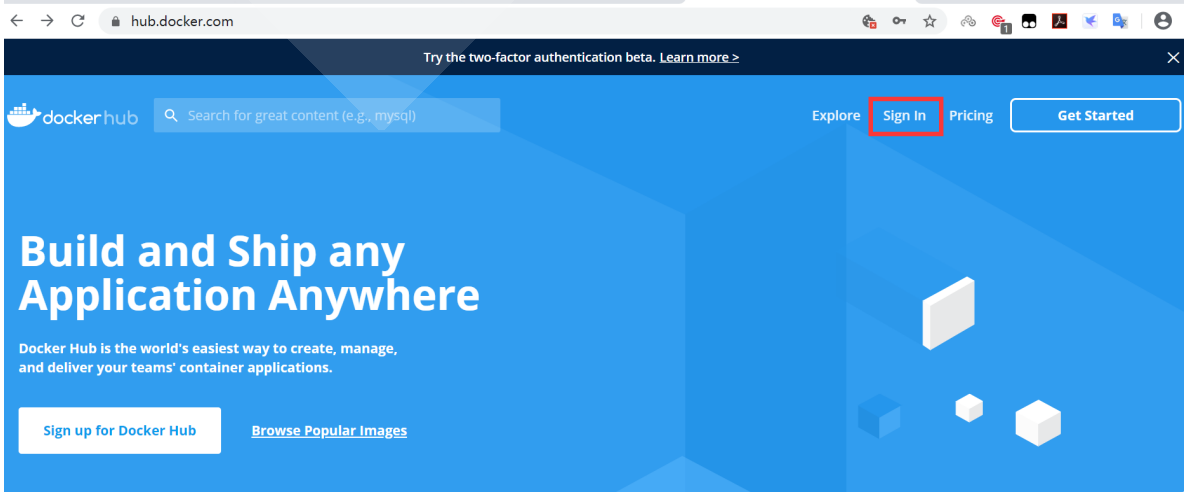
将自制的镜像上传至docker仓库；https://hub.docker.com/

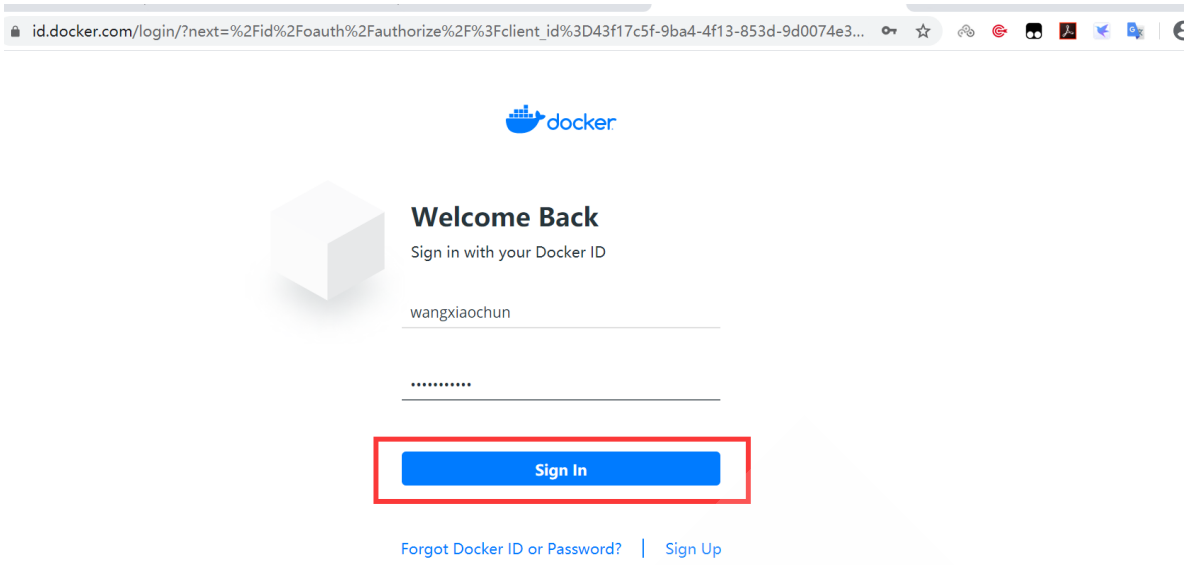
5.1.1 注册账户

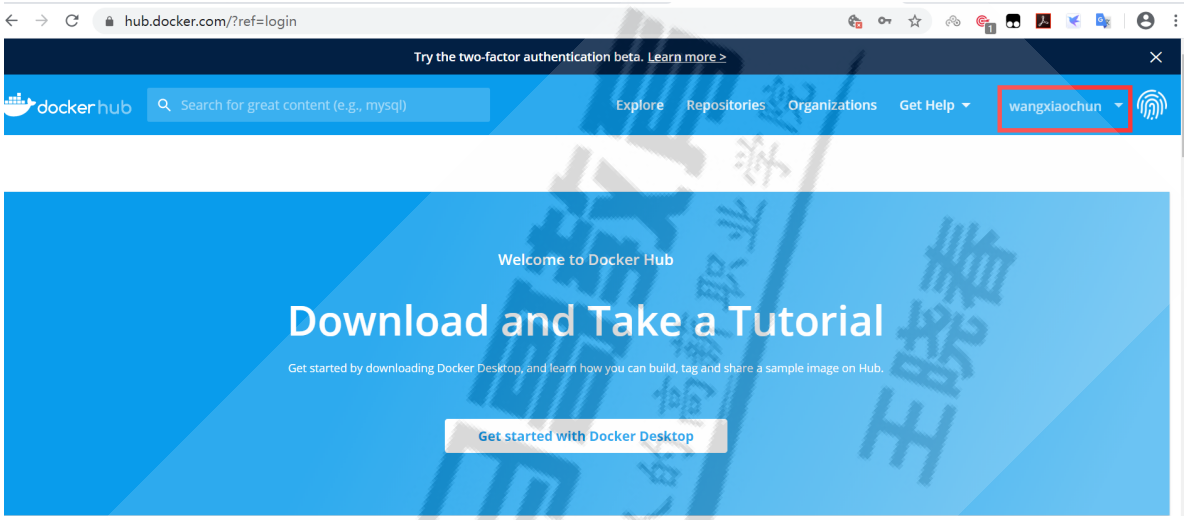
访问hub.docker.com注册账户，并登录











5.1.2 使用用户仓库管理镜像

每个注册用户都可以上传和管理自己的镜像

5.1.2.1 用户登录

上传镜像前需要执行docker login命令登录，登录后生成~/.docker/config.json文件保存验证信息

格式

```
docker login [OPTIONS] [SERVER]
选项:
-p, --password string   Password
    --password-stdin    Take the password from stdin
-u, --username string   Username
```

范例:

```
#登录docker官方仓库方法1
[root@ubuntu1804 ~]#docker login  -u wangxiaochun -pP@sswOrd!  docker.io
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

```
Login Succeeded

#登录docker官方仓库方法2
[root@ubuntu1804 ~]#docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: wangxiaochun
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ubuntu1804 ~]#

#登录成功后,自动生成验证信息,下次会自动登录,而无需手动登录
[root@ubuntu1804 ~]#cat .docker/config.json
{
    "auths": {
        "https://index.docker.io/v1/": {
            "auth": "d2FuZ3hpYW9jaHVuOmxidG9vdGgwNjE4"
        }
    },

    "HttpHeaders": {
        "User-Agent": "Docker-Client/19.03.5 (linux)"
    }
}[root@ubuntu1804 ~]#
```

5.1.2.2 给本地镜像打标签

上传本地镜像前必须先给上传的镜像用docker tag 命令打标签

标签格式: `docker.io/用户帐号/镜像名:TAG`

范例:

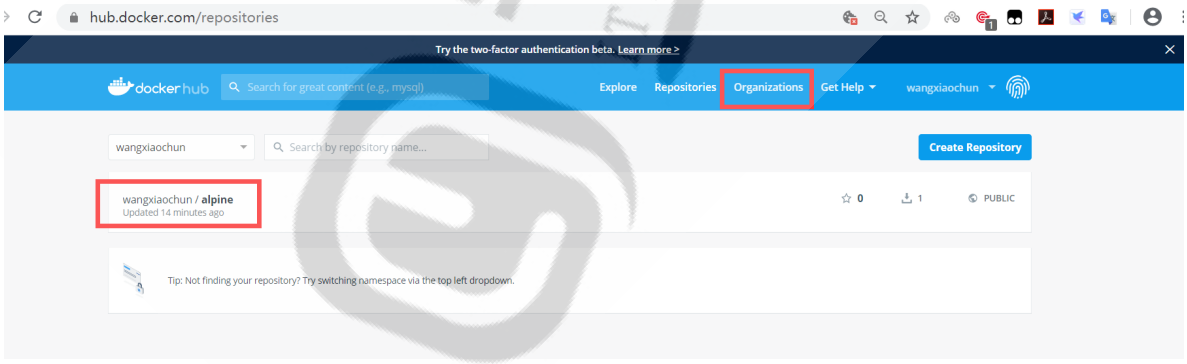
```
[root@ubuntu1804 ~]#docker tag alpine:3.11 docker.io/wangxiaochun/alpine:3.11-v1
[root@ubuntu1804 ~]#docker images
REPOSITORY              TAG                 IMAGE ID            CREATED
    SIZE
haproxy-centos7         2.1.2               5eccdb29a058        4 hours ago
    428MB
nginx-ubuntu1804        1.16.1              19efdd23ac87        18 hours ago
   378MB
nginx-alpine            1.16.1              978a43bbb61d        19 hours ago
   211MB
alpine-base             3.11                b162eecf4da9        20 hours ago
   182MB
tomcat-web              app2                0e1760fe79a6        40 hours ago
   838MB
tomcat-web              app1                76016219a0ca        40 hours ago
   838MB
tomcat-base             v8.5.50             8d5395cb72c4        41 hours ago
   824MB
centos7-jdk             8u212               e0fe770a7ccd        42 hours ago
   809MB
```

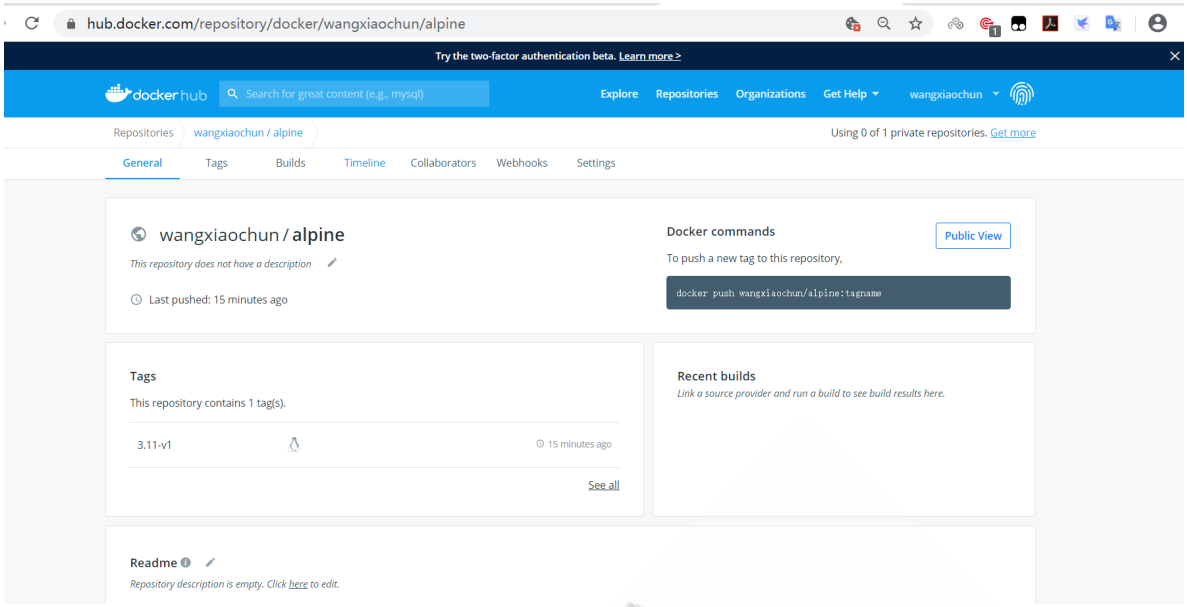
```
centos7-base                v1                  34ab3afcd3b3        43 hours ago
   403MB
alpine                      3.11                e7d92cdc71fe        12 days ago
    5.59MB
alpine                      latest              e7d92cdc71fe        12 days ago
    5.59MB
wangxiaochun/alpine         3.11-v1             e7d92cdc71fe        12 days ago
    5.59MB
ubuntu                      18.04               ccc6e87d482b        2 weeks ago
    64.2MB
ubuntu                      bionic              ccc6e87d482b        2 weeks ago
    64.2MB
centos                      centos7.7.1908      08d05d1d5859        2 months ago
   204MB
```

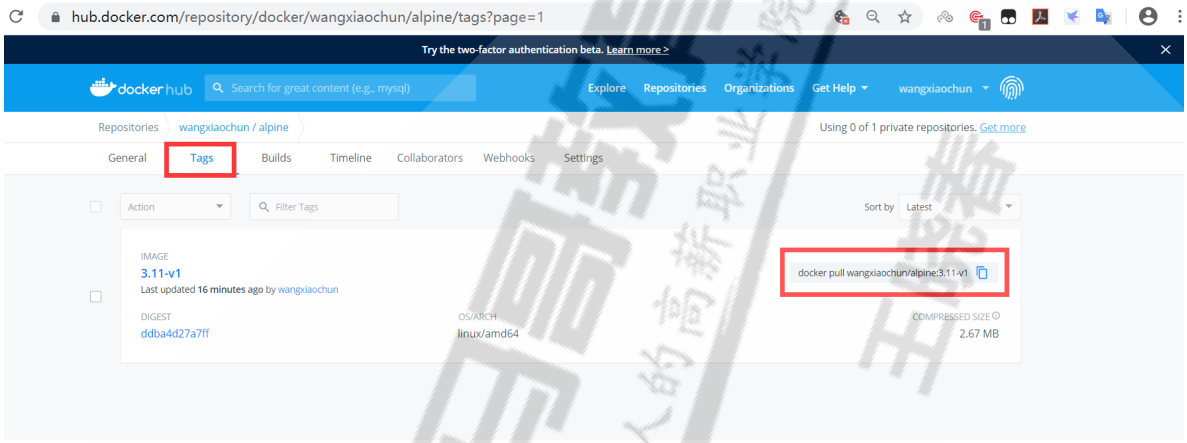
5.1.2.3 上传本地镜像至官网

```
#如tag省略,将上传指定REPOSITORY的所有版本,如下示例
#[root@ubuntu1804 ~]#docker push docker.io/wangxiaochun/alpine

[root@ubuntu1804 ~]#docker push docker.io/wangxiaochun/alpine:3.11-v1
The push refers to repository [docker.io/wangxiaochun/alpine]
5216338b40a7: Mounted from wanglinux/alpine-base
3.11-v1: digest:
sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45 size:
528
```

5.1.2.4 在官网验证上传的镜像







5.1.2.5 下载上传的镜像并创建容器

在另一台主机上下载镜像

```
[root@centos7 ~]#docker pull wangxiaochun/alpine:3.11-v1
3.11-v1: Pulling from wangxiaochun/alpine
c9b1b535fdd9: Already exists
Digest: sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45
Status: Downloaded newer image for wangxiaochun/alpine:3.11-v1
docker.io/wangxiaochun/alpine:3.11-v1
[root@centos7 ~]#docker images
REPOSITORY              TAG                 IMAGE ID            CREATED
    SIZE
wangxiaochun/alpine     3.11-v1             e7d92cdc71fe        12 days ago
    5.59MB
[root@centos7 ~]#docker run -it --rm wangxiaochun/alpine:3.11-v1 sh
/ # cat /etc/issue
Welcome to Alpine Linux 3.11
Kernel \r on an \m (\l)
/ # du -sh /
5.6M    /
/ # exit
[root@centos7 ~]#
```

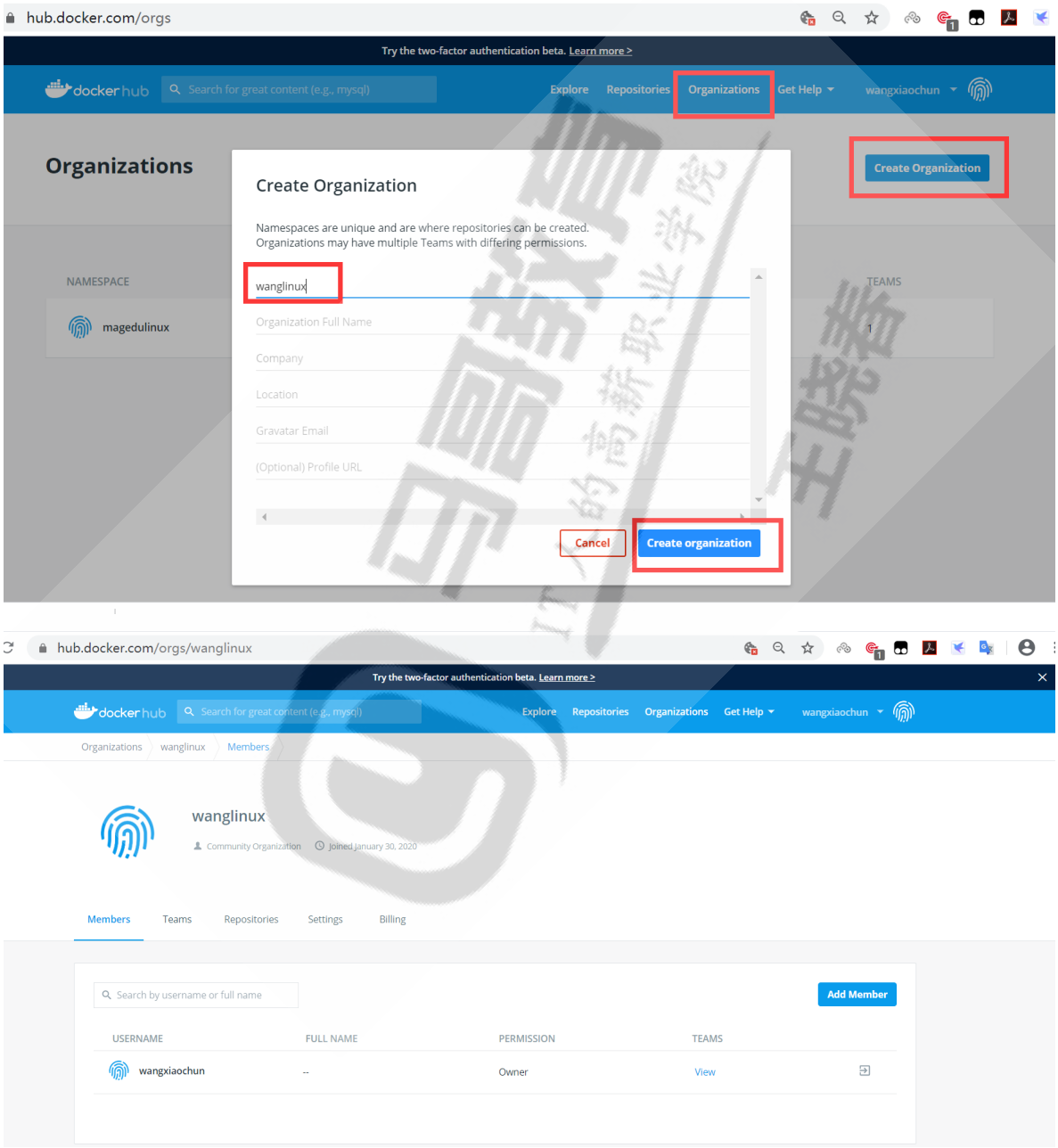
5.1.3 使用组织管理镜像

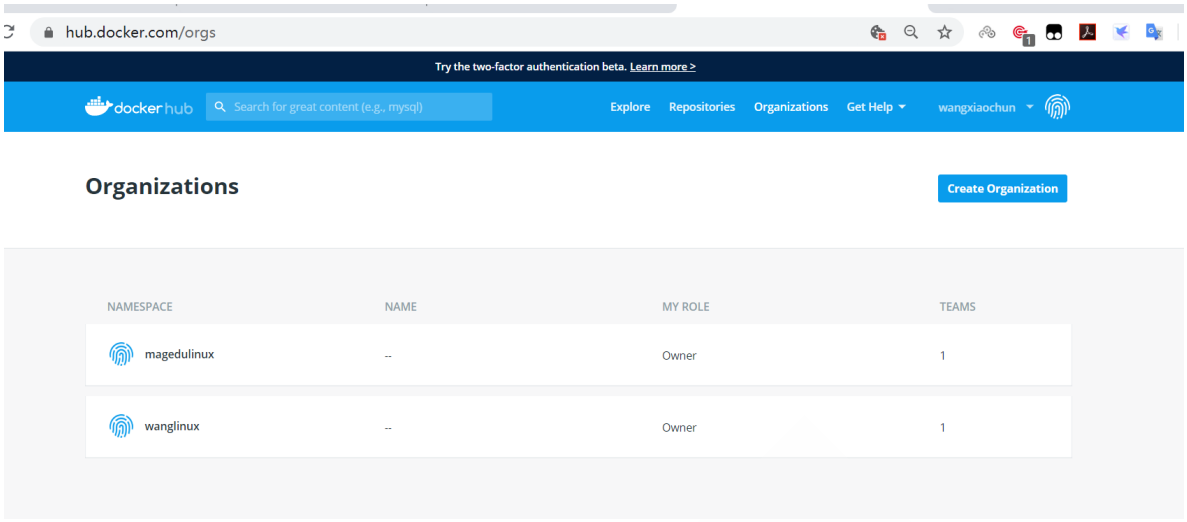
组织类似于名称空间，每个组织的名称全网站唯一，一个组织可以有多个用户帐户使用，并且可以指定不同用户对组织内的仓库不同的权限

三种不同权限

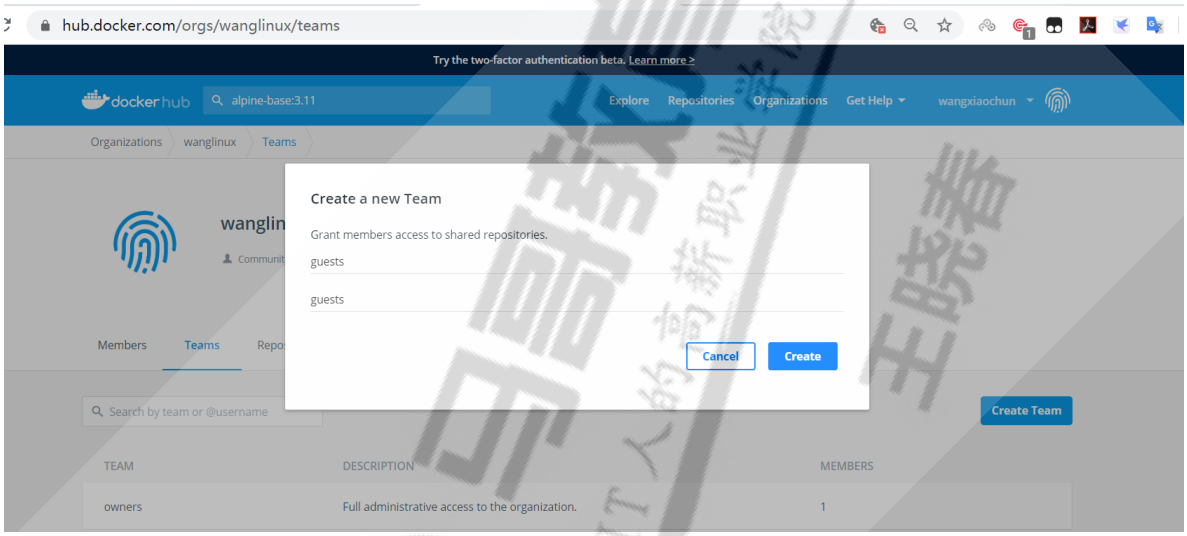
- Read-only: Pull and view repository details and builds
- Read &Write: Pull, push, and view a repository; view, cancel, retry or trigger builds
- Admin: Pull, push, view, edit, and delete a repository; edit build settings; update the repository description

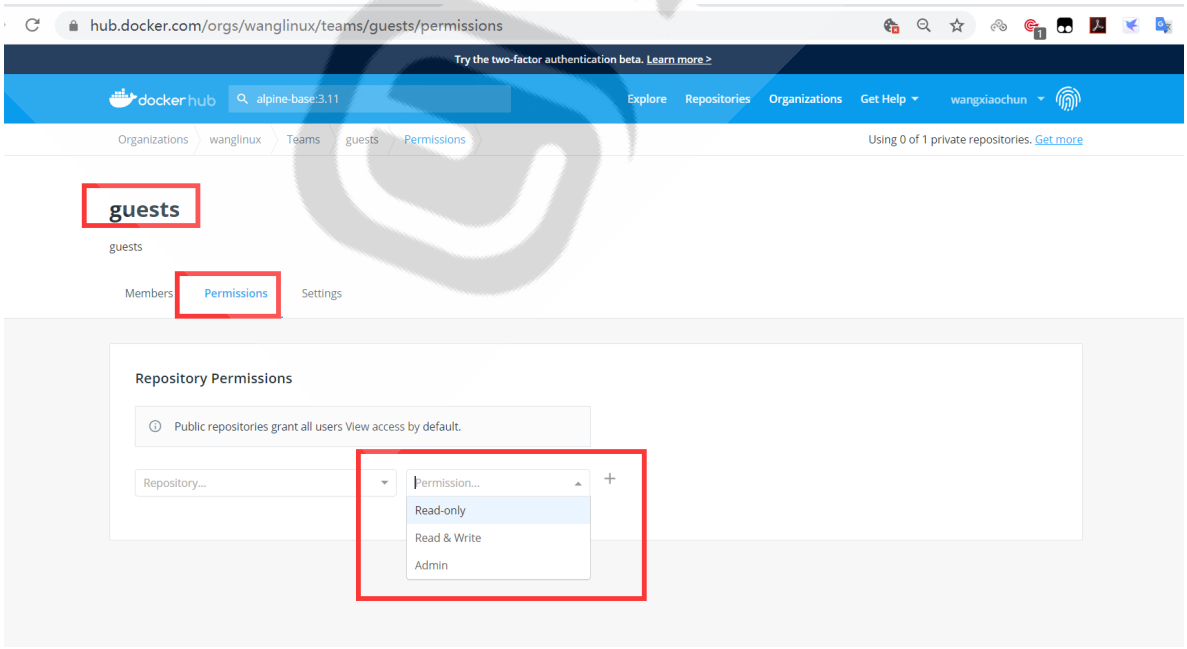
5.1.3.1 创建组织

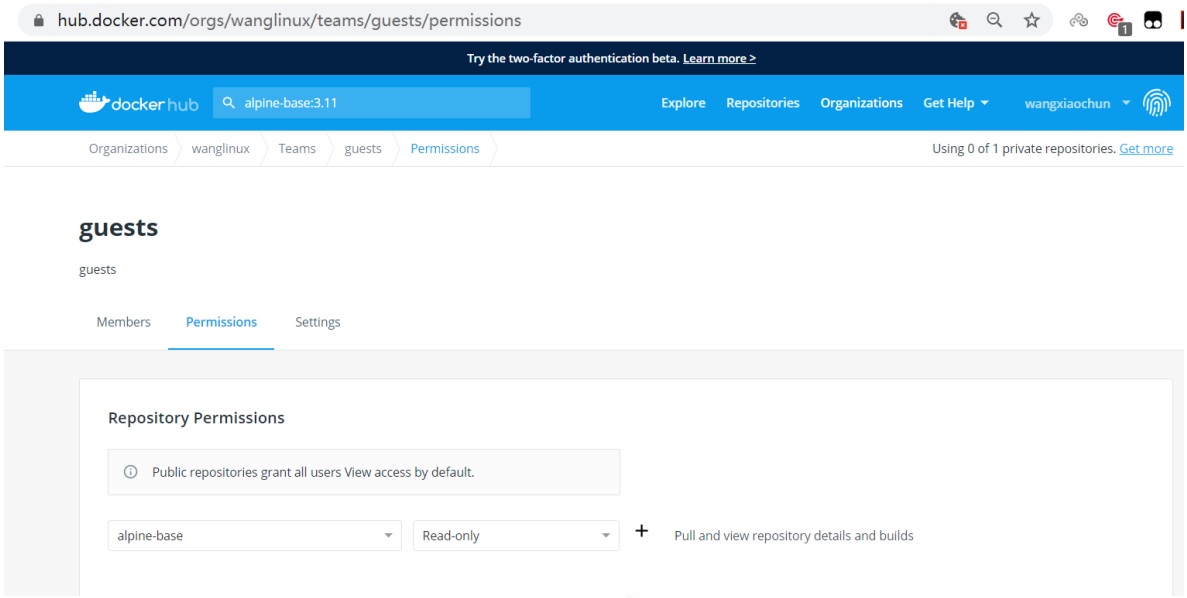




5.1.3.2 创建组织内的团队，并分配权限







5.1.3.3 上传镜像前登录帐号

```
[root@ubuntu1804 ~]#docker login docker.io
Login with your Docker ID to push and pull images from Docker Hub. If you don't
have a Docker ID, head over to https://hub.docker.com to create one.
Username: wangxiaochun
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ubuntu1804 ~]#cat .docker/config.json
{
    "auths": {
        "https://index.docker.io/v1/": {
            "auth": "d2FuZ3hpYW9jaHVuOmxidG9vdGgwNjE4"
        }
    },
    "HttpHeaders": {
        "User-Agent": "Docker-Client/19.03.5 (linux)"
    }
}[root@ubuntu1804 ~]#
```

5.1.3.4 给本地镜像打标签

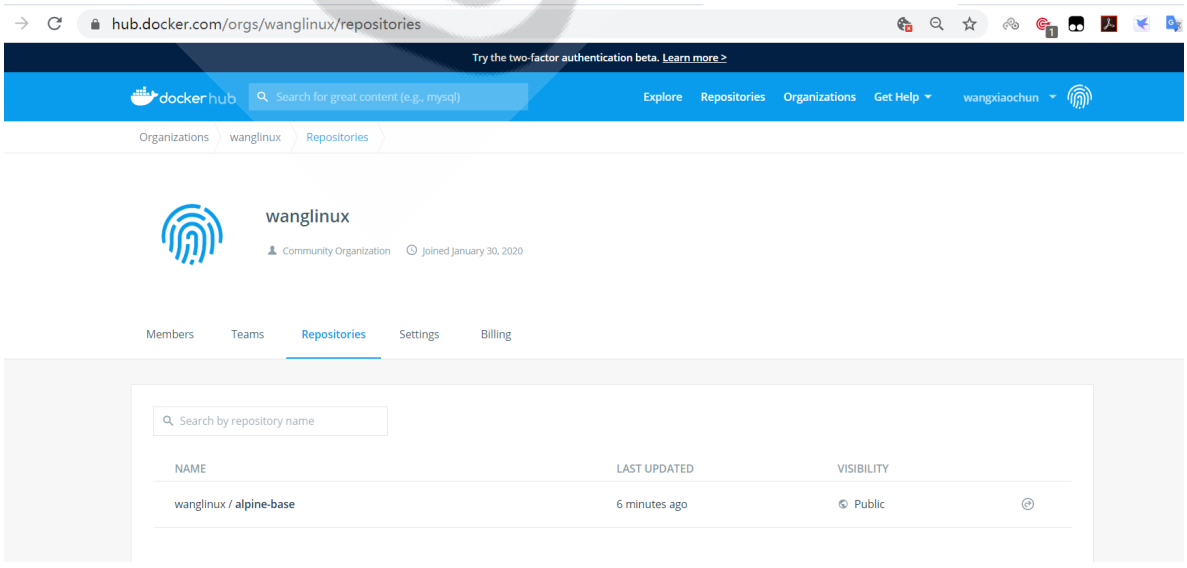
```
[root@ubuntu1804 ~]#docker tag alpine-base:3.11 docker.io/wanglinux/alpine-
base:3.11
[root@ubuntu1804 ~]#docker images
REPOSITORY              TAG                 IMAGE ID            CREATED
    SIZE
haproxy-centos7         2.1.2               5eccdb29a058        4 hours ago
    428MB
nginx-ubuntu1804        1.16.1              19efdd23ac87        18 hours ago
   378MB
nginx-alpine            1.16.1              978a43bbb61d        19 hours ago
   211MB
alpine-base             3.11                b162eecf4da9        20 hours ago
   182MB
```

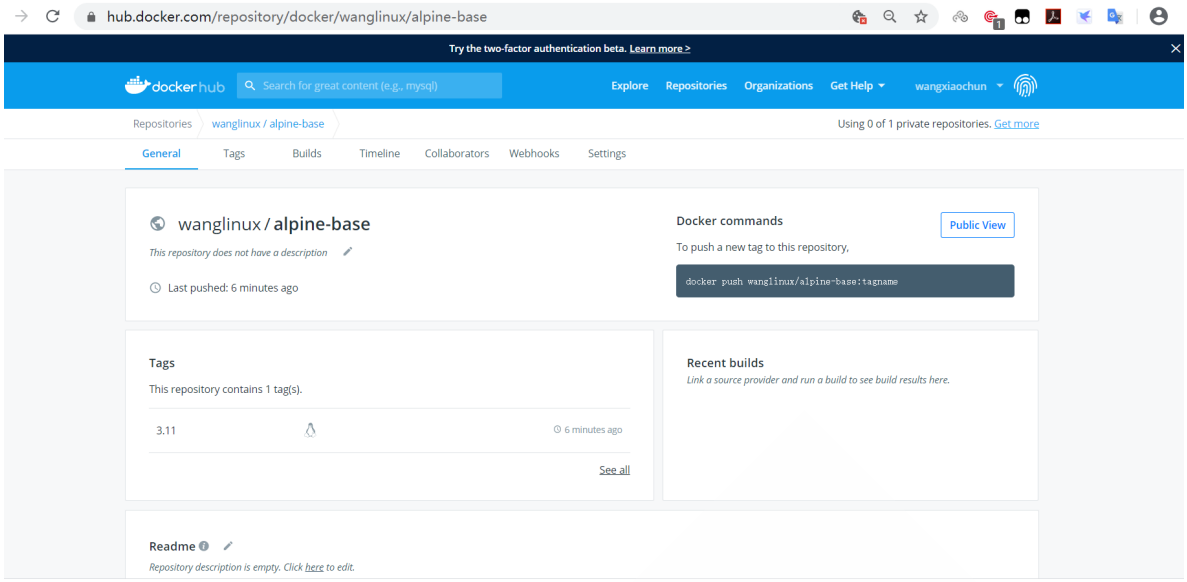
```
wanglinux/alpine-base    3.11                b162eecf4da9        20 hours ago
    182MB
tomcat-web               app2                0e1760fe79a6        40 hours ago
    838MB
tomcat-web               app1                76016219a0ca        40 hours ago
    838MB
tomcat-base              v8.5.50             8d5395cb72c4        41 hours ago
    824MB
centos7-jdk              8u212               e0fe770a7ccd        42 hours ago
    809MB
centos7-base             v1                  34ab3afcd3b3        43 hours ago
    403MB
alpine                   3.11                e7d92cdc71fe        12 days ago
     5.59MB
alpine                   latest              e7d92cdc71fe        12 days ago
     5.59MB
wangxiaochun/alpine      3.11-v1             e7d92cdc71fe        12 days ago
     5.59MB
ubuntu                   18.04               ccc6e87d482b        2 weeks ago
     64.2MB
ubuntu                   bionic              ccc6e87d482b        2 weeks ago
     64.2MB
centos                   centos7.7.1908      08d05d1d5859        2 months ago
    204MB
```

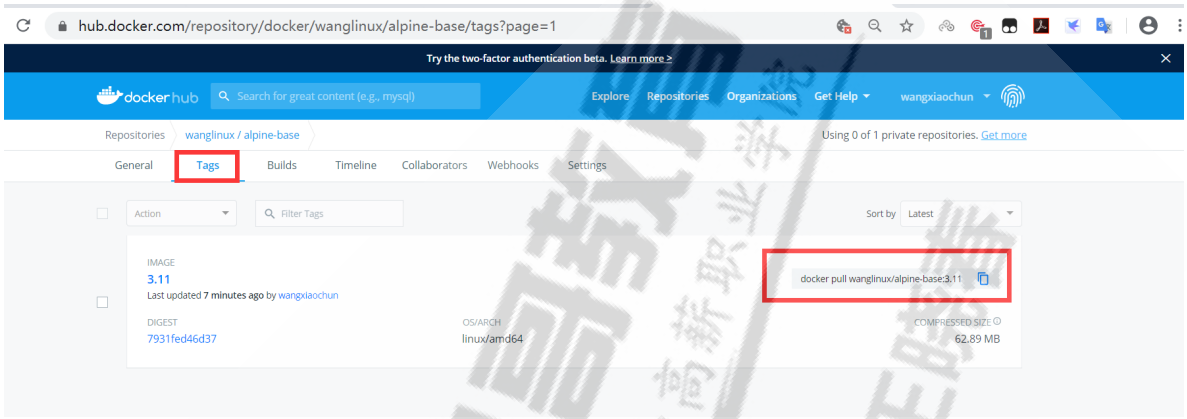
5.1.3.5 上传镜像到指定的组织

```
[root@ubuntu1804 ~]#docker push docker.io/wanglinux/alpine-base:3.11
The push refers to repository [docker.io/wanglinux/alpine-base]
1783f0912dfb: Pushed
1e5e8d5ab333: Pushed
5216338b40a7: Mounted from library/alpine
3.11: digest:
sha256:7931fed46d377698dacb194d46017c53bc24f2e9ee41e893e6900c07d1153536 size:
947
```

5.1.3.6 在网站看查看上传的镜像







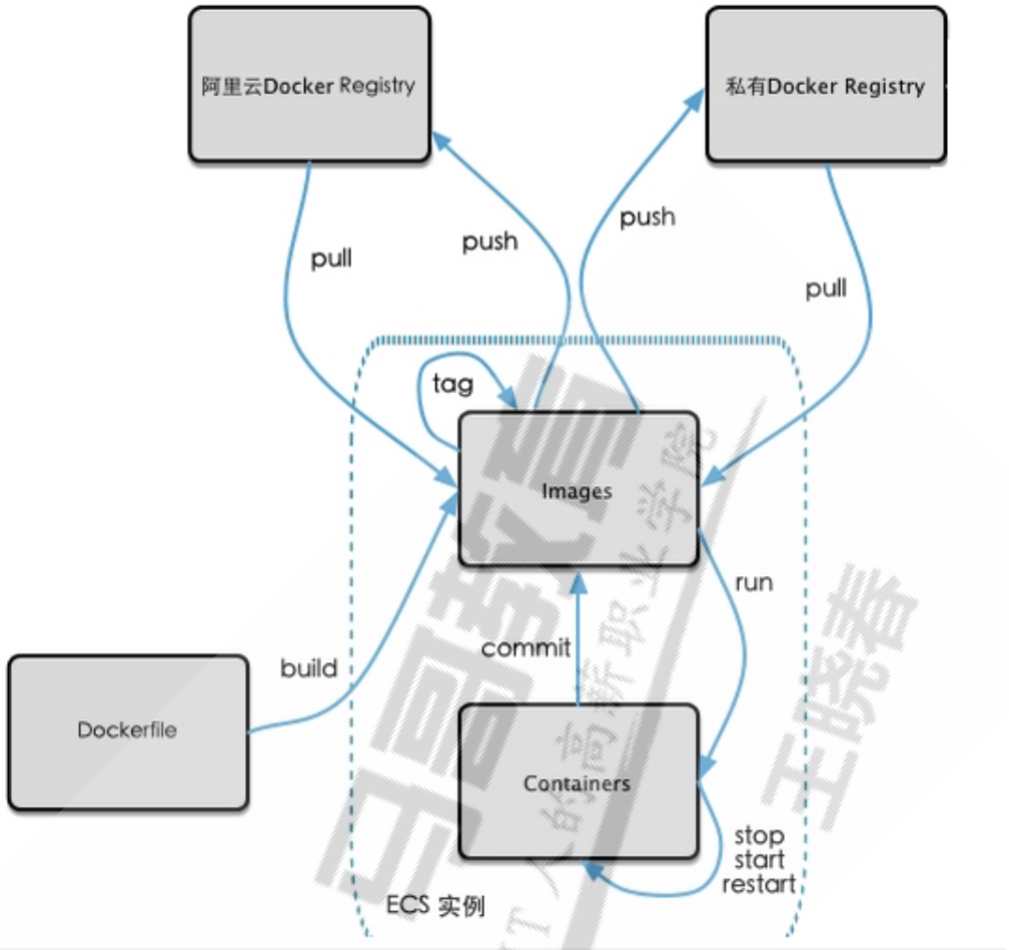
5.1.3.7 下载上传的镜像并运行容器

在另一台主机上下载镜像

```
[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@centos7 ~]#docker pull wanglinux/alpine-base:3.11
3.11: Pulling from wanglinux/alpine-base
c9b1b535fdd9: Pull complete
327af1e87fd8: Pull complete
d88818b49372: Pull complete
Digest: sha256:7931fed46d377698dacb194d46017c53bc24f2e9ee41e893e6900c07d1153536
Status: Downloaded newer image for wanglinux/alpine-base:3.11
docker.io/wanglinux/alpine-base:3.11
[root@centos7 ~]#docker images
REPOSITORY              TAG                 IMAGE ID            CREATED
    SIZE
wanglinux/alpine-base   3.11                b162eecf4da9        20 hours ago
   182MB

[root@centos7 ~]#docker run -it --rm wanglinux/alpine-base:3.11 sh
/ # cat /etc/issue
Welcome to Alpine Linux 3.11
Kernel \r on an \m (\l)
/ # du -sh /
190.1M   /
/ # exit
```

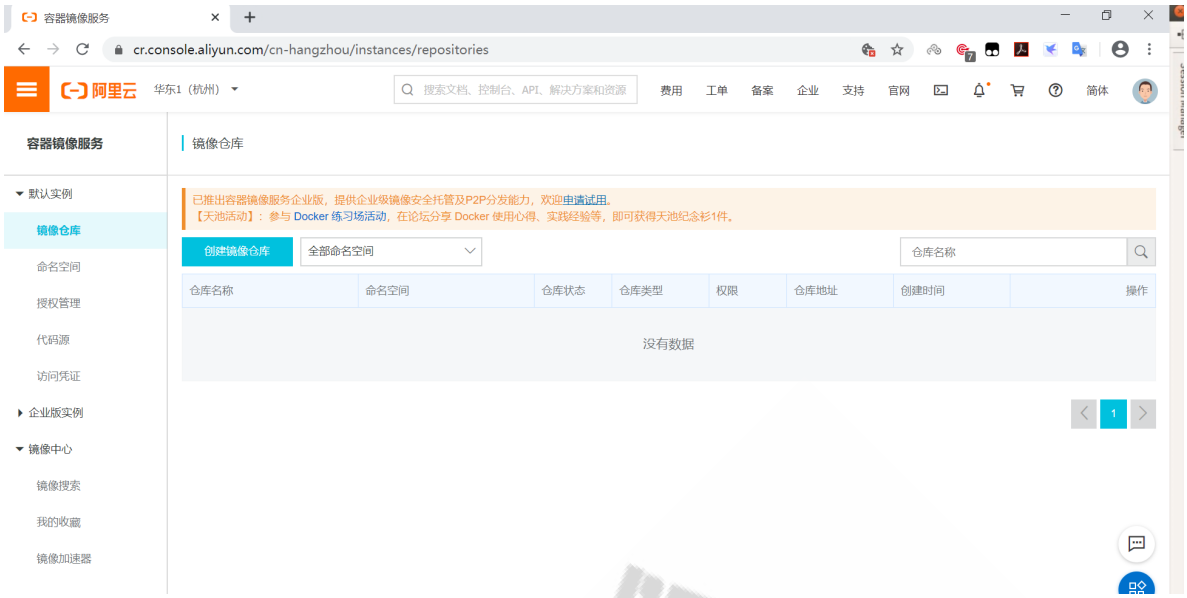
```
[root@centos7 ~]#
```

5.2 阿里云Docker仓库

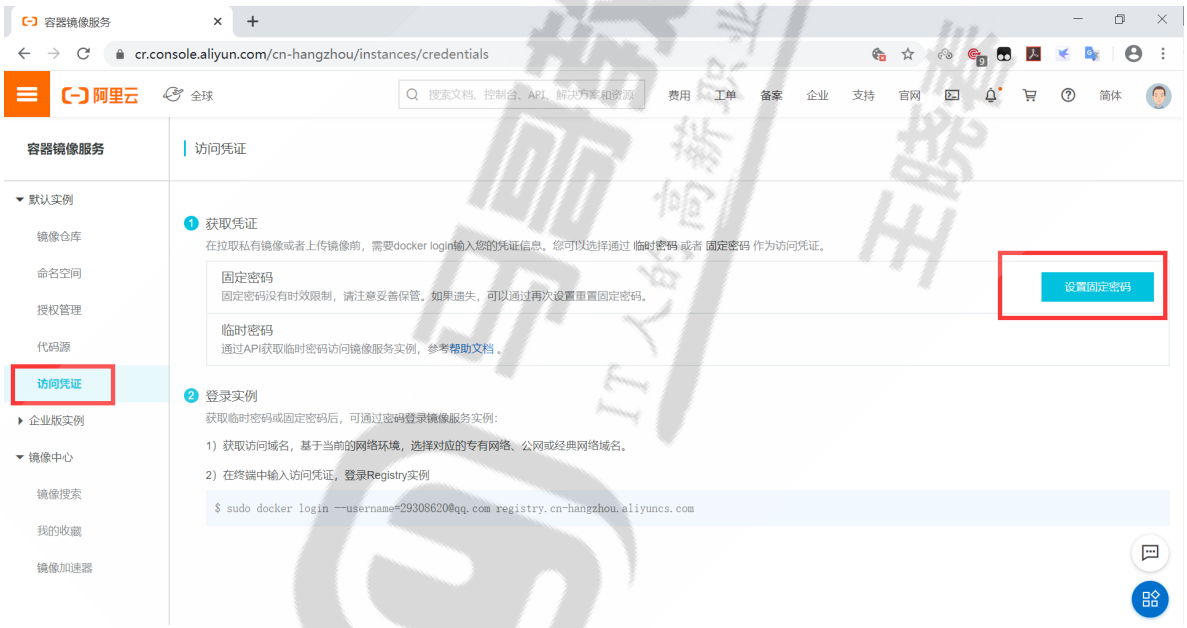


5.2.1 注册和登录阿里云仓库

用浏览器访问http://cr.console.aliyun.com，输入注册的用户信息登录网站

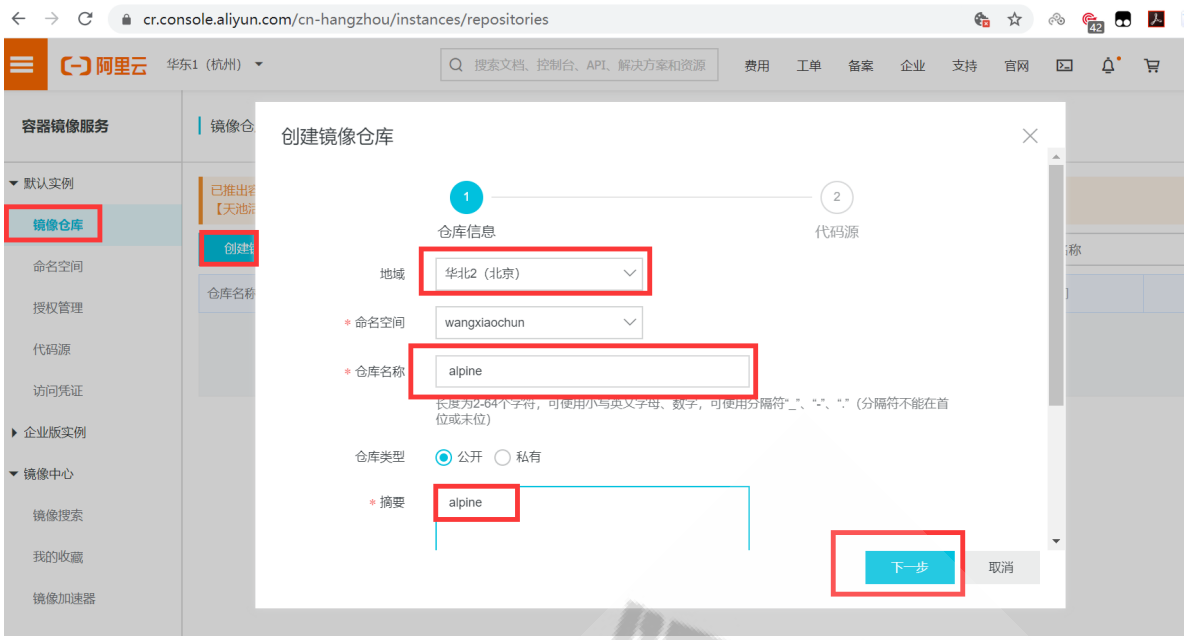


5.2.2 设置仓库专用管理密码

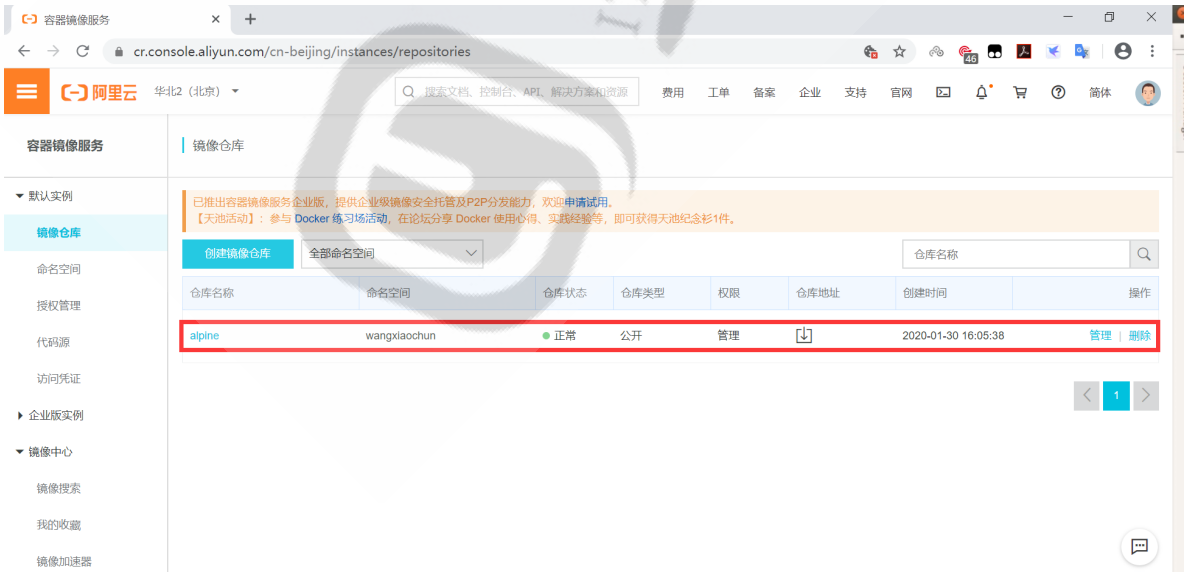


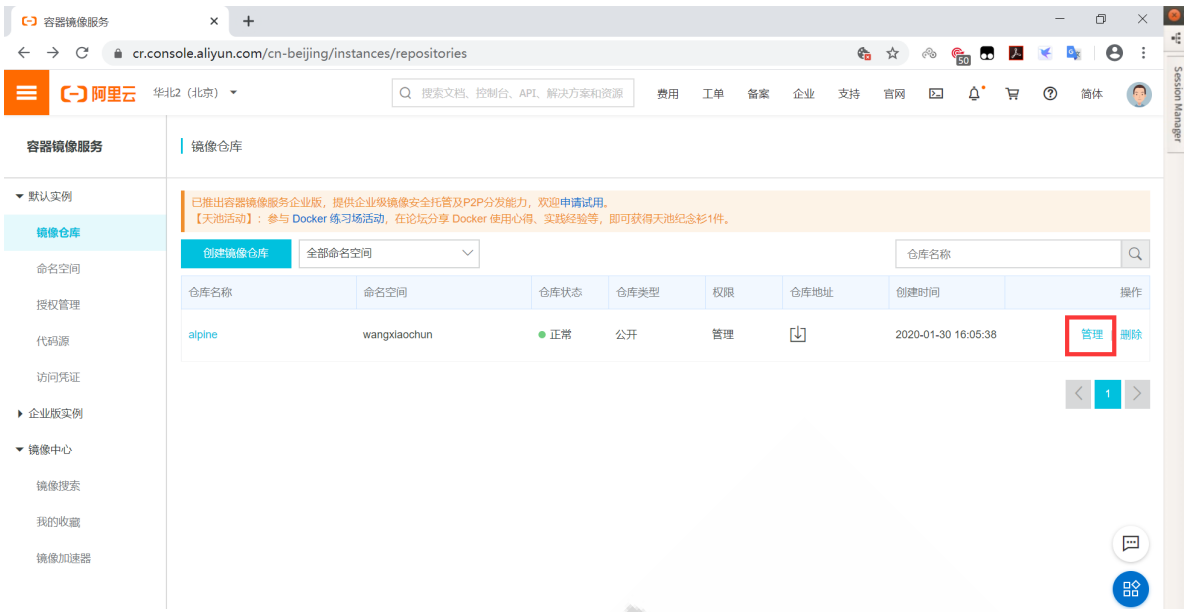
5.2.3 创建仓库

此步可不事先执行，docker push 时可以自动创建私有仓库

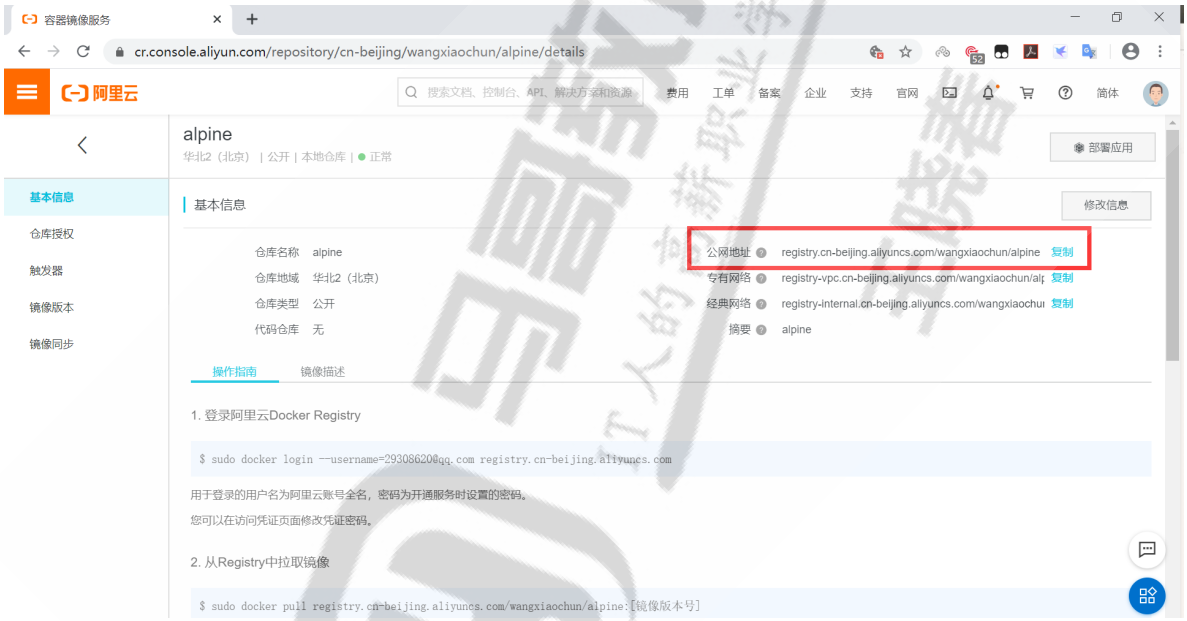

cr.console.aliyun.com/cn-hangzhou/instances/repositories
阿里云
华东1（杭州）
搜索文档、控制台、API、解决方案和资源
费用
工单
备案
企业
支持
官网
容器镜像服务
镜像仓
默认实例
镜像仓库
命名空间
授权管理
代码源
访问凭证
企业版实例
镜像中心
镜像搜索
我的收藏
镜像加速器
创建镜像仓库
1
仓库信息
2
代码源
地域
华北2（北京）
* 命名空间
wangxiaochun
* 仓库名称
alpine
长度为2-64个字符，可使用小写英文字母、数字，可使用分隔符"_"、"-"、"."（分隔符不能在首位或末位）
仓库类型
公开
私有
* 摘要
alpine
下一步
取消


创建镜像仓库
仓库信息
2
代码源
代码源
云Code
GitHub
Bitbucket
私有GitLab
本地仓库
您可以通过命令行推送镜像到镜像仓库。
上一步
创建镜像仓库
取消


容器镜像服务
cr.console.aliyun.com/cn-beijing/instances/repositories
阿里云
华北2（北京）
搜索文档、控制台、API、解决方案和资源
费用
工单
备案
企业
支持
官网
简体
容器镜像服务
镜像仓库
默认实例
镜像仓库
命名空间
授权管理
代码源
访问凭证
企业版实例
镜像中心
镜像搜索
我的收藏
镜像加速器
已推出容器镜像服务企业版，提供企业级镜像安全托管及P2P分发能力，欢迎申请试用。
【天池活动】：参与 Docker 练习场活动，在论坛分享 Docker 使用心得、实践经验等，即可获得天池纪念衫1件。
创建镜像仓库
全部命名空间
仓库名称
仓库名称
命名空间
仓库状态
仓库类型
权限
仓库地址
创建时间
操作
alpine
wangxiaochun
正常
公开
管理
2020-01-30 16:05:38
管理
删除
1



查看仓库的路径用于上传镜像使用



5.2.4 上传镜像前先登录阿里云

```
#用前面设置的专用仓库管理密码登录
[root@ubuntu1804 ~]#docker login --username=29308620@qq.com registry.cn-beijing.aliyuncs.com
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
#登录密码保存在~/.docker/config.json文件中，下次将不会需要再输入密码登录
[root@ubuntu1804 ~]#cat .docker/config.json
{
    "auths": {
        "https://index.docker.io/v1/": {
            "auth": "d2FuZ3hpYW9jaHVuOmxidG9vdGgwNjE4"
```

```
        },
        "registry.cn-beijing.aliyuncs.com": {
            "auth": "MjkzMDg2MjBAcXEuY29tOmxidG9vdGgwNjE4"
        }
    },
    "HttpHeaders": {
        "User-Agent": "Docker-Client/19.03.5 (linux)"
    }
}[root@ubuntu1804 ~]#
```

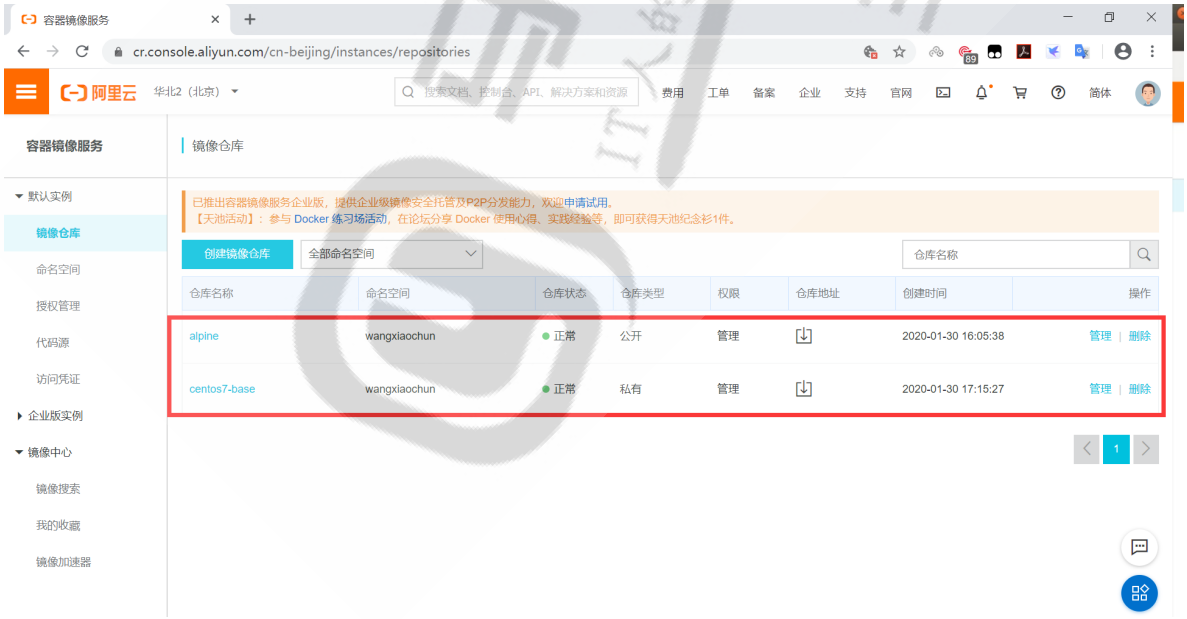
5.2.5 给上传的镜像打标签

```
[root@ubuntu1804 ~]#docker tag alpine-base:3.11 registry.cn-
beijing.aliyuncs.com/wangxiaochun/alpine:3.11-v1
[root@ubuntu1804 ~]#docker tag centos7-base:v1 registry.cn-
beijing.aliyuncs.com/wangxiaochun/centos7-base:v1
[root@ubuntu1804 ~]#docker images
REPOSITORY                                                      TAG
IMAGE ID            CREATED             SIZE
haproxy-centos7                                                 2.1.2
5eccdb29a058        6 hours ago         428MB
nginx-ubuntu1804                                                1.16.1
19efdd23ac87        21 hours ago        378MB
nginx-alpine                                                    1.16.1
978a43bbb61d        22 hours ago        211MB
alpine-base                                                     3.11
b162eecf4da9        23 hours ago        182MB
wanglinux/alpine-base                                           3.11
b162eecf4da9        23 hours ago        182MB
registry.cn-beijing.aliyuncs.com/wangxiaochun/alpine            3.11-v1
b162eecf4da9        23 hours ago        182MB
tomcat-web                                                      app2
0e1760fe79a6        43 hours ago        838MB
tomcat-web                                                      app1
76016219a0ca        43 hours ago        838MB
tomcat-base                                                     v8.5.50
8d5395cb72c4        44 hours ago        824MB
centos7-jdk                                                     8u212
e0fe770a7ccd        45 hours ago        809MB
centos7-base                                                    v1
34ab3afcd3b3        46 hours ago        403MB
registry.cn-beijing.aliyuncs.com/wangxiaochun/centos7-base      v1
34ab3afcd3b3        46 hours ago        403MB
alpine                                                          3.11
e7d92cdc71fe        12 days ago         5.59MB
alpine                                                          latest
e7d92cdc71fe        12 days ago         5.59MB
wangxiaochun/alpine                                             3.11-v1
e7d92cdc71fe        12 days ago         5.59MB
ubuntu                                                          18.04
ccc6e87d482b        2 weeks ago         64.2MB
ubuntu                                                          bionic
ccc6e87d482b        2 weeks ago         64.2MB
centos                                                          centos7.7.1908
08d05d1d5859        2 months ago        204MB
```

5.2.6 上传镜像至阿里云

```
[root@ubuntu1804 ~]# docker push registry.cn-
beijing.aliyuncs.com/wangxiaochun/alpine:3.11-v1
The push refers to repository [registry.cn-
beijing.aliyuncs.com/wangxiaochun/alpine]
1783f0912dfb: Pushed
1e5e8d5ab333: Pushed
5216338b40a7: Pushed
3.11-v1: digest:
sha256:7931fed46d377698dacb194d46017c53bc24f2e9ee41e893e6900c07d1153536 size:
947

[root@ubuntu1804 ~]#docker push registry.cn-
beijing.aliyuncs.com/wangxiaochun/centos7-base:v1
The push refers to repository [registry.cn-
beijing.aliyuncs.com/wangxiaochun/centos7-base]
2073413aebd6: Pushed
6ec9af97c369: Pushed
034f282942cd: Pushed
v1: digest:
sha256:02cd943f2569c7c55f08a979fd9661f1fd7893c424bca7b343188654ba63d98d size:
949
```

5.2.7 在网站查看上传的镜像



5.2.8 从另一台主机上下载刚上传的镜像并运行容器

```
[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@centos7 ~]# docker pull registry.cn-
beijing.aliyuncs.com/wangxiaochun/alpine:3.11-v1
3.11-v1: Pulling from wangxiaochun/alpine
c9b1b535fdd9: Pull complete
327af1e87fd8: Pull complete
```

```
d88818b49372: Pull complete
Digest: sha256:7931fed46d377698dacb194d46017c53bc24f2e9ee41e893e6900c07d1153536
Status: Downloaded newer image for registry.cn-
beijing.aliyuncs.com/wangxiaochun/alpine:3.11-v1
registry.cn-beijing.aliyuncs.com/wangxiaochun/alpine:3.11-v1
[root@centos7 ~]#docker images
REPOSITORY                                                TAG                 IMAGE
ID            CREATED             SIZE
registry.cn-beijing.aliyuncs.com/wangxiaochun/alpine   3.11-v1
b162eecf4da9        22 hours ago        182MB
[root@centos7 ~]#docker run -it --rm b162eecf4da9 sh
/ # cat /etc/issue
Welcome to Alpine Linux 3.11
Kernel \r on an \m (\l)

/ # du -sh /
190.1M  /
/ # exit
[root@centos7 ~]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@centos7 ~]#

#上传的centos7-base:v1为私有镜像，需要登录才能下载
[root@centos7 ~]#docker pull registry.cn-
beijing.aliyuncs.com/wangxiaochun/centos7-base:v1
Error response from daemon: pull access denied for registry.cn-
beijing.aliyuncs.com/wangxiaochun/centos7-base, repository does not exist or may
require 'docker login': denied: requested access to the resource is denied
[root@centos7 ~]#docker login registry.cn-beijing.aliyuncs.com
Username: lbtooth
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@centos7 ~]#cat .docker/config.json
{
    "auths": {
        "registry.cn-beijing.aliyuncs.com": {
            "auth": "bGJ0b290aDpsYnRvb3RoMDYxOA=="
        }
    },
    "HttpHeaders": {
        "User-Agent": "Docker-Client/19.03.5 (linux)"
    }
}[root@centos7 ~]#

[root@centos7 ~]#docker pull registry.cn-
beijing.aliyuncs.com/wangxiaochun/centos7-base:v1
v1: Pulling from wangxiaochun/centos7-base
f34b00c7da20: Pull complete
544476d462f7: Pull complete
39345915aa1b: Pull complete
Digest: sha256:02cd943f2569c7c55f08a979fd9661f1fd7893c424bca7b343188654ba63d98d
Status: Downloaded newer image for registry.cn-
beijing.aliyuncs.com/wangxiaochun/centos7-base:v1
```

```
registry.cn-beijing.aliyuncs.com/wangxiaochun/centos7-base:v1
[root@centos7 ~]#docker images
REPOSITORY                                                      TAG
IMAGE ID            CREATED             SIZE
registry.cn-beijing.aliyuncs.com/wangxiaochun/alpine            3.11-v1
b162eecf4da9        23 hours ago        182MB
registry.cn-beijing.aliyuncs.com/wangxiaochun/centos7-base      v1
34ab3afcd3b3        46 hours ago        403MB
[root@centos7 ~]#
```

5.3 私有云单机仓库Docker Registry

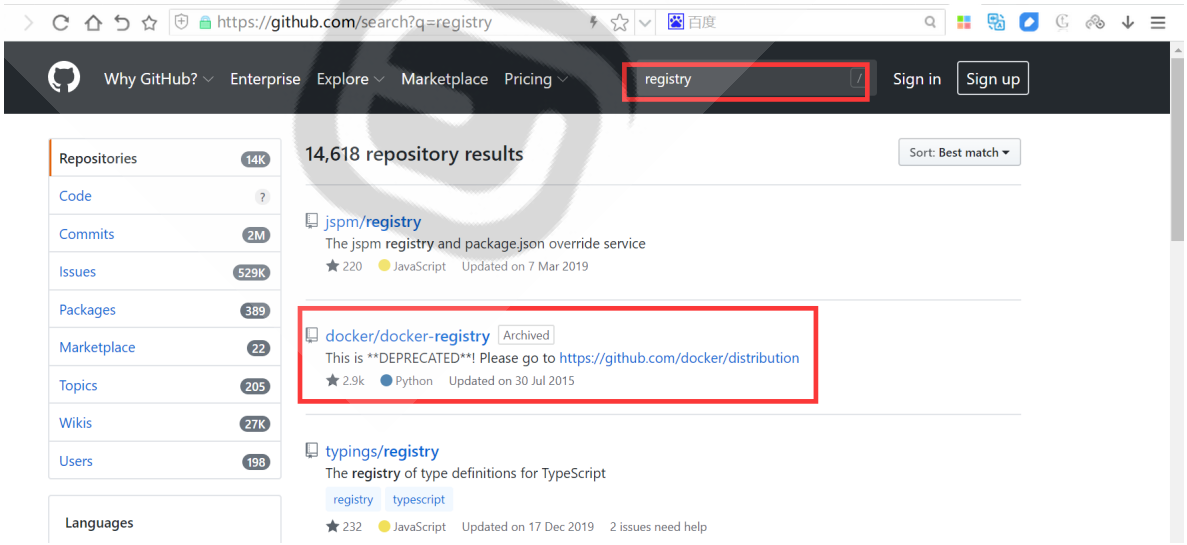
5.3.1 Docker Registry 介绍

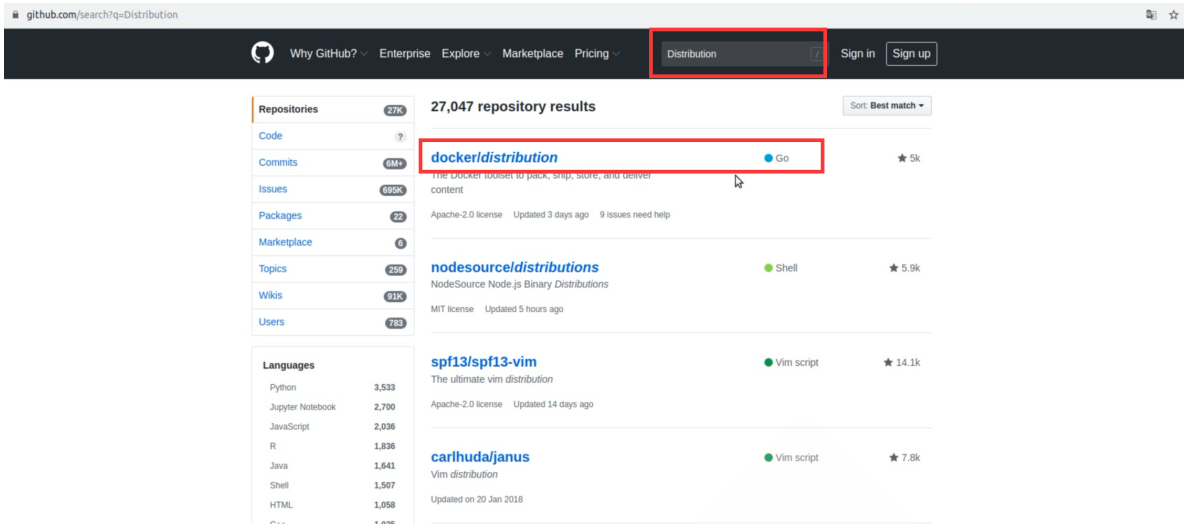
Docker Registry作为Docker的核心组件之一负责单主机的镜像内容的存储与分发，客户端的docker pull以及push命令都将直接与registry进行交互,最初版本的registry 由Python实现，由于设计初期在安全性，性能以及API的设计上有着诸多的缺陷，该版本在0.9之后停止了开发，由新项目distribution（新的docker register被称为Distribution）来重新设计并开发下一代registry，新的项目由go语言开发，所有的API，底层存储方式，系统架构都进行了全面的重新设计已解决上一代registry中存在的问题，2016年4月份registry 2.0正式发布，docker 1.6版本开始支持registry 2.0，而八月份随着docker 1.8 发布，docker hub正式启用2.1版本registry全面替代之前版本 registry，新版registry对镜像存储格式进行了重新设计并和旧版不兼容，docker 1.5和之前的版本无法读取2.0的镜像，另外，Registry 2.4版本之后支持了回收站机制，也就是可以删除镜像了，在2.4版本之前是无法支持删除镜像的，所以如果你要使用最好是大于Registry 2.4版本的

官方文档地址: https://docs.docker.com/registry/

官方github 地址: https://github.com/docker/distribution

官方部署文档: https://github.com/docker/docker.github.io/blob/master/registry/deploying.md





以下介绍通过官方提供的docker registry 镜像来简单搭建本地私有仓库环境

环境：三台主机

10.0.0.100：充当registry仓库服务器

10.0.0.101：上传镜像

10.0.0.102：下载镜像

5.3.2 下载 docker registry 镜像

```
[root@ubuntu1804 ~]#docker pull registry:2.7.1
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
registry            2.7.1               708bc6af7e5e        6 days ago
 25.8MB
```

5.3.3 搭建单机仓库

5.3.2.1 创建授权用户密码使用目录

```
[root@ubuntu1804 ~]#mkdir -p  /etc/docker/auth
```

5.3.2.2 创建授权的registry用户和密码

创建registry用户，用于上传和下载镜像

```
[root@ubuntu1804 ~]#apt -y install apache2
[root@ubuntu1804 ~]#htpasswd -Bbn wang 123456 > /etc/docker/auth/registry
[root@ubuntu1804 ~]#cat /etc/docker/auth/registry
wang:$2y$05$nlRIIYEUBTSLdN2PkzodUue4ry7X/UyscpkkEufTDhEdI8nsyJMR6

#[root@ubuntu1804 ~]#docker run --entrypoint htpasswd registry:2.7.1 -Bbn wang
123456 > /etc/docker/auth/registry
```

5.3.2.3 启动docker registry 容器

```
[root@ubuntu1804 ~]# docker run -d -p 5000:5000 --restart=always --name registry
-v /etc/docker/auth:/auth -e "REGISTRY_AUTH=htpasswd" -e
"REGISTRY_AUTH_HTPASSWD_REALM=Registry Realm" -e
REGISTRY_AUTH_HTPASSWD_PATH=/auth/registry registry:2.7.1
998f970dd8ca6b98002f20ae27330fe607ca78f35bedcc8a6180688e48a907a7
```

5.3.2.4 验证端口和容器

```
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
     STATUS              PORTS                    NAMES
998f970dd8ca        registry:2.7.1      "/entrypoint.sh /etc…"   About a minute
ago   Up About a minute   0.0.0.0:5000->5000/tcp   registry

[root@ubuntu1804 ~]#ss -ntl
State    Recv-Q   Send-Q      Local Address:Port     Peer Address:Port
LISTEN   0        64                0.0.0.0:2049          0.0.0.0:*

LISTEN   0        128               0.0.0.0:48131         0.0.0.0:*

LISTEN   0        128               0.0.0.0:33835         0.0.0.0:*

LISTEN   0        128               0.0.0.0:58029         0.0.0.0:*

LISTEN   0        128               0.0.0.0:111           0.0.0.0:*

LISTEN   0        128         127.0.0.53%lo:53            0.0.0.0:*

LISTEN   0        128               0.0.0.0:22            0.0.0.0:*

LISTEN   0        64                0.0.0.0:46429         0.0.0.0:*

LISTEN   0        128             127.0.0.1:6014          0.0.0.0:*

LISTEN   0        64                   [::]:2049             [::]:*

LISTEN   0        128                   *:5000                *:*

LISTEN   0        128                  [::]:39471            [::]:*

LISTEN   0        128                  [::]:111              [::]:*

LISTEN   0        64                   [::]:43601            [::]:*

LISTEN   0        128                  [::]:56725            [::]:*

LISTEN   0        128                  [::]:22               [::]:*

LISTEN   0        128                  [::]:57881            [::]:*

LISTEN   0        128                 [::1]:6014             [::]:*
```

5.3.4 登录仓库

5.3.4.1 直接登录报错

```
#docker login 默认使用https登录,而docker registry为http,所以默认登录失败
[root@ubuntu1804 ~]#docker login 10.0.0.100:500
Username: wang
Password:
Error response from daemon: Get https://10.0.0.100:500/v2/: dial tcp
10.0.0.100:500: connect: connection refused
```

5.3.4.2 将registry仓库服务器地址加入service 单元文件

```
#修改配置让docker login支持http协议
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
[root@ubuntu1804 ~]#grep ExecStart /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
--insecure-registry 10.0.0.100:5000
#或者修改下面文件
[root@ubuntu1804 ~]#vim  /etc/docker/daemon.json
{
  "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"],
  "insecure-registry": ["10.0.0.100:5000"]
}


[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#ps aux|grep dockerd
root       2092  1.3  8.4 757088 83056 ?        Ssl  19:19   0:00
/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --
insecure-registry 10.0.0.100:5000
root       2233  0.0  0.1  14428  1012 pts/0    S+   19:20   0:00 grep --
color=auto dockerd
```

5.3.4.3 再次登录验证成功

在10.0.0.101主机上执行下面登录

```
[root@ubuntu1804 ~]#docker login 10.0.0.100:5000
Username: wang
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ubuntu1804 ~]#
```

5.3.5 打标签并上传镜像

在10.0.0.101主机上执行打标签上传

```
[root@ubuntu1804 ~]#docker tag centos7-base:v1 10.0.0.100:5000/centos7-base:v1
[root@ubuntu1804 ~]#docker push 10.0.0.100:5000/centos7-base:v1
The push refers to repository [10.0.0.100:5000/centos7-base]
2073413aebd6: Pushed
6ec9af97c369: Pushed
034f282942cd: Pushed
v1: digest:
sha256:02cd943f2569c7c55f08a979fd9661f1fd7893c424bca7b343188654ba63d98d size:
949
```

5.3.6 下载镜像并启动容器

在10.0.0.102主机上下载镜像并启动容器

5.3.6.1 先修改docker的service 文件

```
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
[root@ubuntu1804 ~]#grep ExecStart /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
--insecure-registry 10.0.0.100:5000
[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
```

5.3.6.2 登录registry仓库服务器

```
[root@ubuntu1804 ~]#docker login 10.0.0.100:5000
Username: wang
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

5.3.6.3 下载镜像并启动容器

```
[root@ubuntu1804 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@ubuntu1804 ~]#docker pull  10.0.0.100:5000/centos7-base:v1
v1: Pulling from centos7-base
f34b00c7da20: Pull complete
544476d462f7: Pull complete
39345915aa1b: Pull complete
Digest: sha256:02cd943f2569c7c55f08a979fd9661f1fd7893c424bca7b343188654ba63d98d
Status: Downloaded newer image for 10.0.0.100:5000/centos7-base:v1
10.0.0.100:5000/centos7-base:v1
[root@ubuntu1804 ~]#docker images
REPOSITORY                     TAG                 IMAGE ID            CREATED
          SIZE
10.0.0.100:5000/centos7-base   v1                  34ab3afcd3b3        2 days
ago          403MB
```

```
[root@ubuntu1804 ~]#docker run -it --rm  34ab3afcd3b3 bash
[root@2bcb26b1b568 /]# cat /etc/redhat-release
CentOS Linux release 7.7.1908 (Core)
[root@2bcb26b1b568 /]# exit
exit
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
[root@ubuntu1804 ~]#
```

5.4 Docker 之分布式仓库 Harbor

5.4.1 Harbor 介绍和架构

5.4.1.1 Harbor 介绍

Harbor是一个用于存储和分发Docker镜像的企业级Registry服务器，由VMware开源，其通过添加一些企业必需的功能特性，例如安全、标识和管理等，扩展了开源 Docker Distribution。作为一个企业级私有Registry服务器，Harbor 提供了更好的性能和安全。提升用户使用Registry构建和运行环境传输镜像的效率。Harbor支持安装在多个Registry节点的镜像资源复制，镜像全部保存在私有 Registry 中，确保数据和知识产权在公司内部网络中管控，另外，Harbor也提供了高级的安全特性，诸如用户管理，访问控制和活动审计等

vmware 官方开源服务: https://vmware.github.io/

harbor 官方github 地址: https://github.com/vmware/harbor

harbor 官方网址: https://goharbor.io/

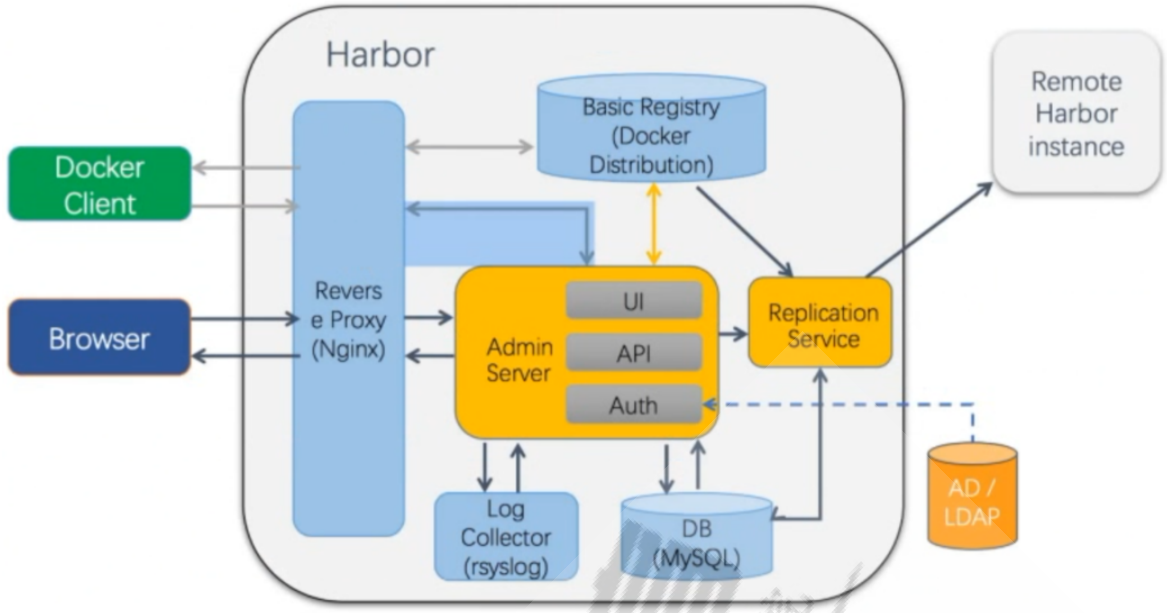
harbor 官方文档: https://goharbor.io/docs/

github文档: https://github.com/goharbor/harbor/tree/master/docs

5.4.1.2 Harbor功能官方介绍

- 基于角色的访问控制: 用户与Docker镜像仓库通过“项目”进行组织管理，一个用户可以对多个镜像仓库在同一命名空间（project）里有不同的权限
- 镜像复制: 镜像可在多个Registry实例中复制（同步）。尤其适合于负载均衡，高可用，混合云和多云的场景
- 图形化用户界面: 用户可以通过浏览器来浏览，检索当前Docker镜像仓库，管理项目和命名空间
- AD/LDAP 支: Harbor可以集成企业内部已有的AD/LDAP，用于鉴权认证管理
- 审计管理: 所有针对镜像仓库的操作都可以被记录追溯，用于审计管理
- 国际化: 已拥有英文、中文、德文、日文和俄文的本地化版本。更多的语言将会添加进来
- RESTful API: 提供给管理员对于Harbor更多的操控, 使得与其它管理软件集成变得更容易
- 部署简单: 提供在线和离线两种安装工具， 也可以安装到vSphere平台(OVA方式)虚拟设备

5.4.1.3 Harbor 组成




```
#harbor是由很多容器组成实现完整功能
[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                                  COMMAND
     CREATED              STATUS                          PORTS
                                                NAMES
4ec3c3885407        goharbor/nginx-photon:v1.7.6           "nginx -g 'daemon
of…"   About a minute ago   Up About a minute (healthy)   0.0.0.0:80->80/tcp,
0.0.0.0:443->443/tcp, 0.0.0.0:4443->4443/tcp   nginx
5707b4ac41d8        goharbor/harbor-portal:v1.7.6          "nginx -g 'daemon
of…"   About a minute ago   Up About a minute (healthy)   80/tcp
                                                 harbor-portal
0ed230b9b714        goharbor/harbor-jobservice:v1.7.6      "/harbor/start.sh"
      About a minute ago   Up About a minute
                                                harbor-jobservice
fec659188349        goharbor/harbor-core:v1.7.6            "/harbor/start.sh"
      About a minute ago   Up About a minute (healthy)
                                                harbor-core
910d14c1d7f7        goharbor/harbor-adminserver:v1.7.6     "/harbor/start.sh"
      2 minutes ago        Up About a minute (healthy)
                                                harbor-adminserver
4348f503aa0e        goharbor/harbor-db:v1.7.6              "/entrypoint.sh
post…"   2 minutes ago        Up About a minute (healthy)   5432/tcp
                                                  harbor-db
beff6886f0f1        goharbor/harbor-registryctl:v1.7.6     "/harbor/start.sh"
      2 minutes ago        Up About a minute (healthy)
                                                registryctl
428c99d274bf        goharbor/registry-photon:v2.6.2-v1.7.6   "/entrypoint.sh
/etc…"   2 minutes ago        Up About a minute (healthy)   5000/tcp
                                                  registry
775b4026fa4e        goharbor/redis-photon:v1.7.6           "docker-
entrypoint.s…"   2 minutes ago        Up About a minute             6379/tcp
                                                    redis
c6f44e2034c6        goharbor/harbor-log:v1.7.6             "/bin/sh -c
/usr/loc…"   2 minutes ago        Up 2 minutes (healthy)
```

- Proxy：对应启动组件nginx。它是一个nginx反向代理，代理Notary client（镜像认证）、Docker client（镜像上传下载等）和浏览器的访问请求（Core Service）给后端的各服务
- UI（Core Service）：对应启动组件harbor-ui。底层数据存储使用mysql数据库，主要提供了四个子功能:
  - UI：一个web管理页面ui
  - API：Harbor暴露的API服务
  - Auth：用户认证服务，decode后的token中的用户信息在这里进行认证；auth后端可以接db、ldap、uaa三种认证实现
  - Token服务（上图中未体现）：负责根据用户在每个project中的role来为每一个docker push/pull命令发布一个token，如果从docker client发送给registry的请求没有带token，registry会重定向请求到token服务创建token
- Registry：对应启动组件registry。负责存储镜像文件，和处理镜像的pull/push命令。Harbor对镜像进行强制的访问控制，Registry会将客户端的每个pull、push请求转发到token服务来获取有效的token
- Admin Service：对应启动组件harbor-adminserver。是系统的配置管理中心附带检查存储用量，ui和jobserver启动时候需要加载adminserver的配置
- Job Sevice：对应启动组件harbor-jobservice。负责镜像复制工作的，他和registry通信，从一个registry pull镜像然后push到另一个registry，并记录job_log
- Log Collector：对应启动组件harbor-log。日志汇总组件，通过docker的log-driver把日志汇总到一起
- DB：对应启动组件harbor-db，负责存储project、 user、 role、replication、image_scan、access等的metadata数据

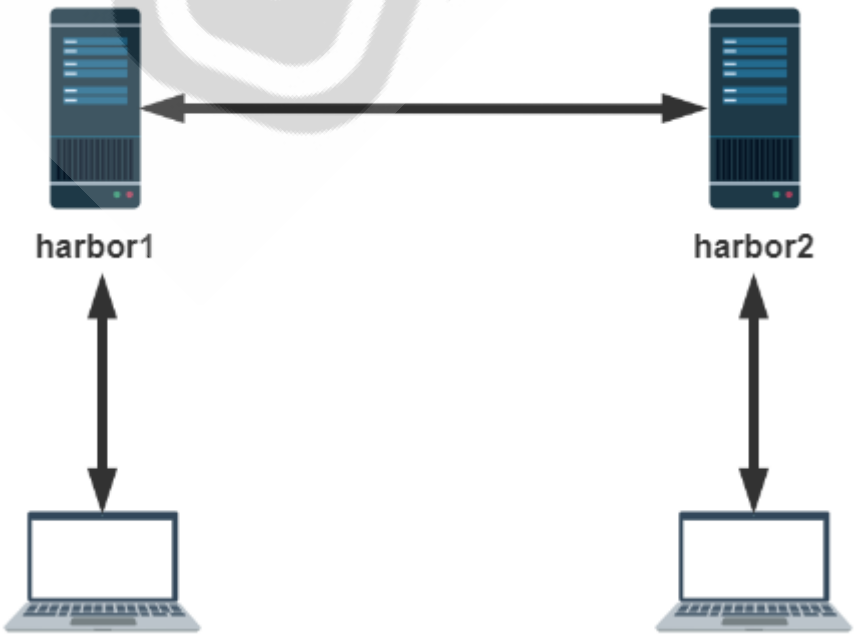
5.4.2 安装Harbor

下载地址：https://github.com/vmware/harbor/releases

安装文档：https://github.com/goharbor/harbor/blob/master/docs/install-config/_index.md

环境准备：共四台主机

- 两台主机harbor服务器，地址：10.0.0.101|102
- 两台主机harbor客户端上传和下载镜像



5.4.2.1 安装 docker

```
[root@ubuntu1804 ~]#cat install_docker_for_ubuntu1804.sh
COLOR="echo -e \\033[1;31m"
END="\033[m"
DOCKER_VERSION="5:19.03.5~3-0~ubuntu-bionic"

install_docker(){
apt update
apt  -y install apt-transport-https ca-certificates curl software-properties-
common
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key
add -
add-apt-repository "deb [arch=amd64] https://mirrors.aliyun.com/docker-
ce/linux/ubuntu $(lsb_release -cs) stable"
apt update
${COLOR}"Docker有以下版本"${END}
apt-cache madison docker-ce
${COLOR}"5秒后即将安装: docker-"${DOCKER_VERSION}" 版本....."${END}
${COLOR}"如果想安装其它Docker版本，请按ctrl+c键退出，修改版本再执行"${END}
sleep 5

apt -y  install docker-ce=${DOCKER_VERSION} docker-ce-cli=${DOCKER_VERSION}

mkdir -p /etc/docker
tee /etc/docker/daemon.json <<-'EOF'
{
      "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}
EOF
systemctl daemon-reload
systemctl enable --now docker
docker version && ${COLOR}"Docker 安装成功"${END} ||  ${COLOR}"Docker 安装失
败"${END}
}
dpkg -s docker-ce &> /dev/null && ${COLOR}"Docker已安装"${END} || install_docker

[root@ubuntu1804 ~]#bash install_docker_for_ubuntu1804.sh
[root@ubuntu1804 ~]#docker version
Client: Docker Engine - Community
 Version:           19.03.5
 API version:       1.40
 Go version:        go1.12.12
 Git commit:        633a0ea838
 Built:             Wed Nov 13 07:29:52 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.5
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.12
  Git commit:       633a0ea838
  Built:            Wed Nov 13 07:28:22 2019
  OS/Arch:          linux/amd64
  Experimental:     false
```

```
 containerd:
  Version:          1.2.10
  GitCommit:        b34a5c8af56e510852c35414db4c1f4fa6172339
 runc:
  Version:          1.0.0-rc8+dev
  GitCommit:        3e425f80a8c931f88e6d94a8c831b9d5aa481657
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

5.4.2.2 先安装docker compose

```
#docker compose 必须先于harbor安装，否则会报以下错误
[root@ubuntu1804 ~]#/apps/harbor/install.sh

[Step 0]: checking installation environment ...

Note: docker version: 19.03.5
✕ Need to install docker-compose(1.7.1+) by yourself first and run this script
again
[root@ubuntu1804 ~]#
```

安装docker compose

```
#方法1:  通过pip安装，版本较新docker_compose-1.25.3，推荐使用
[root@ubuntu1804 ~]#apt -y install python-pip
[root@ubuntu1804 ~]#pip install docker-compose
[root@ubuntu1804 ~]#docker-compose --version
docker-compose version 1.25.3, build unknown

#方法2:  直接从github下载安装对应版本
#参看说明:  https://github.com/docker/compose/releases
curl -L https://github.com/docker/compose/releases/download/1.25.3/docker-
compose-`uname -s`-`uname -m` -o /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose

#方法3:  直接安装，版本较旧docker-compose-1.17.1-2，不推荐使用
[root@ubuntu1804 ~]#apt -y install docker-compose
[root@ubuntu1804 ~]#docker-compose --version
docker-compose version 1.17.1, build unknown
```

5.4.2.3 下载Harbor安装包并解压缩

以下使用 harbor 稳定版本1.7.6 安装包

方法1：下载离线完整安装包,推荐使用

```
[root@ubuntu1804 ~]#wget https://storage.googleapis.com/harbor-releases/release-
1.7.0/harbor-offline-installer-v1.7.6.tgz
```

方法2：下载在线安装包 ,比较慢，不是很推荐

```
[root@ubuntu1804 ~]#wget https://storage.googleapis.com/harbor-releases/release-1.7.0/harbor-online-installer-v1.7.6.tgz
```

```
[root@ubuntu1804 ~]#ls -lh harbor-o*
-rw-r--r-- 1 root root 568M Sep 18 13:24 harbor-offline-installer-v1.7.6.tgz
-rw-r--r-- 1 root root 275K Sep 18 13:37 harbor-online-installer-v1.7.6.tgz
```

解压缩离线包

```
[root@ubuntu1804 ~]#mkdir /apps
[root@ubuntu1804 ~]#tar xvf harbor-offline-installer-v1.7.6.tgz  -C /apps/
```

5.4.2.4 编辑配置文件 harbor.cfg

最新文档: https://github.com/goharbor/harbor/blob/master/docs/install-config/configure-yml-file.md

```
[root@ubuntu1804 ~]#vim /apps/harbor/harbor.cfg
#只需要修改下面两行
hostname = 10.0.0.101  #修改此行,指向当前主机IP 或 FQDN
harbor_admin_password = 123456 #修改此行指定harbor登录用户admin的密码,默认用户/密码:admin/Harbor12345

#可选项
ui_url_protocol = http #默认即可,如果修改为https,需要指定下面证书路径
ssl_cert = /data/cert/server.crt #默认即可,https时，需指定下面证书文件路径
ss_cert_key = /data/cert/server.key   #默认即可,https时，需指定下面私钥文件路径
```

5.4.2.5 运行 harbor 安装脚本

```
#先安装python
root@ubuntu1804 ~]#apt -y install python
#安装docker harbor
root@ubuntu1804 ~]#/apps/harbor/install.sh

[Step 0]: checking installation environment ...

Note: docker version: 19.03.5

Note: docker-compose version: 1.25.3

[Step 1]: loading Harbor images ...
......
[Step 4]: starting Harbor ...
Creating network "harbor_harbor" with the default driver
Creating harbor-log ... done
Creating registryctl        ... done
Creating harbor-db          ... done
Creating redis              ... done
Creating registry           ... done
Creating harbor-adminserver ... done
Creating harbor-core        ... done
```

```
Creating harbor-jobservice  ... done
Creating harbor-portal      ... done
Creating nginx              ... done

✓ ----Harbor has been installed and started successfully.----

Now you should be able to visit the admin portal at http://10.0.0.101.
For more details, please visit https://github.com/goharbor/harbor .

#安装harbor后会自动开启很多相关容器
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                                  COMMAND
     CREATED             STATUS                      PORTS
                                  NAMES
1b47a3eeedd2        goharbor/nginx-photon:v1.7.6           "nginx -g 'daemon
of…"   14 minutes ago      Up 14 minutes (healthy)   0.0.0.0:80->80/tcp,
0.0.0.0:443->443/tcp, 0.0.0.0:4443->4443/tcp   nginx
5f3a0a0db734        goharbor/harbor-portal:v1.7.6          "nginx -g 'daemon
of…"   14 minutes ago      Up 14 minutes (healthy)   80/tcp
                                  harbor-portal
8e4265efe8ee        goharbor/harbor-jobservice:v1.7.6      "/harbor/start.sh"
      14 minutes ago      Up 14 minutes
                                  harbor-jobservice
d1a048525d79        goharbor/harbor-core:v1.7.6            "/harbor/start.sh"
      14 minutes ago      Up 14 minutes (healthy)
                                  harbor-core
4a989eb92af1        goharbor/harbor-adminserver:v1.7.6     "/harbor/start.sh"
      14 minutes ago      Up 14 minutes (healthy)
                                  harbor-adminserver
c875d3959c56        goharbor/registry-photon:v2.6.2-v1.7.6   "/entrypoint.sh
/etc…"   14 minutes ago      Up 14 minutes (healthy)   5000/tcp
                                  registry
2a963125a0e6        goharbor/redis-photon:v1.7.6           "docker-
entrypoint.s…"   14 minutes ago      Up 14 minutes             6379/tcp
                                  redis
a0751df44d68        goharbor/harbor-registryctl:v1.7.6     "/harbor/start.sh"
      14 minutes ago      Up 14 minutes (healthy)
                                  registryctl
b0ef6ed0d46b        goharbor/harbor-db:v1.7.6              "/entrypoint.sh
post…"   14 minutes ago      Up 14 minutes (healthy)   5432/tcp
                                  harbor-db
8e667c6ccbc1        goharbor/harbor-log:v1.7.6             "/bin/sh -c
/usr/loc…"   14 minutes ago      Up 14 minutes (healthy)   127.0.0.1:1514-
>10514/tcp                                  harbor-log
[root@ubuntu1804 ~]#
```

5.4.2.5 实现开机自动启动 harbor

5.4.2.5.1 方法1: 通过service文件实现

```
[root@harbor ~]#vim /lib/systemd/system/harbor.service
[Unit]
Description=Harbor
After=docker.service systemd-networkd.service systemd-resolved.service
Requires=docker.service
Documentation=http://github.com/vmware/harbor
```

```
[Service]
Type=simple
Restart=on-failure
RestartSec=5
ExecStart=/usr/bin/docker-compose -f  /apps/harbor/docker-compose.yml up
ExecStop=/usr/bin/docker-compose -f /apps/harbor/docker-compose.yml down

[Install]
WantedBy=multi-user.target


[root@harbor ~]#systemctl daemon-reload
[root@harbor ~]#systemctl enable harbor
```

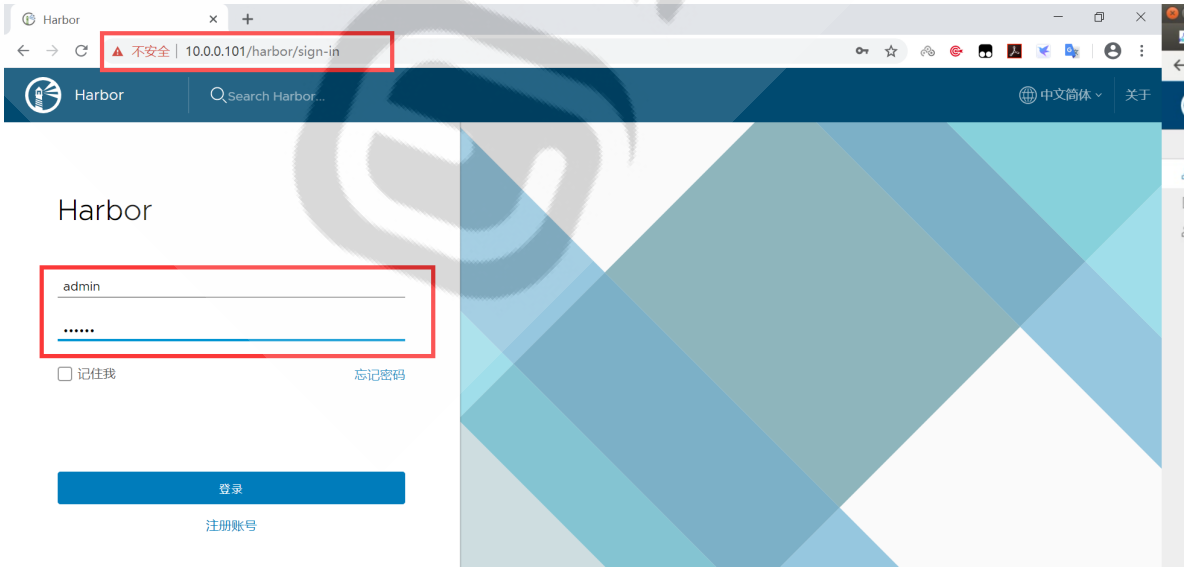
5.4.2.5.2 方法2: 通过 rc.local实现

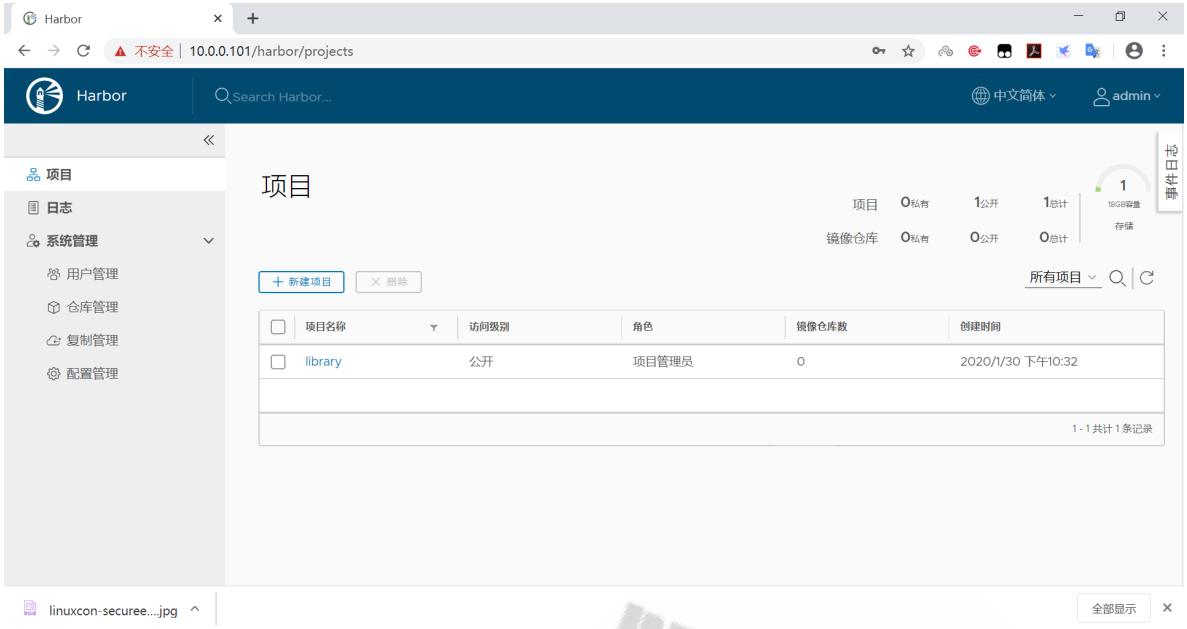
```
[root@harbor ~]#cat /etc/rc.local
#!/bin/bash
cd /apps/harbor
/usr/bin/docker-compose up

[root@harbor ~]#chmod +x /etc/rc.local
```

5.4.2.6 登录 harbor 主机网站

用浏览器访问: http://10.0.0.101/

- 用户名: admin
- 密码: 即前面harbor.cfg中指定的密码





5.4.2.7 实战案例: 一键安装Harbor脚本

```
[root@ubuntu1804 ~]#cat install_harbor_for_ubuntu1804.sh
#!/bin/bash
#Description: Install harbor on ubuntu1804
#Author: laowang

COLOR="echo -e \E[1;31m"
END="\E[m"
DOCKER_VERSION="5:19.03.5~3-0~ubuntu-bionic"
HARBOR_VERSION=1.7.6
IPADDR=`hostname -I|awk '{print $1}'`
HARBOR_ADMIN_PASSWORD=123456

install_docker(){
${COLOR}"开始安装 Docker....."${END}
sleep 1

apt update
apt  -y install apt-transport-https ca-certificates curl software-properties-
common
curl -fsSL https://mirrors.aliyun.com/docker-ce/linux/ubuntu/gpg | sudo apt-key
add -
add-apt-repository "deb [arch=amd64] https://mirrors.aliyun.com/docker-
ce/linux/ubuntu $(lsb_release -cs) stable"
apt update

${COLOR}"Docker有以下版本:"${END}
sleep 2
apt-cache madison docker-ce
${COLOR}"5秒后即将安装: docker-"${DOCKER_VERSION}" 版本....."${END}
${COLOR}"如果想安装其它Docker版本，请按ctrl+c键退出，修改版本再执行"${END}
sleep 5

apt -y  install docker-ce=${DOCKER_VERSION} docker-ce-cli=${DOCKER_VERSION}

mkdir -p /etc/docker
```

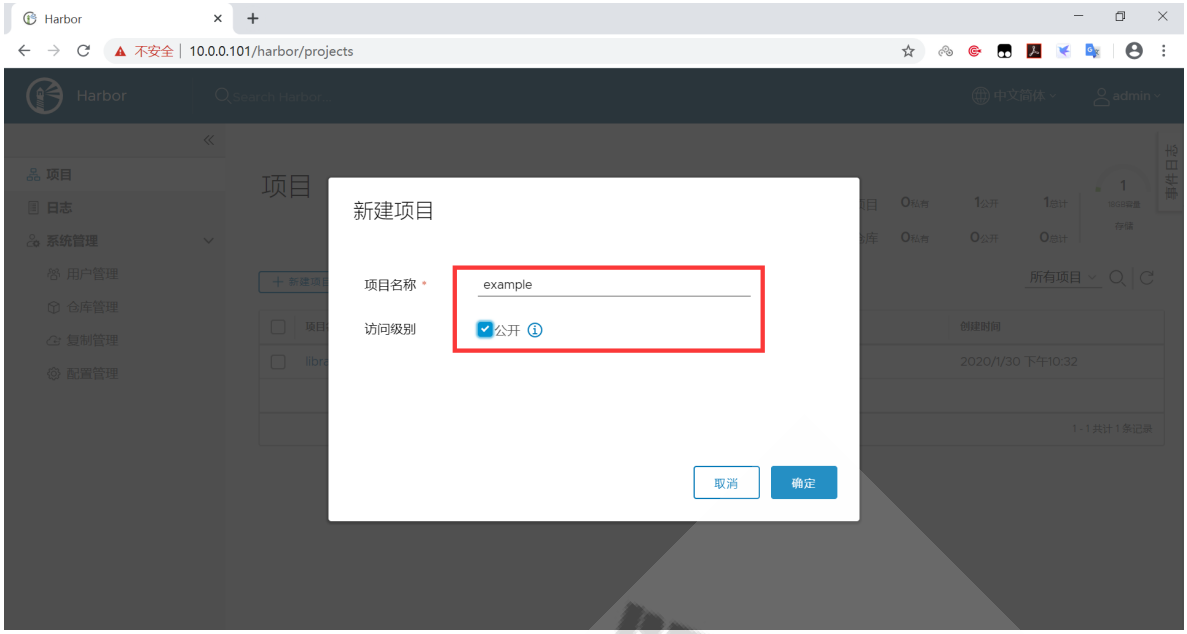
```
tee /etc/docker/daemon.json <<-'EOF'
{
      "registry-mirrors": ["https://si7y70hh.mirror.aliyuncs.com"]
}
EOF
systemctl daemon-reload
systemctl restart docker
docker version && ${COLOR}"Docker 安装完成"${END} ||  ${COLOR}"Docker 安装失
败"${END}
}

install_docker_compose(){
${COLOR}"开始安装 Docker compose....."${END}
sleep 1

curl -L https://github.com/docker/compose/releases/download/1.25.3/docker-
compose-`uname -s`-`uname -m` -o /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose

docker-compose --version &&  ${COLOR}"Docker Compose 安装完成"${END} ||
 ${COLOR}"Docker compose 安装失败"${END}
}

install_harbor(){
${COLOR}"开始安装 Harbor....."${END}
sleep 1

wget https://storage.googleapis.com/harbor-releases/release-1.7.0/harbor-
offline-installer-v${HARBOR_VERSION}.tgz
mkdir /apps
tar xvf harbor-offline-installer-v${HARBOR_VERSION}.tgz  -C /apps/

sed -i.bak -e 's/^hostname =.*/hostname = '''$IPADDR'''/' -e
's/^harbor_admin_password =.*/harbor_admin_password =
'''$HARBOR_ADMIN_PASSWORD'''/' /apps/harbor/harbor.cfg

apt -y install python

/apps/harbor/install.sh && ${COLOR}"Harbor 安装完成"${END} ||  ${COLOR}"Harbor 安
装失败"${END}

}
harbor_service (){
cat > /lib/systemd/system/harbor.service <<EOF
[Unit]
Description=Harbor
After=docker.service systemd-networkd.service systemd-resolved.service
Requires=docker.service
Documentation=http://github.com/vmware/harbor

[Service]
Type=simple
Restart=on-failure
RestartSec=5
ExecStart=/usr/bin/docker-compose -f  /apps/harbor/docker-compose.yml up
ExecStop=/usr/bin/docker-compose -f /apps/harbor/docker-compose.yml down

[Install]
```

```
WantedBy=multi-user.target
EOF

systemctl daemon-reload
systemctl enable  harbor &>/dev/null ||  ${COLOR}"Harbor已配置为开机自动启动"${END}

}
dpkg -s docker-ce &> /dev/null && ${COLOR}"Docker已安装"${END} || install_docker

docker-compose --version &> /dev/null && ${COLOR}"Docker Compose已安装"${END} ||
install_docker_compose

install_harbor

harbor_service
```

5.4.3 使用单主机 harbor

5.4.3.1 建立项目

harbor上必须先建立项目，才能上传镜像



5.4.3.2 命令行登录 harbor

```
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
 --insecure-registry 10.0.0.101 --insecure-registry 10.0.0.102

[root@ubuntu1804 ~]#systemctl daemon-reload
[root@ubuntu1804 ~]#systemctl restart docker
[root@ubuntu1804 ~]#docker login 10.0.0.101
Username: admin
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded

#查看进程是否添加上面设置
[root@ubuntu1804 ~]#ps aux|grep dockerd
```

```
root      17347  7.8  9.6 839272 94784 ?        Ssl  22:54   0:15
/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --
insecure-registry 10.0.0.101 --insecure-registry 10.0.0.102
root      17630  0.0  0.1  14428  1008 pts/4    S+   22:57   0:00 grep --
color=auto dockerd

[root@ubuntu1804 ~]#cat .docker/config.json
{
    "auths": {
        "10.0.0.101": {
            "auth": "YWRtaW46MTIzNDU2"
        },
        "https://index.docker.io/v1/": {
            "auth": "d2FuZ3hpYW9jaHVuOmxidG9vdGgwNjE4"
        },
        "registry.cn-beijing.aliyuncs.com": {
            "auth": "MjkzMDg2MjBAcXEuY29tOmxidG9vdGgwNjE4"
        }
    },
    "HttpHeaders": {
        "User-Agent": "Docker-Client/19.03.5 (linux)"
    }
}
```

5.4.3.3 给本地镜像打标签并上传到harbor

修改 images 的名称，不修改成指定格式无法将镜像上传到 harbor 仓库

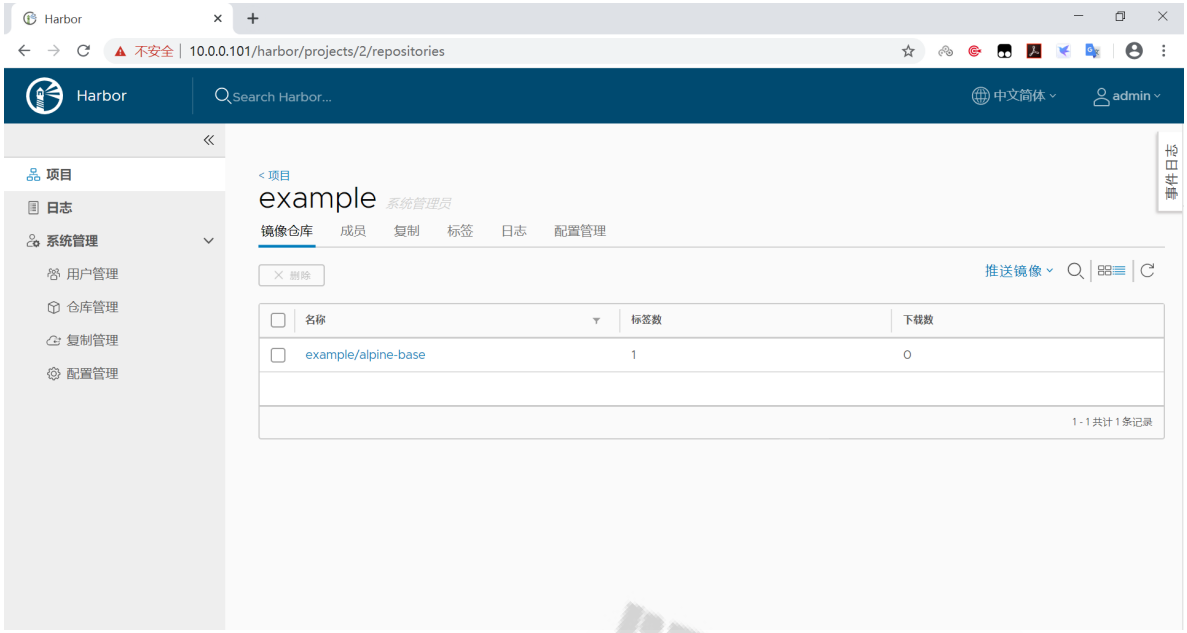
格式为:

```
Harbor主机IP/项目名/image名字:版本
```

范例:

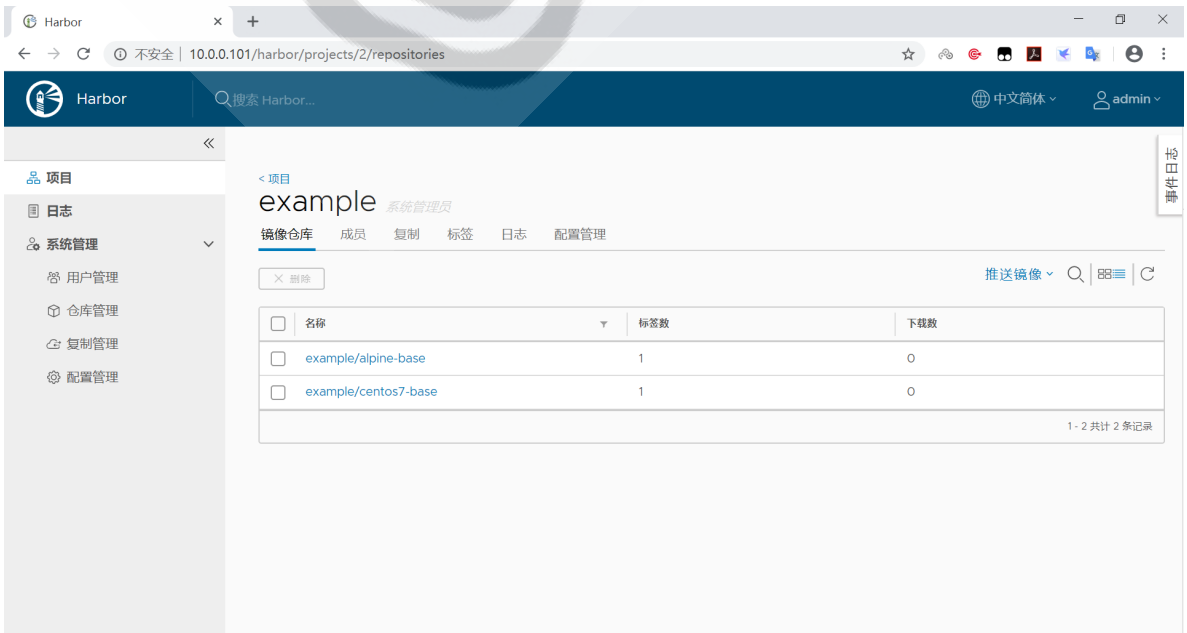
```
#上传镜像前，必须先登录harbor
[root@ubuntu1804 ~]#docker login 10.0.0.101
Username: admin
Password:
WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
[root@ubuntu1804 ~]#docker tag alpine-base:3.11 10.0.0.101/example/alpine-
base:3.11
[root@ubuntu1804 ~]#docker push 10.0.0.101/example/alpine-base:3.11
```

访问harbor网站验证上传镜像成功

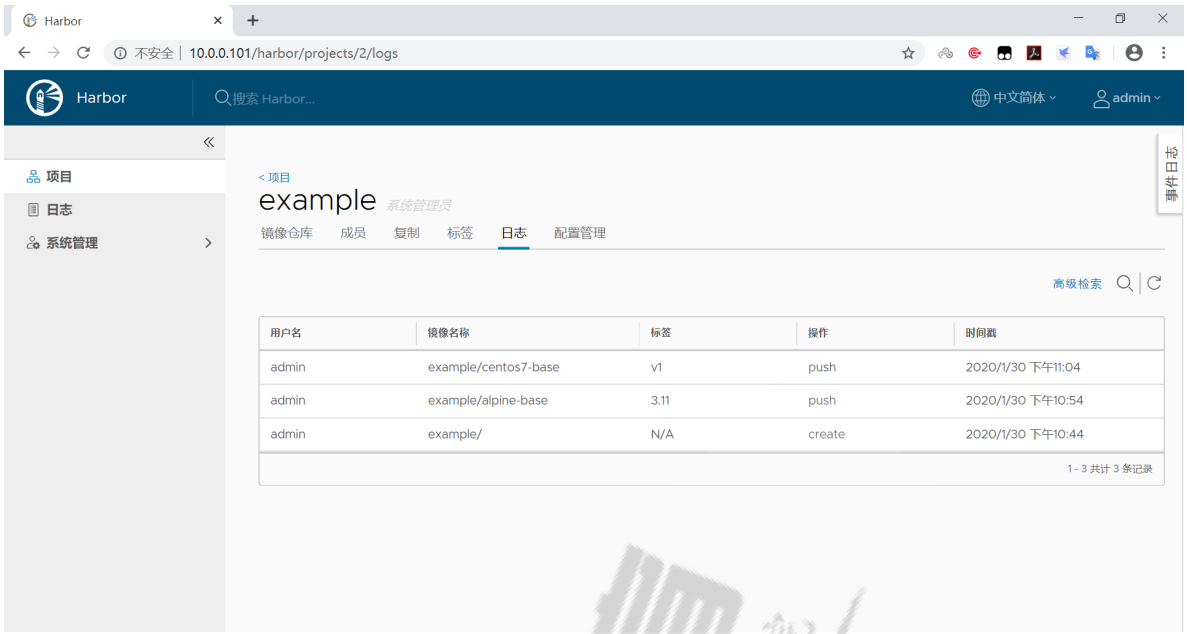


范例：如果不事先建立项目，上传镜像失败

```
[root@ubuntu1804 ~]#docker tag centos7-base:v1 10.0.0.101/example2/centos7-base:v1
[root@ubuntu1804 ~]#docker push 10.0.0.101/example2/centos7-base:v1
The push refers to repository [10.0.0.101/example2/centos7-base]
2073413aebd6: Preparing
6ec9af97c369: Preparing
034f282942cd: Preparing
denied: requested access to the resource is denied
[root@ubuntu1804 ~]#docker tag centos7-base:v1 10.0.0.101/example/centos7-base:v1
[root@ubuntu1804 ~]#docker push 10.0.0.101/example/centos7-base:v1
The push refers to repository [10.0.0.101/example/centos7-base]
2073413aebd6: Pushed
6ec9af97c369: Pushed
034f282942cd: Pushed
v1: digest: sha256:02cd943f2569c7c55f08a979fd9661f1fd7893c424bca7b343188654ba63d98d size: 949
```



可以看到操作的日志记录



5.4.3.4 下载harbor的镜像

在10.0.0.103的CentOS 7 的主机上无需登录，即可下载镜像

下载前必须修改docker的service 文件，加入harbor服务器的地址才可以下载

范例：修改docker的service文件

```
[root@centos7 ~]#docker pull 10.0.0.101/example/centos7-base:v1
Error response from daemon: Get https://10.0.0.101/v2/: dial tcp 10.0.0.101:443:
connect: connection refused
[root@ubuntu1804 ~]#vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
 --insecure-registry 10.0.0.101 --insecure-registry 10.0.0.102
[root@centos7 ~]#systemctl daemon-reload
[root@centos7 ~]#systemctl restart docker
[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
```

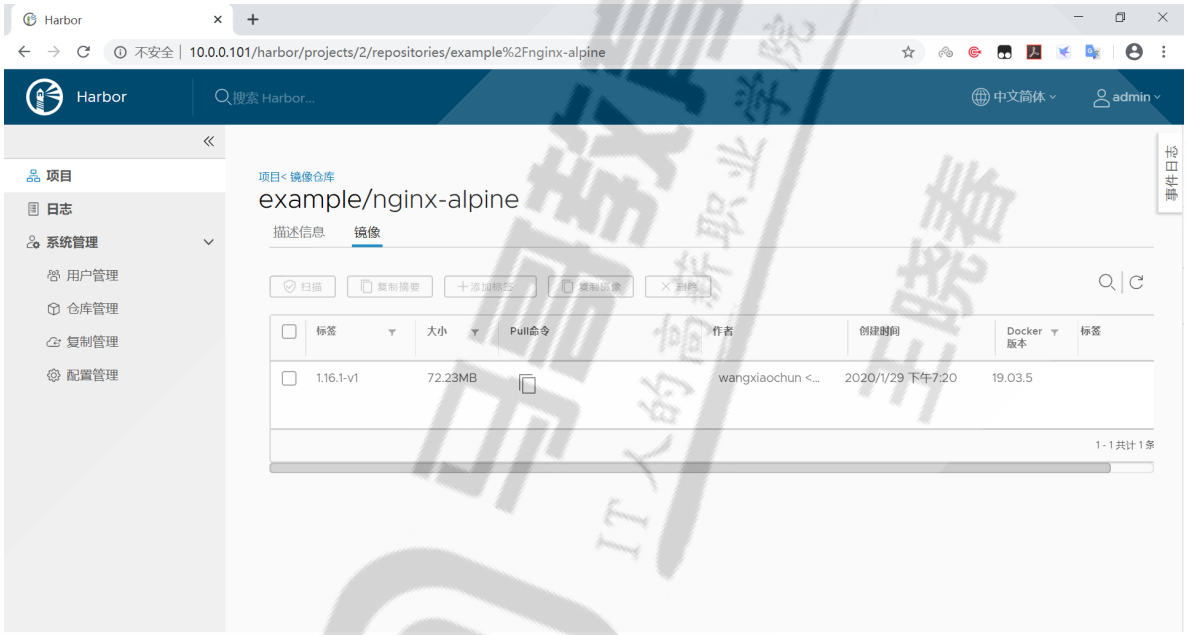
范例：从harbor下载镜像

```
[root@centos7 ~]#docker pull 10.0.0.101/example/centos7-base:v1
v1: Pulling from example/centos7-base
f34b00c7da20: Pull complete
544476d462f7: Pull complete
39345915aa1b: Pull complete
Digest: sha256:02cd943f2569c7c55f08a979fd9661f1fd7893c424bca7b343188654ba63d98d
Status: Downloaded newer image for 10.0.0.101/example/centos7-base:v1
10.0.0.101/example/centos7-base:v1
[root@centos7 ~]#docker images
REPOSITORY                          TAG                 IMAGE ID
CREATED             SIZE
10.0.0.101/example/centos7-base   v1                  34ab3afcd3b3        2 days
ago          403MB
```

5.4.3.5 创建自动打标签上传镜像脚本

```
#在10.0.0.100上修改以前的build.sh脚本
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16.1-alpine/
[root@ubuntu1804 1.16.1-alpine]#vim build.sh
[root@ubuntu1804 1.16.1-alpine]#cat build.sh
#!/bin/bash
TAG=$1
docker build -t 10.0.0.101/example/nginx-alpine:1.16.1-${TAG} .
docker push 10.0.0.101/example/nginx-alpine:1.16.1-${TAG}
docker rmi -f 10.0.0.101/example/nginx-alpine:1.16.1-${TAG}
[root@ubuntu1804 1.16.1-alpine]#bash build.sh v1
```

登录harbor网站验证脚本上传镜像成功



5.4.3.6 修改 harbor 配置

后期如果修改harbor配置，比如：修改IP地址等，可执行以下步骤生效

方法1:

```
[root@ubuntu1804 ~]#cd /apps/harbor/
[root@ubuntu1804 harbor]#docker-compose stop
Stopping nginx              ... done
Stopping harbor-portal      ... done
Stopping harbor-jobservice  ... done
Stopping harbor-core        ... done
Stopping harbor-adminserver ... done
Stopping harbor-db          ... done
Stopping registryctl        ... done
Stopping registry           ... done
Stopping redis              ... done
Stopping harbor-log         ...
```

```
#所有相关容器都退出
[root@ubuntu1804 harbor]#docker ps  -a
CONTAINER ID        IMAGE                                   COMMAND
     CREATED             STATUS                          PORTS               NAMES
4ec3c3885407        goharbor/nginx-photon:v1.7.6            "nginx -g 'daemon
of…"   32 minutes ago      Exited (0) 51 seconds ago
nginx
5707b4ac41d8        goharbor/harbor-portal:v1.7.6           "nginx -g 'daemon
of…"   32 minutes ago      Exited (0) 50 seconds ago
harbor-portal
0ed230b9b714        goharbor/harbor-jobservice:v1.7.6       "/harbor/start.sh"
      32 minutes ago      Exited (137) 41 seconds ago
harbor-jobservice
fec659188349        goharbor/harbor-core:v1.7.6             "/harbor/start.sh"
      32 minutes ago      Exited (137) 30 seconds ago
harbor-core
910d14c1d7f7        goharbor/harbor-adminserver:v1.7.6      "/harbor/start.sh"
      32 minutes ago      Exited (137) 20 seconds ago
harbor-adminserver
4348f503aa0e        goharbor/harbor-db:v1.7.6               "/entrypoint.sh
post…"   32 minutes ago      Exited (255) 48 seconds ago
harbor-db
beff6886f0f1        goharbor/harbor-registryctl:v1.7.6      "/harbor/start.sh"
      32 minutes ago      Exited (137) 41 seconds ago
registryctl
428c99d274bf        goharbor/registry-photon:v2.6.2-v1.7.6  "/entrypoint.sh
/etc…"   32 minutes ago      Exited (137) 20 seconds ago
registry
775b4026fa4e        goharbor/redis-photon:v1.7.6            "docker-
entrypoint.s…"   32 minutes ago      Exited (137) 30 seconds ago
      redis
c6f44e2034c6        goharbor/harbor-log:v1.7.6              "/bin/sh -c
/usr/loc…"   32 minutes ago      Exited (137) 9 seconds ago
   harbor-log

#修改harbor配置
[root@ubuntu1804 harbor]#vim harbor.cfg

#更新配置
[root@ubuntu1804 ~]#/apps/harbor/prepare
Clearing the configuration file: /apps/harbor/common/config/db/env
Clearing the configuration file: /apps/harbor/common/config/core/private_key.pem
Clearing the configuration file: /apps/harbor/common/config/core/env
Clearing the configuration file: /apps/harbor/common/config/core/app.conf
Clearing the configuration file: /apps/harbor/common/config/adminserver/env
Clearing the configuration file: /apps/harbor/common/config/registryctl/env
Clearing the configuration file:
/apps/harbor/common/config/registryctl/config.yml
Clearing the configuration file: /apps/harbor/common/config/registry/root.crt
Clearing the configuration file: /apps/harbor/common/config/registry/config.yml
Clearing the configuration file: /apps/harbor/common/config/log/logrotate.conf
Clearing the configuration file: /apps/harbor/common/config/nginx/nginx.conf
Clearing the configuration file: /apps/harbor/common/config/jobservice/env
Clearing the configuration file:
/apps/harbor/common/config/jobservice/config.yml
loaded secret from file: /data/secretkey
Generated configuration file: /apps/harbor/common/config/nginx/nginx.conf
Generated configuration file: /apps/harbor/common/config/adminserver/env
```

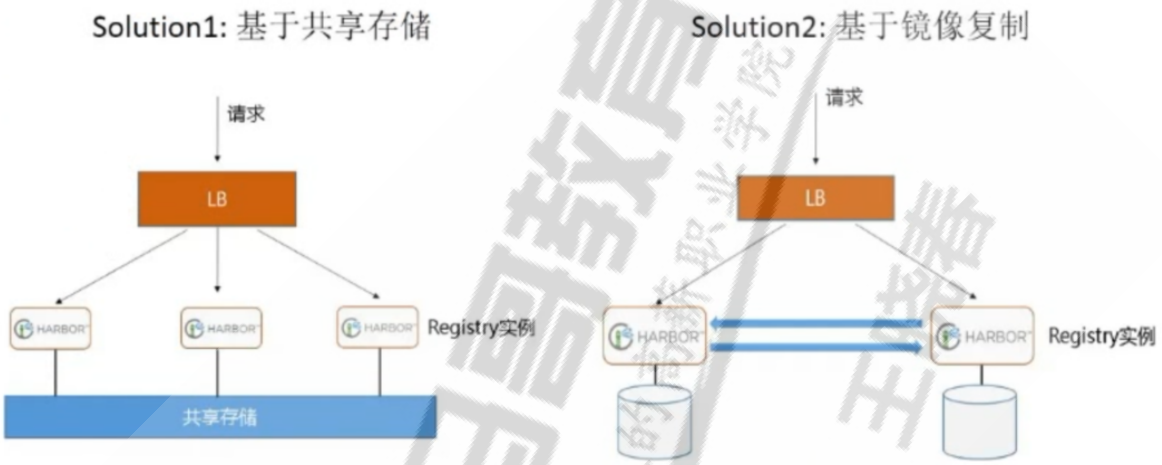
```
Generated configuration file: /apps/harbor/common/config/core/env
Generated configuration file: /apps/harbor/common/config/registry/config.yml
Generated configuration file: /apps/harbor/common/config/db/env
Generated configuration file: /apps/harbor/common/config/jobservice/env
Generated configuration file: /apps/harbor/common/config/jobservice/config.yml
Generated configuration file: /apps/harbor/common/config/log/logrotate.conf
Generated configuration file: /apps/harbor/common/config/registryctl/env
Generated configuration file: /apps/harbor/common/config/core/app.conf
Generated certificate, key file:
/apps/harbor/common/config/core/private_key.pem, cert file:
/apps/harbor/common/config/registry/root.crt
The configuration files are ready, please use docker-compose to start the
service.

#重新启动docker compose
[root@ubuntu1804 harbor]#docker-compose  start
Starting log         ... done
Starting postgresql  ... done
Starting redis       ... done
Starting adminserver ... done
Starting registry    ... done
Starting core        ... done
Starting jobservice  ... done
Starting portal      ... done
Starting proxy       ... done
Starting registryctl ... done

#相关容器自动启动
[root@ubuntu1804 harbor]#docker ps
CONTAINER ID        IMAGE                                  COMMAND
     CREATED             STATUS                            PORTS
                                                NAMES
4ec3c3885407        goharbor/nginx-photon:v1.7.6           "nginx -g 'daemon
of…"   34 minutes ago      Up 9 seconds (health: starting)    0.0.0.0:80-
>80/tcp, 0.0.0.0:443->443/tcp, 0.0.0.0:4443->4443/tcp   nginx
5707b4ac41d8        goharbor/harbor-portal:v1.7.6          "nginx -g 'daemon
of…"   34 minutes ago      Up 9 seconds (health: starting)    80/tcp
                                                 harbor-portal
0ed230b9b714        goharbor/harbor-jobservice:v1.7.6      "/harbor/start.sh"
      34 minutes ago      Up 10 seconds
                                                 harbor-jobservice
fec659188349        goharbor/harbor-core:v1.7.6            "/harbor/start.sh"
      34 minutes ago      Up 11 seconds (health: starting)
                                                harbor-core
910d14c1d7f7        goharbor/harbor-adminserver:v1.7.6     "/harbor/start.sh"
      34 minutes ago      Up 14 seconds (health: starting)
                                                harbor-adminserver
4348f503aa0e        goharbor/harbor-db:v1.7.6              "/entrypoint.sh
post…"   34 minutes ago      Up 13 seconds (health: starting)   5432/tcp
                                                   harbor-db
beff6886f0f1        goharbor/harbor-registryctl:v1.7.6     "/harbor/start.sh"
      34 minutes ago      Up 12 seconds (health: starting)
                                                registryctl
428c99d274bf        goharbor/registry-photon:v2.6.2-v1.7.6   "/entrypoint.sh
/etc…"   34 minutes ago      Up 13 seconds (health: starting)   5000/tcp
                                                   registry
```

```
775b4026fa4e        goharbor/redis-photon:v1.7.6              "docker-
entrypoint.s…"   34 minutes ago      Up 11 seconds                          6379/tcp
                                                                  redis
c6f44e2034c6        goharbor/harbor-log:v1.7.6                "/bin/sh -c
/usr/loc…"   34 minutes ago      Up 16 seconds (health: starting)
127.0.0.1:1514->10514/tcp                                             harbor-log
[root@ubuntu1804 harbor]#
```

方法2:

```
[root@ubuntu1804 ~]#/apps/harbor/install.sh
```

5.4.4 实现 harbor 高可用

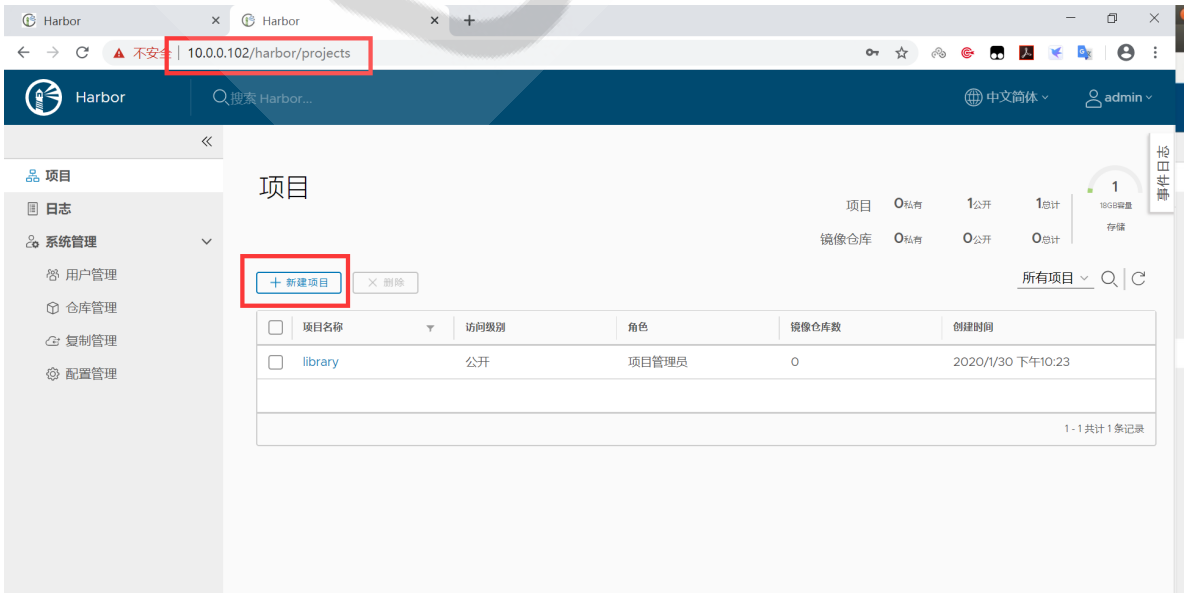


Harbor支持基于策略的Docker镜像复制功能，这类似于MySQL的主从同步，其可以实现不同的数据中心、不同的运行环境之间同步镜像，并提供友好的管理界面，大大简化了实际运维中的镜像管理工作，已经有用很多互联网公司使用harbor搭建内网docker仓库的案例，并且还有实现了双向复制功能

5.4.4.1 安装第二台 harbor主机

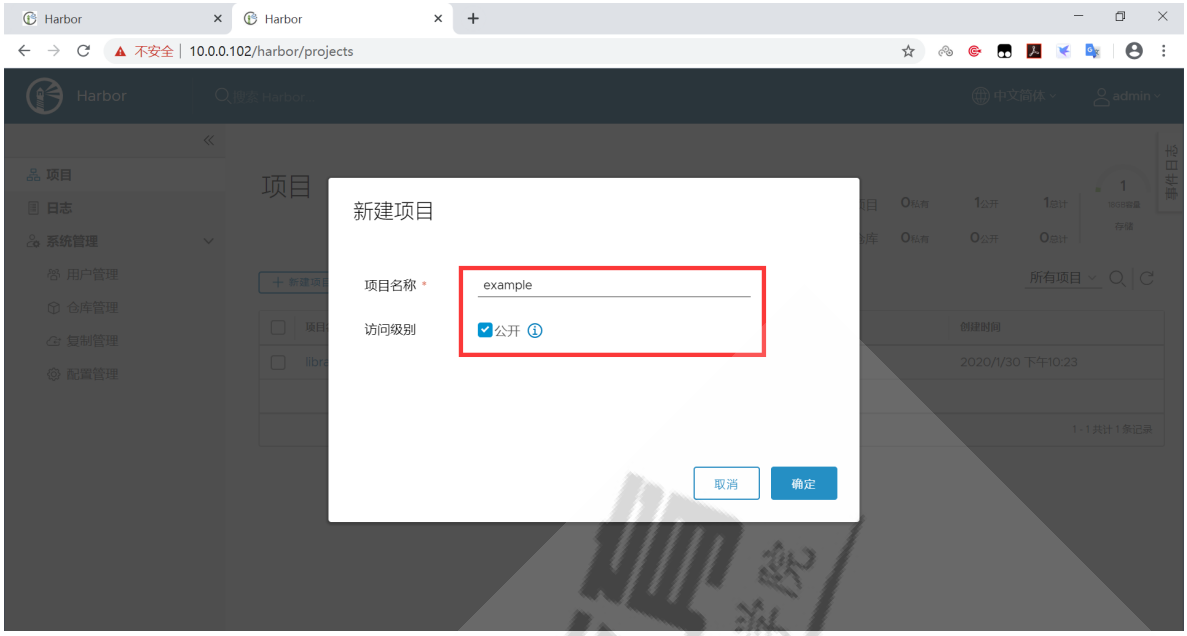
参考5.4.2的过程，在第二台主机上安装部署好harbor，并登录系统

注意: harbor.cfg中配置 hostname = 10.0.0.102



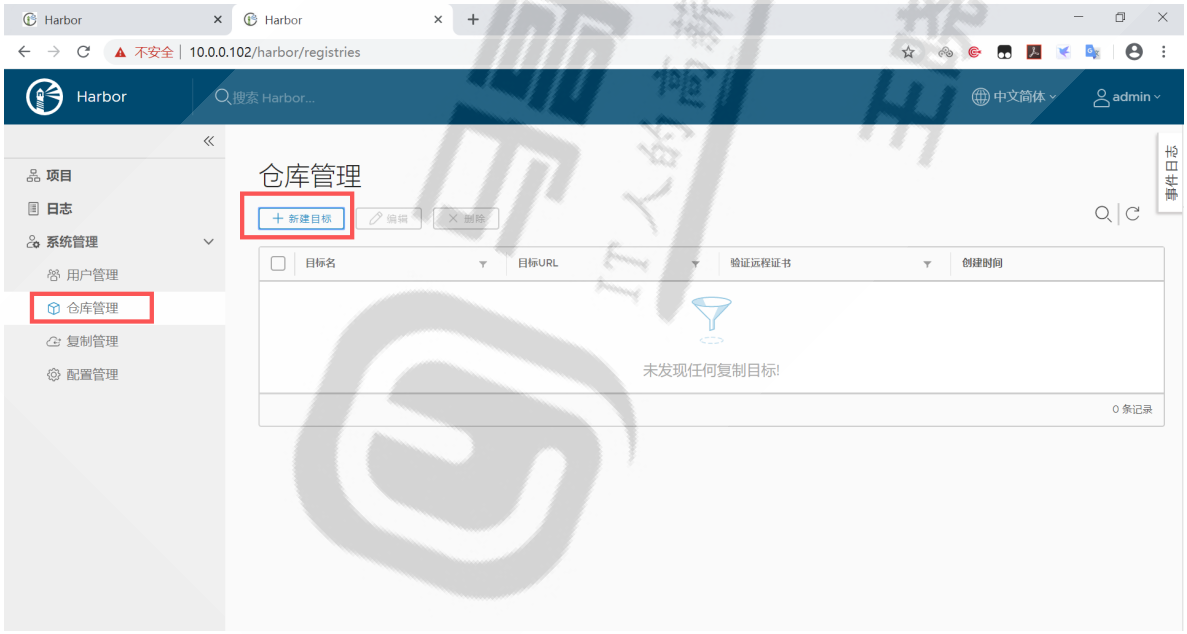
5.4.4.2 第二台harbor上新建项目

参考第一台harbor服务器的项目名称，在第二台harbor服务器上新建与之同名的项目

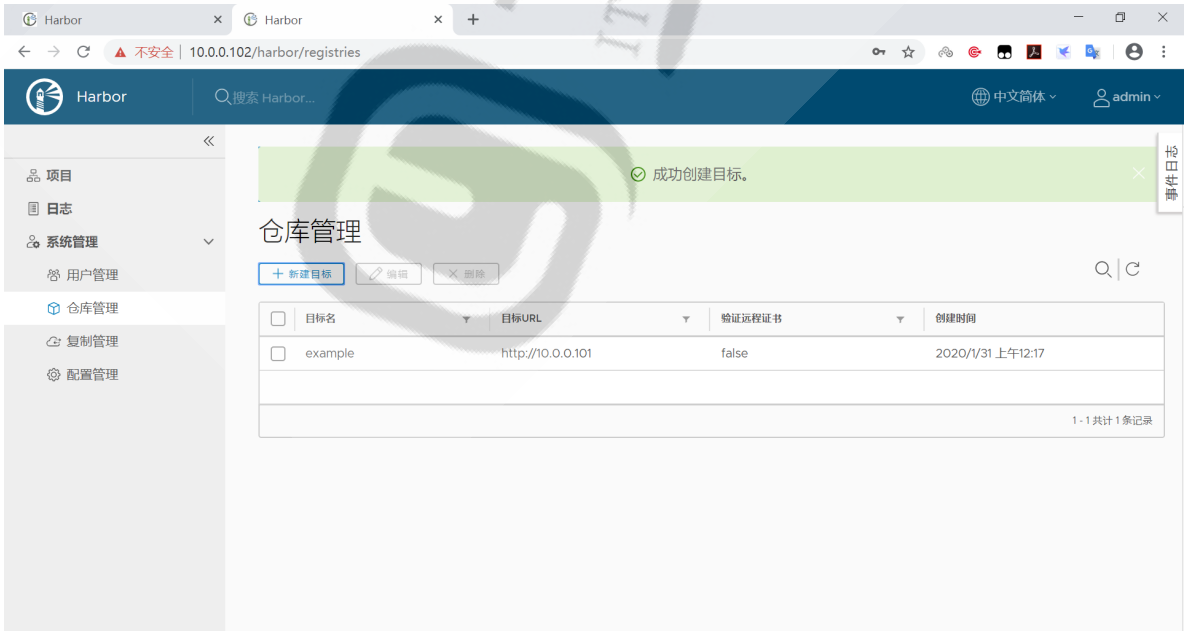


5.4.4.3 第二台harbor上仓库管理中新建目标

参考第一台主机信息,新建复制（同步）目标信息,将第一台主机设为复制的目标

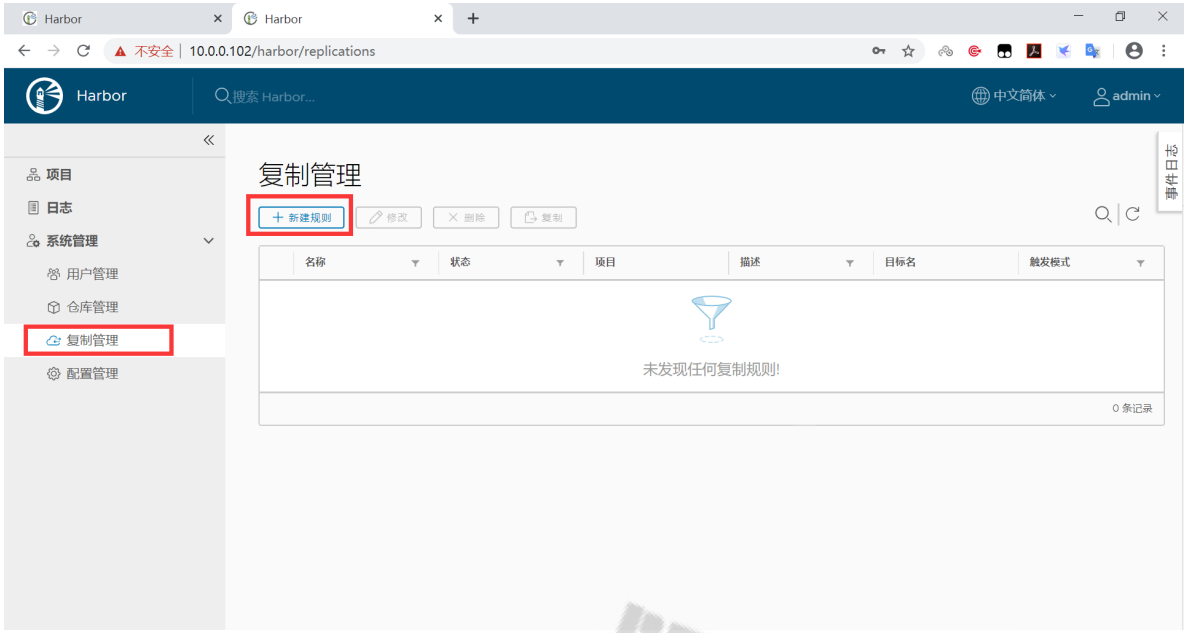


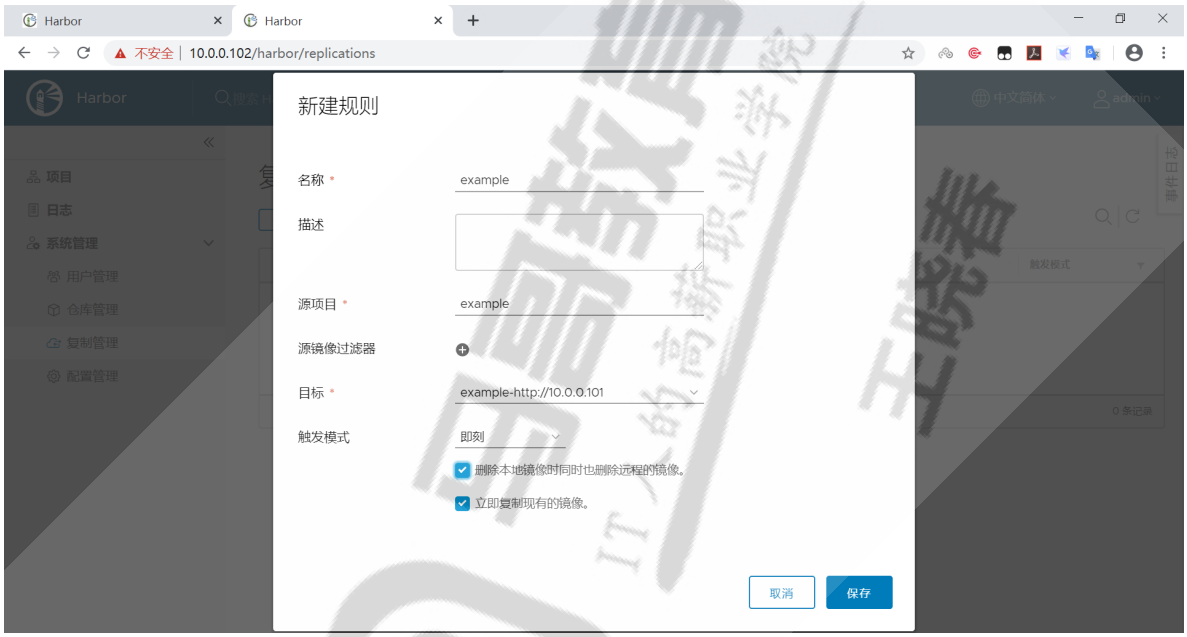
输入第一台harbor服务器上的主机和用户信息



5.4.4.4 第二台harbor上新建复制规则实现到第一台harbor的单向复制

在第二台harbor上建立复制的目标主机,即第一台harbor主机,将第二台harbor上面的镜像复制到第一台harbor上

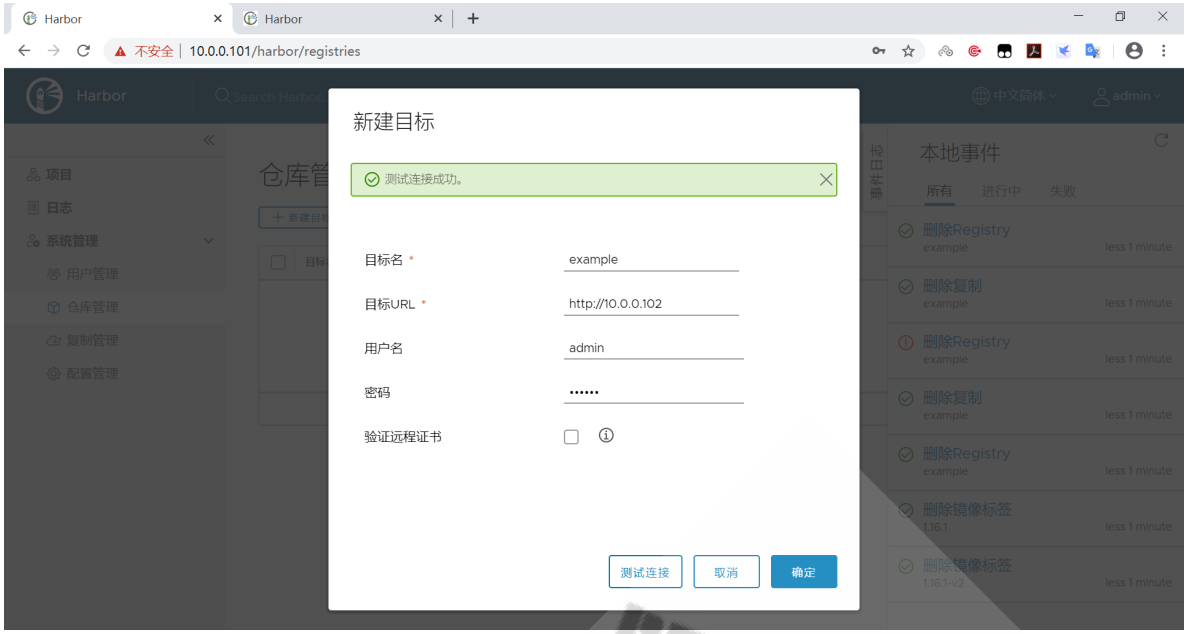


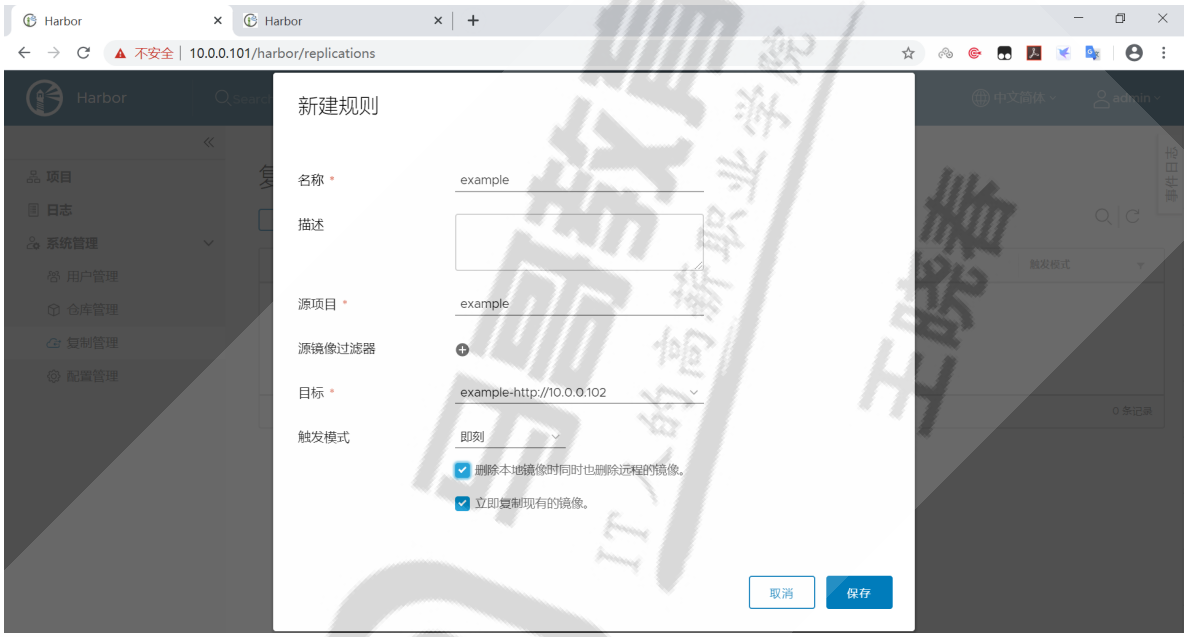


5.4.4.5 在第一台harbor主机上重复上面操作

以上操作，只是实现了从第二台harbor主机10.0.0.102到第一台harbor主机10.0.0.101的单向同步

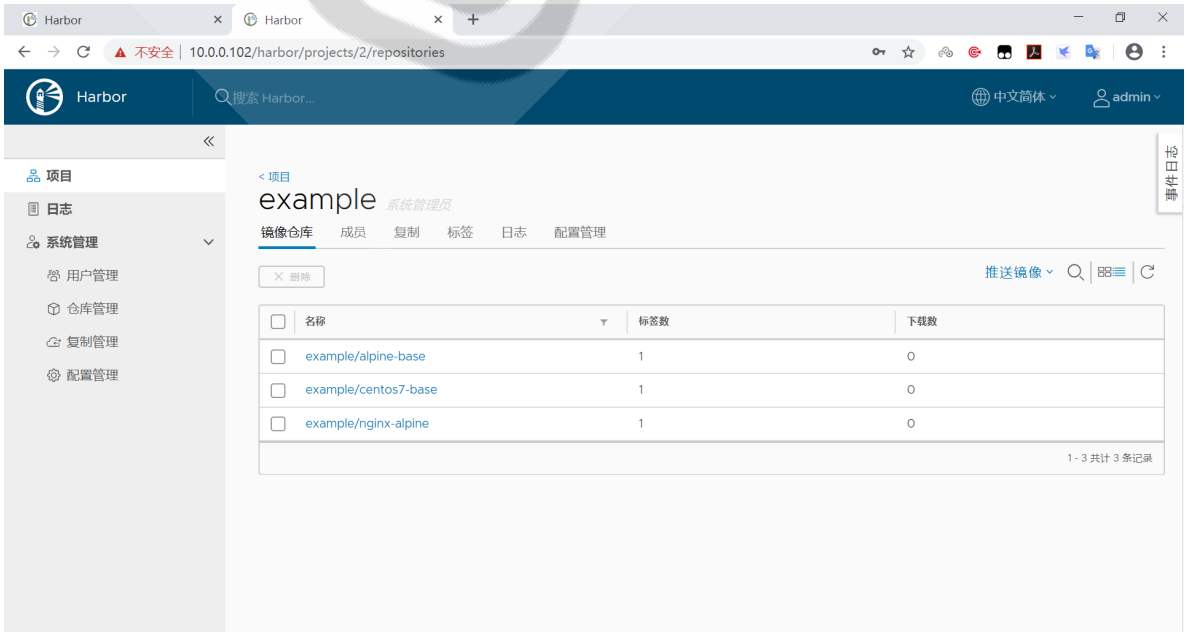
在第一台harbor上再执行下面操作，才实现双向同步



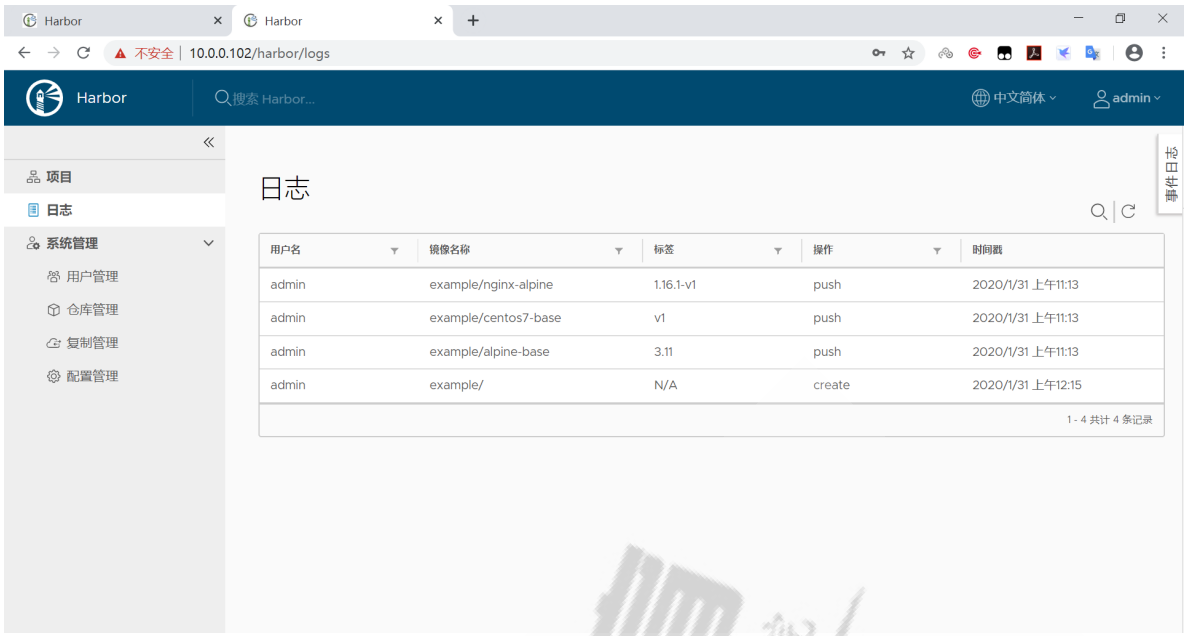


5.4.4.6 确认同步成功

在第二台harbor主机上可以查看到从第一台主机同步过来的镜像

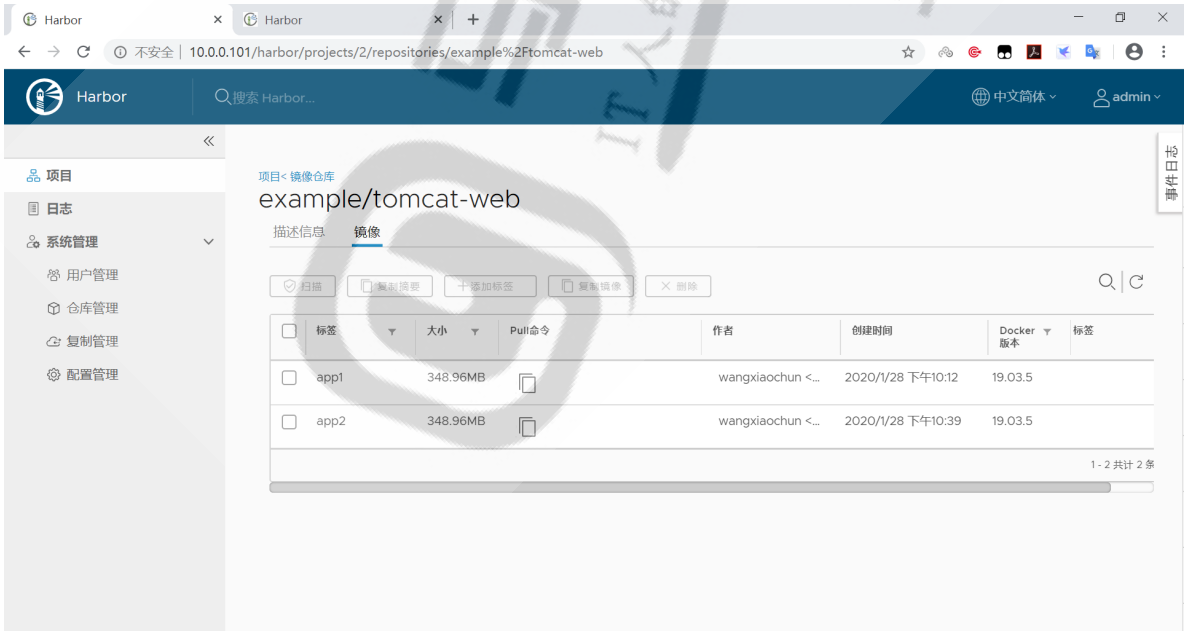


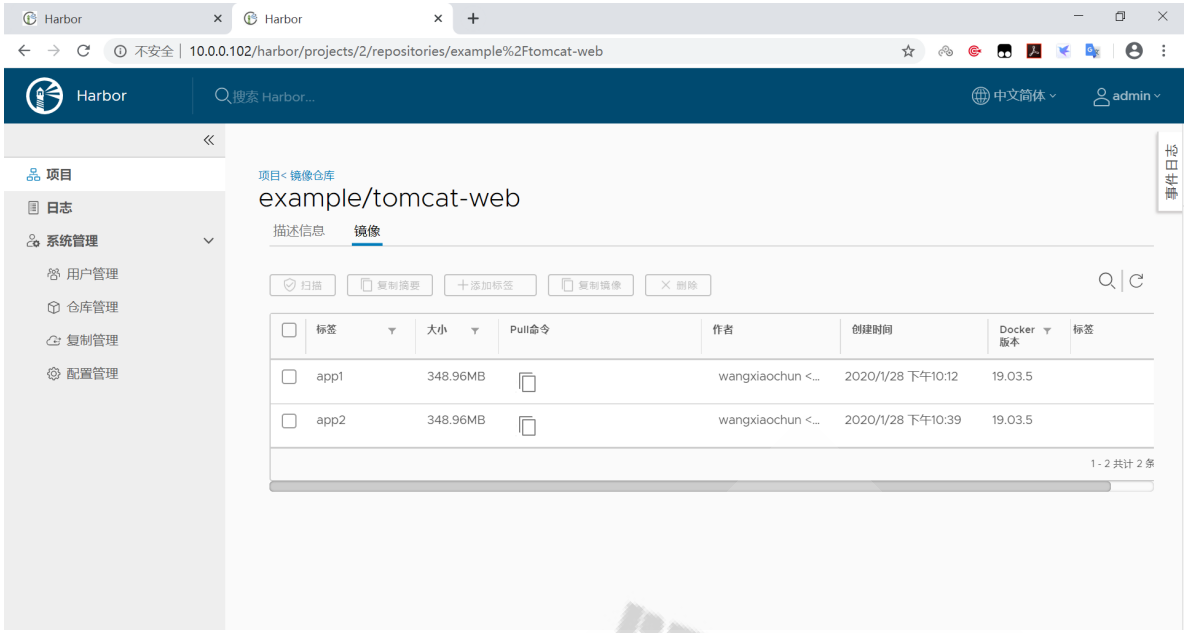
也可以查看到同步日志



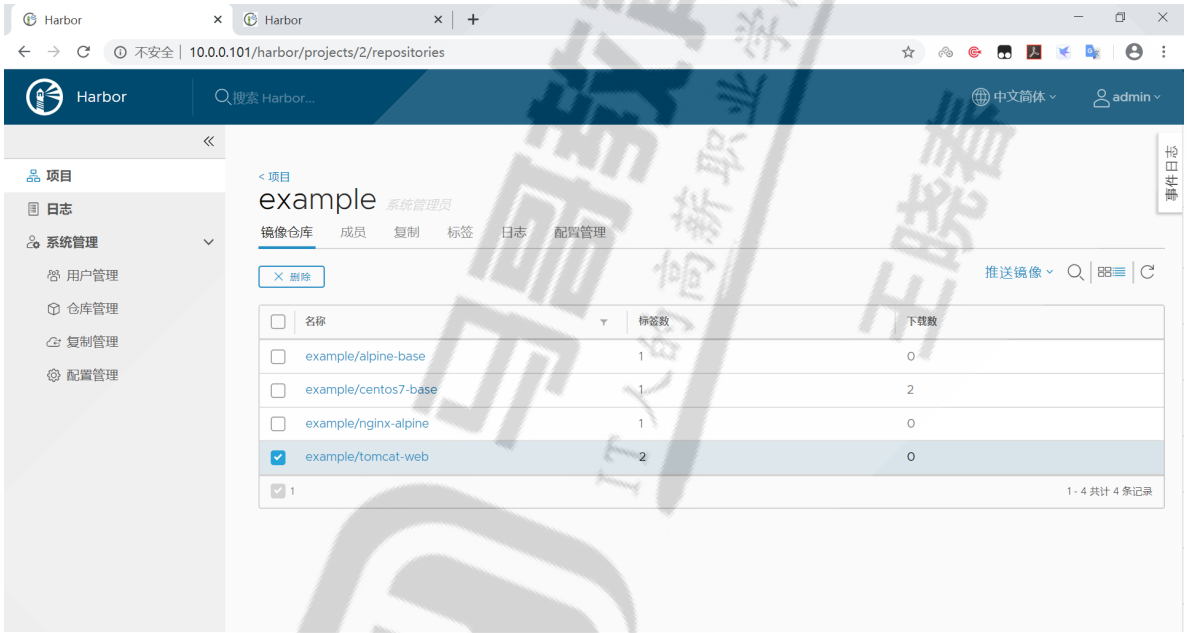
5.4.4.7 上传镜像观察是否可以双高同步

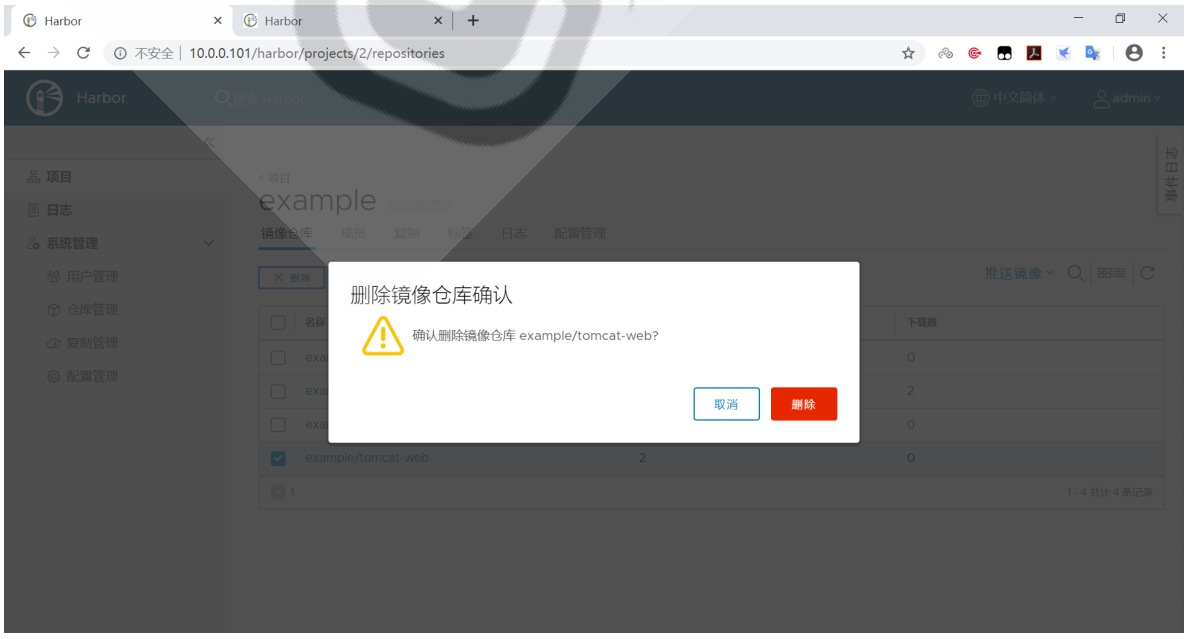
```
[root@ubuntu1804 ~]#docker tag tomcat-web:app1 10.0.0.101/example/tomcat-web:app1
[root@ubuntu1804 ~]#docker push 10.0.0.101/example/tomcat-web:app1
[root@ubuntu1804 ~]#docker tag tomcat-web:app2 10.0.0.102/example/tomcat-web:app2
[root@ubuntu1804 ~]#docker push 10.0.0.102/example/tomcat-web:app2
```

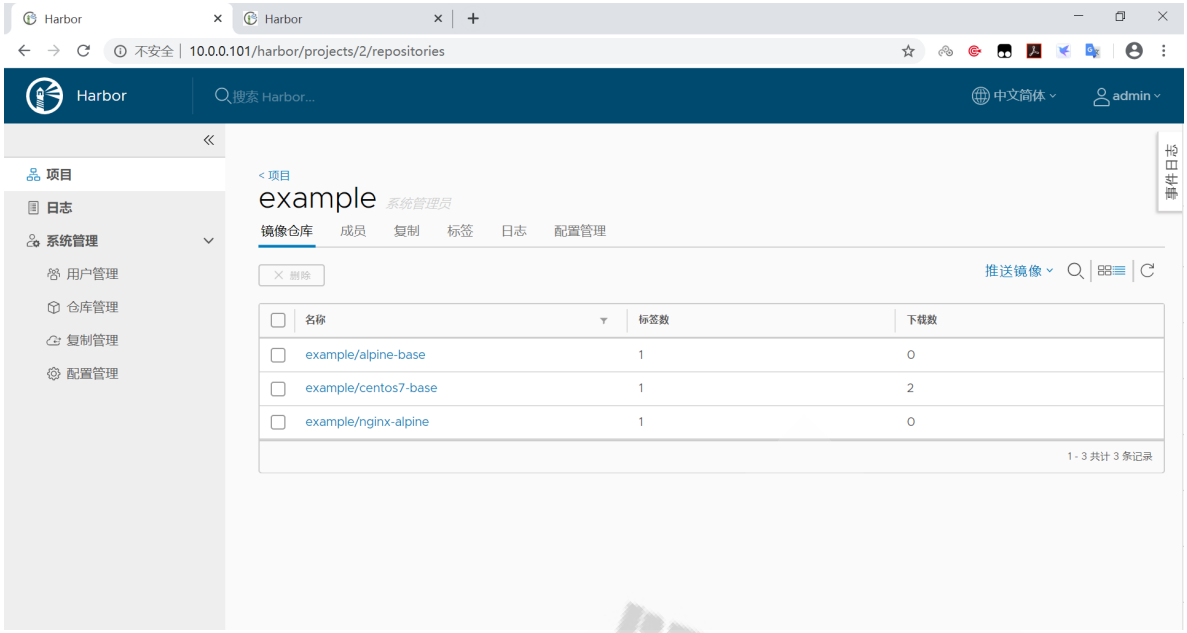


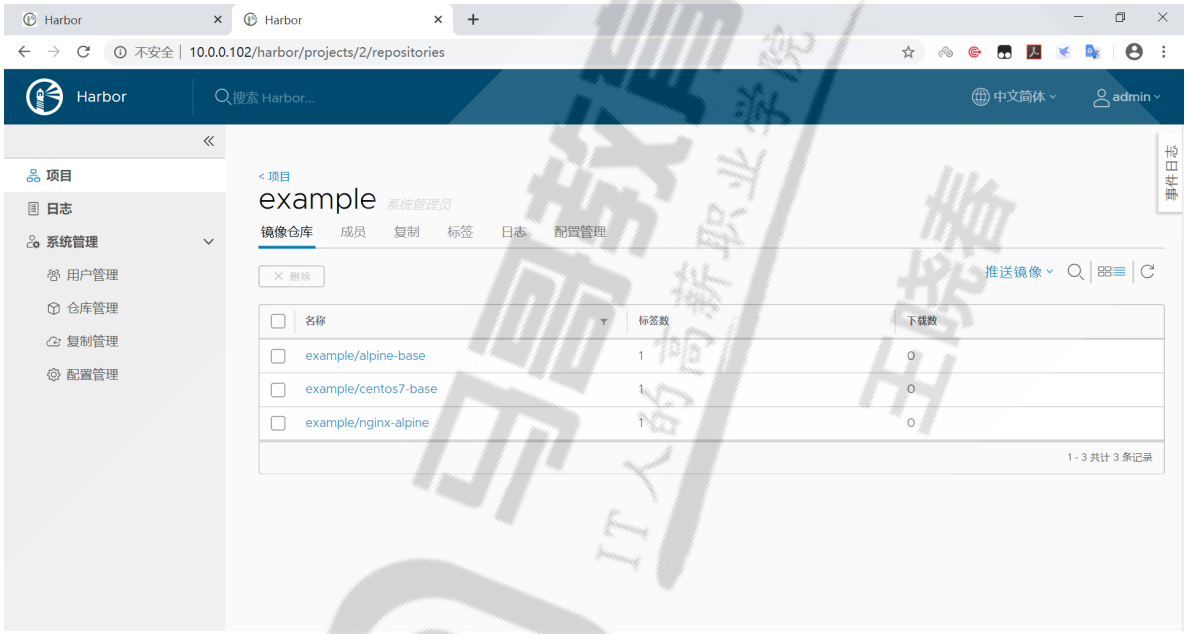


5.4.4.8 删除镜像观察是否可自动同步









5.4.5 harbor 安全 https 配置

harbor默认使用http,为了安全，可以使用https

5.4.5.1 实现Harbor的 https 认证

```
#安装docker
[root@ubuntu1804 ~]#bash install_docker_for_ubuntu1804.sh

#安装docker compose
[root@ubuntu1804 ~]#curl -L
https://github.com/docker/compose/releases/download/1.25.3/docker-compose-`uname
-s`-`uname -m` -o /usr/local/bin/docker-compose
[root@ubuntu1804 ~]#chmod +x /usr/local/bin/docker-compose
[root@ubuntu1804 ~]#docker-compose  --version
docker-compose version 1.25.3, build d4d1b42b


#下载harbor离线安装包且解压缩
```

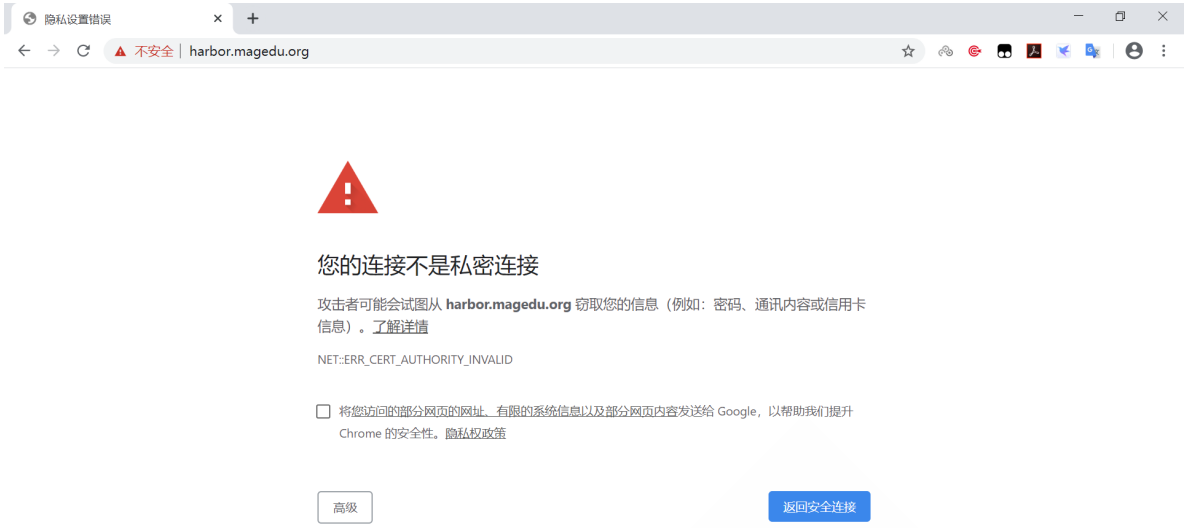
```
[root@ubuntu1804 ~]#wget https://storage.googleapis.com/harbor-releases/release-1.7.0/harbor-offline-installer-v1.7.6.tgz
[root@ubuntu1804 ~]#mkdir /apps
[root@ubuntu1804 ~]#tar xvf harbor-offline-installer-v1.7.6.tgz  -C /apps/

#生成私钥和证书
[root@ubuntu1804 ~]#touch /root/.rnd
[root@ubuntu1804 ~]#mkdir /apps/harbor/certs/
[root@ubuntu1804 ~]#cd /apps/harbor/certs/

#生成CA证书
[root@ubuntu1804 certs]#openssl req  -newkey rsa:4096 -nodes -sha256 -keyout ca.key -x509  -subj "/CN=ca.magedu.org" -days 365 -out ca.crt

#生成harbor主机的证书申请
[root@ubuntu1804 certs]#openssl req  -newkey rsa:4096 -nodes -sha256 -subj "/CN=harbor.magedu.org" -keyout harbor.magedu.org.key -out harbor.magedu.org.csr

#给harbor主机颁发证书
[root@ubuntu1804 certs]#openssl x509 -req -in harbor.magedu.org.csr -CA ca.crt -CAkey ca.key -CAcreateserial  -out harbor.magedu.org.crt

[root@ubuntu1804 ~]#tree /apps/harbor/certs
/apps/harbor/certs
├── ca.crt
├── ca.key
├── ca.srl
├── harbor.magedu.org.crt
├── harbor.magedu.org.csr
└── harbor.magedu.org.key

0 directories, 6 files
[root@ubuntu1804 ~]#vim /apps/harbor/harbor.cfg
hostname = harbor.magedu.org
ui_url_protocol = https
ssl_cert = /apps/harbor/certs/harbor.magedu.org.crt
ssl_cert_key = /apps/harbor/certs/harbor.magedu.org.key
harbor_admin_password = 123456

[root@ubuntu1804 ~]#apt -y install python
[root@ubuntu1804 ~]#/apps/harbor/install.sh
```

5.4.5.2 用https方式访问harbor网站

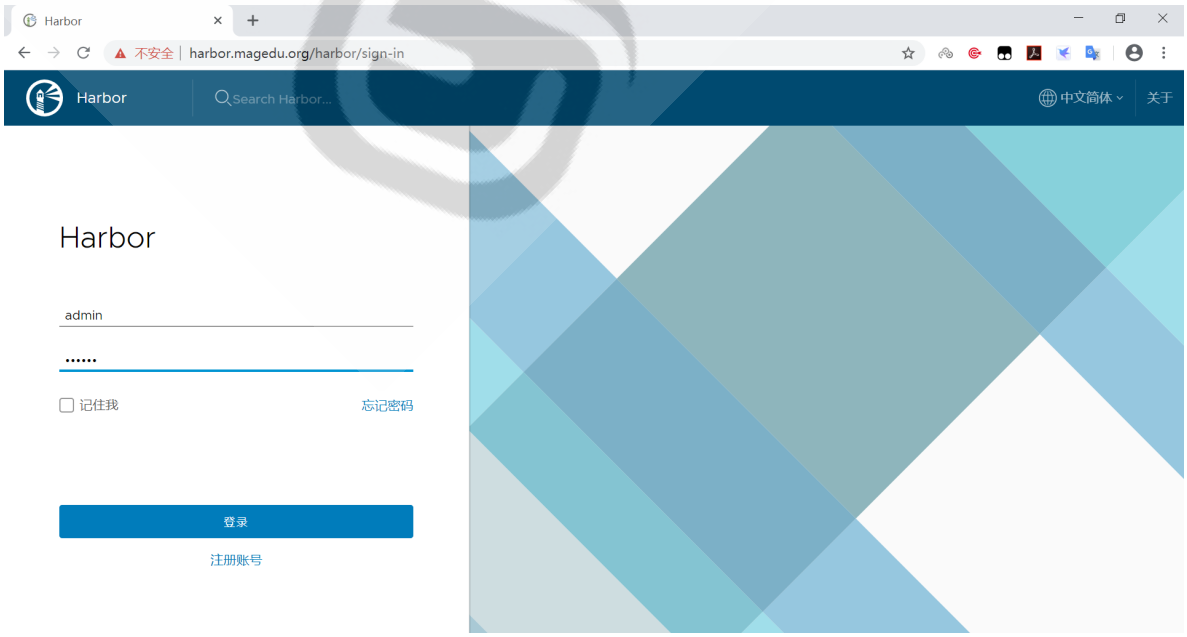
修改/etc/hosts文件

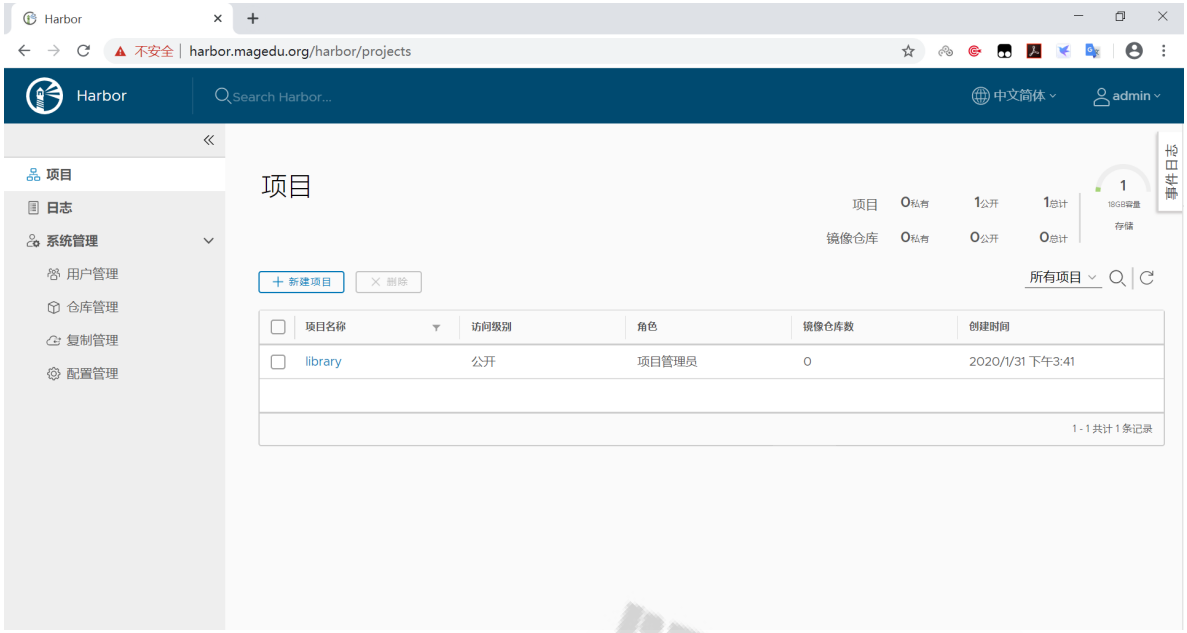
```
10.0.0.103 harbor.magedu.org
```

打开浏览器，访问http://harbor.magedu.org ,可以看到以下界面

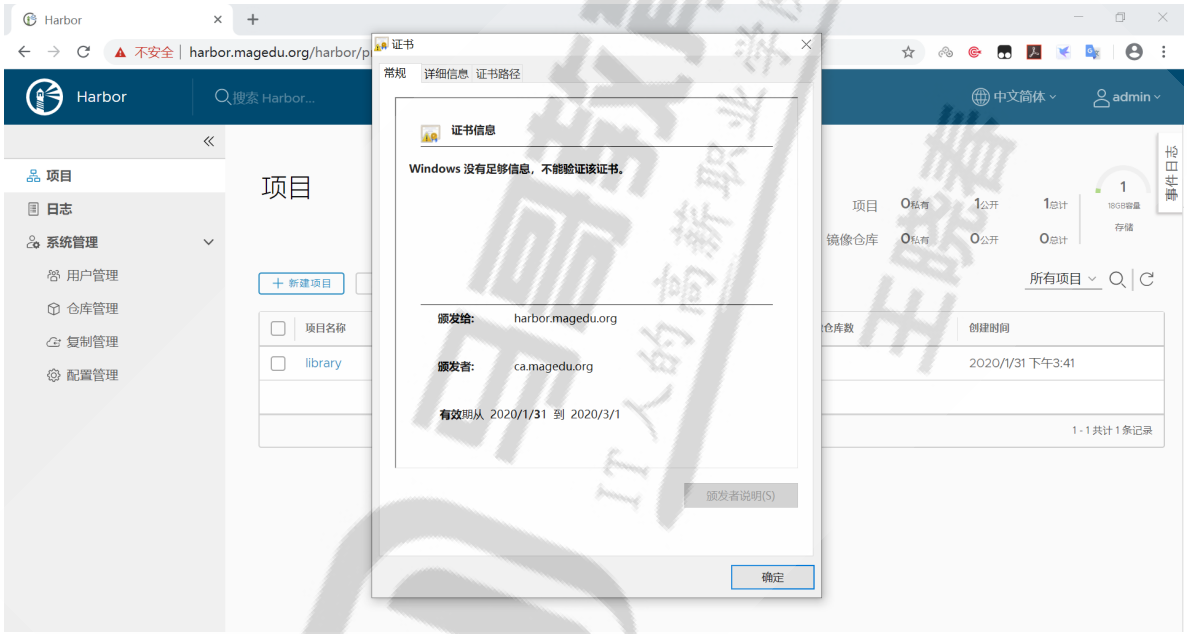

隐私设置错误
不安全 | harbor.magedu.org
您的连接不是私密连接
攻击者可能会试图从 harbor.magedu.org 窃取您的信息（例如：密码、通讯内容或信用卡信息）。了解详情
NET::ERR_CERT_AUTHORITY_INVALID
将您访问的部分网页的网址、有限的系统信息以及部分网页内容发送给 Google，以帮助我们提升 Chrome 的安全性。隐私权政策
高级
返回安全连接


隐私设置错误
不安全 | harbor.magedu.org
您的连接不是私密连接
攻击者可能会试图从 harbor.magedu.org 窃取您的信息（例如：密码、通讯内容或信用卡信息）。了解详情
NET::ERR_CERT_AUTHORITY_INVALID
将您访问的部分网页的网址、有限的系统信息以及部分网页内容发送给 Google，以帮助我们提升 Chrome 的安全性。隐私权政策
隐藏详情
返回安全连接
此服务器无法证明它是harbor.magedu.org；您计算机的操作系统不信任其安全证书。出现此问题的原因可能是配置有误或您的连接被拦截了。
继续前往harbor.magedu.org（不安全）

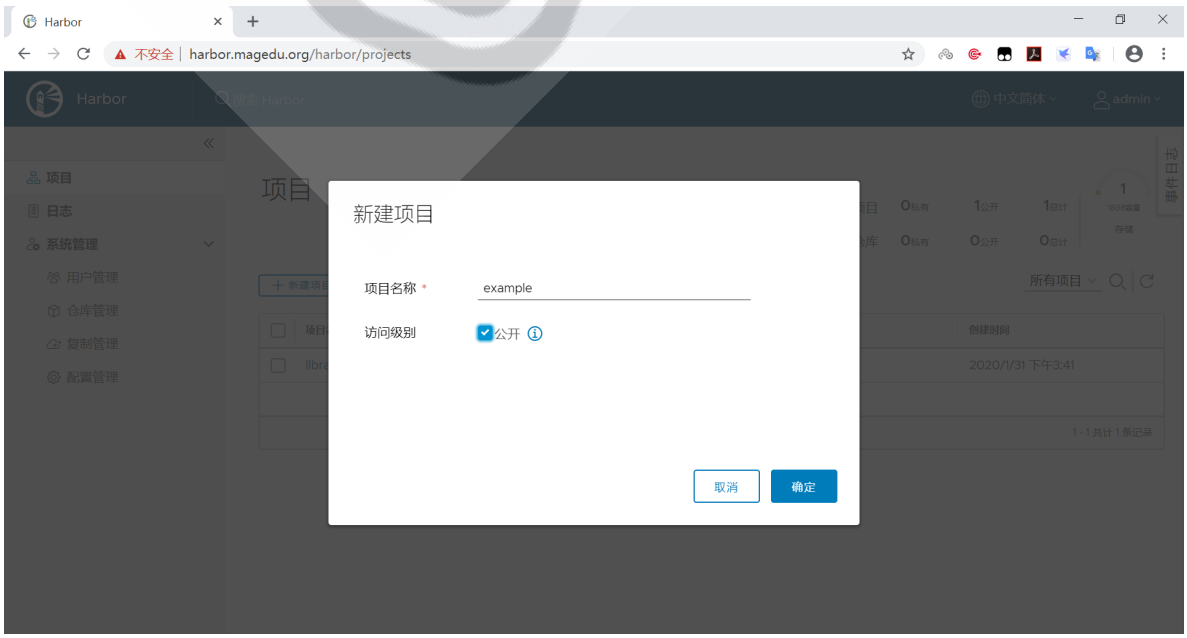

Harbor
不安全 | harbor.magedu.org/harbor/sign-in
Harbor
Search Harbor...
中文简体
关于
Harbor
admin
记住我
忘记密码
登录
注册账号



查看证书



5.4.5.3 在harbor网站新建项目



5.2.5.4 在客户端下载CA的证书

直接登录和上传下载镜像会报错

```
[root@ubuntu1804 ~]#vim /etc/hosts
10.0.0.103 harbor.magedu.org


[root@ubuntu1804 ~]#docker login harbor.magedu.org
Username: admin
Password:
Error response from daemon: Get https://harbor.magedu.org/v2/: x509: certificate
signed by unknown authority
```

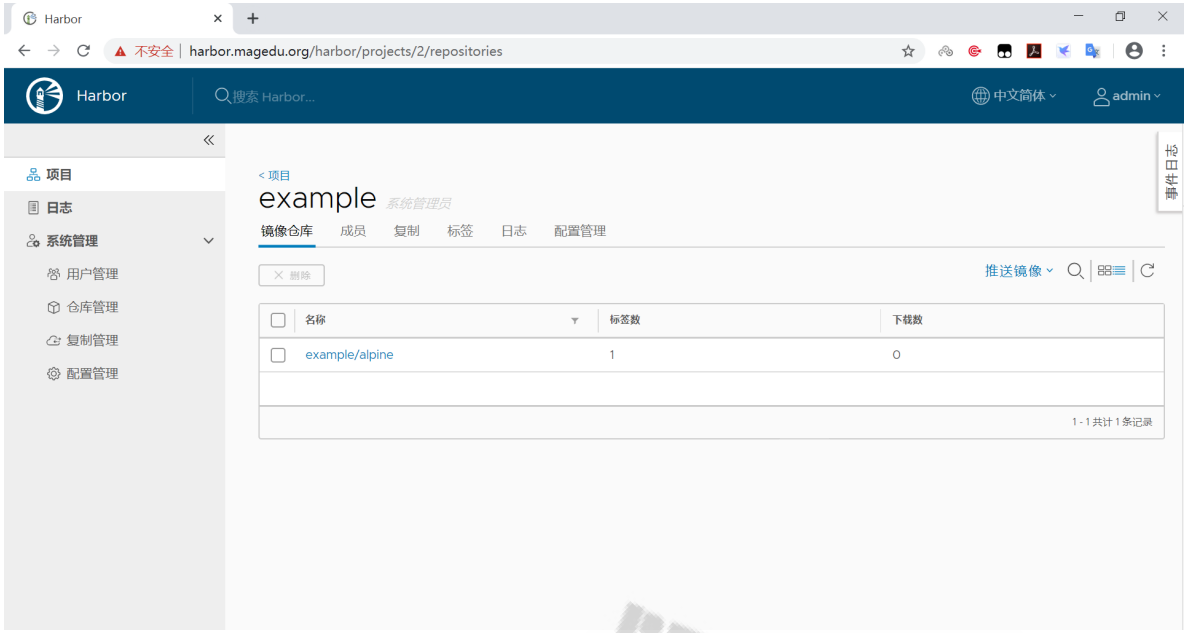
在客户端下载ca的证书

```
[root@ubuntu1804 ~]#mkdir -pv /etc/docker/certs.d/harbor.magedu.org/
[root@ubuntu1804 ~]#scp -r harbor.magedu.org:/apps/harbor/certs/ca.crt
/etc/docker/certs.d/harbor.magedu.org/
[root@ubuntu1804 ~]#tree /etc/docker/certs.d/
/etc/docker/certs.d/
└── harbor.magedu.org
    └── ca.crt

1 directory, 1 file
```

5.2.5.5 从客户端上传镜像

```
[root@ubuntu1804 ~]#docker tag alpine:3.11
harbor.magedu.org/example/alpine:3.11
[root@ubuntu1804 ~]#docker push harbor.magedu.org/example/alpine:3.11
The push refers to repository [harbor.magedu.org/example/alpine]
5216338b40a7: Pushed
3.11: digest:
sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45 size:
528
```

在harbor网站上验证上传的镜像

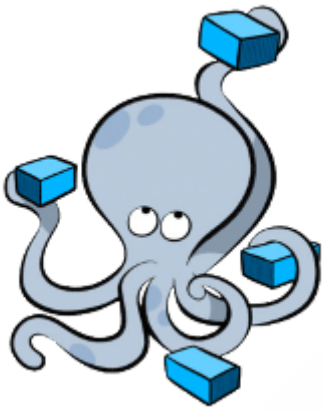


5.4.5.6 在客户端下载镜像

```
[root@centos7 ~]#vim /etc/hosts
10.0.0.103 harbor.magedu.org
[root@centos7 ~]#docker pull  harbor.magedu.org/example/alpine:3.11
Error response from daemon: Get https://harbor.magedu.org/v2/: x509: certificate
signed by unknown authority

[root@centos7 ~]#mkdir -pv /etc/docker/certs.d/harbor.magedu.org/
[root@centos7 ~]#scp -r harbor.magedu.org:/apps/harbor/certs/ca.crt
/etc/docker/certs.d/harbor.magedu.org/
[root@centos7 ~]#tree /etc/docker/certs.d/
/etc/docker/certs.d/
└── harbor.magedu.org
    └── ca.crt

1 directory, 1 file
[root@centos7 ~]#docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
[root@centos7 ~]#docker pull  harbor.magedu.org/example/alpine:3.11
3.11: Pulling from example/alpine
c9b1b535fdd9: Pull complete
Digest: sha256:ddba4d27a7ffc3f86dd6c2f92041af252a1f23a8e742c90e6e1297bfa1bc0c45
Status: Downloaded newer image for harbor.magedu.org/example/alpine:3.11
harbor.magedu.org/example/alpine:3.11
[root@centos7 ~]#docker images
REPOSITORY                          TAG                 IMAGE ID
CREATED             SIZE
harbor.magedu.org/example/alpine    3.11                e7d92cdc71fe        13
days ago         5.59MB
```

6 单机编排之Docker Compose

6.1 Docker Compse介绍



当在宿主机启动较多的容器时候，如果都是手动操作会觉得比较麻烦而且容易出错，此时推荐使用docker 单机编排工具 docker-compose

docker-compose 是 docker 容器的一种单机编排服务，docker-compose 是一个管理多个容器的工具，比如：可以解决容器之间的依赖关系，就像启动一个nginx 前端服务的时候会调用后端的tomcat，那就得先启动tomcat，但是启动tomcat 容器还需要依赖数据库，那就还得先启动数据库，docker-compose 可以用来解决这样的嵌套依赖关系，并且可以替代docker命令对容器进行创建、启动和停止等手工的操作

因此，如果说docker命令就像linux的命令，docker compse就像shell脚本，可以自动的执行容器批量操作，从而实现自动化的容器管理，或者说docker命令相当于ansible命令，那么docker compose文件，就相当于ansible-playbook的yaml文件

docker-compose 项目是Docker 官方的开源项目，负责实现对Docker 容器集群的快速编排，docker-compose 将所管理的容器分为三层，分别是工程（project），服务（service）以及容器（container）

github地址：https://github.com/docker/compose

官方地址：https://docs.docker.com/compose/

6.2 安装和准备

6.2.1 安装Docker Compose

6.2.1.1 方法1: 通过pip安装

python-pip 包将安装一个 pip 的命令，pip 命令是一个pyhton 安装包的安装工具，其类似于ubuntu的apt 或者 redhat 的yum，但是pip 只安装 python 相关的安装包，可以在多种操作系统安装和使用pip

此方式当前安装的版本较新，为docker_compose-1.25.3，推荐使用

```
Ubuntu:
# apt  update
# apt  install -y  python-pip

Centos:
# yum install epel-release
# yum install -y python-pip
# pip install --upgrade pip
```

范例:

```
[root@ubuntu1804 ~]#apt -y install python-pip
[root@ubuntu1804 ~]#pip install docker-compose
[root@ubuntu1804 ~]#docker-compose --version
docker-compose version 1.25.3, build unknown

#基于python3安装
[root@ubuntu1804 ~]#apt -y install python3-pip
[root@ubuntu1804 ~]#pip3 install docker-compose
[root@ubuntu1804 ~]#docker-compose --version
docker-compose version 1.27.4, build unknown
```

6.2.1.2 方法2: 直接从github下载安装对应版本

参看说明: https://github.com/docker/compose/releases

此方法安装版本可方便指定，推荐方法，但网络下载较慢

```
[root@ubuntu1804 ~]#curl -L
https://github.com/docker/compose/releases/download/1.25.3/docker-compose-`uname
-s`-`uname -m` -o /usr/local/bin/docker-compose
[root@ubuntu1804 ~]#chmod +x /usr/local/bin/docker-compose
```

6.2.1.3 方法3: 直接从包仓库安装

此方法安装的版本较旧，不推荐使用

```
#ubuntu安装
[root@ubuntu1804 ~]#apt -y install docker-compose
[root@ubuntu1804 ~]#docker-compose --version
docker-compose version 1.17.1, build unknown


#CentOS7安装，依赖EPEL源
[root@centos7 ~]#yum -y install docker-compose
[root@centos7 ~]#docker-compose --version
docker-compose version 1.18.0, buil 8dd22a9
```

6.2.2 查看命令格式

官方文档: https://docs.docker.com/compose/reference/

```
docker-compose --help
Define and run multi-container applications with Docker.
Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

#选项说明:
-f, --file FILE #指定Compose 模板文件，默认为docker-compose.yml
-p, --project-name NAME #指定项目名称，默认将使用当前所在目录名称作为项目名。
--verbose   #显示更多输出信息
--log-level LEVEL    #定义日志级别 (DEBUG, INFO, WARNING, ERROR, CRITICAL)
--no-ansi #不显示ANSI 控制字符
```

```
-v, --version #显示版本

#以下为命令选项，需要在docker-compose.yml|yaml 文件所在在目录里执行
build  #构建镜像
bundle #从当前docker compose 文件生成一个以<当前目录>为名称的json格式的Docker Bundle 备份文件
config  -q #查看当前配置，没有错误不输出任何信息
create #创建服务，较少使用
down #停止和删除所有容器、网络、镜像和卷
events #从容器接收实时事件，可以指定json 日志格式，较少使用
exec #进入指定容器进行操作
help #显示帮助细信息
images #显示镜像信息，较少使用
kill #强制终止运行中的容器
logs #查看容器的日志
pause #暂停服务
port #查看端口
ps #列出容器，较少使用
pull #重新拉取镜像，镜像发生变化后，需要重新拉取镜像，较少使用
push #上传镜像
restart #重启服务，较少使用
rm #删除已经停止的服务
run #一次性运行容器
scale  #设置指定服务运行的容器个数
start #启动服务 ，较少使用
stop #停止服务，较少使用
top #显示容器运行状态
unpause #取消暂定
up #创建并启动容器 ，较少使用
```

范例:

```
[root@ubuntu1804 ~]#docker-compose --version
docker-compose version 1.25.3, build d4d1b42b
```

6.2.3 docker compse 文件格式

官方文档: https://docs.docker.com/compose/compose-file/

docker compose 文件是一个yaml格式的文件，所以注意行首的缩进很严格

默认docker-compose命令会调用当前目录下的docker-compose.yml的文件，因此一般执行docker-compose命令前先进入docker-compose.yml文件所在目录

docker compose文件的格式很不同版本，版本不同，语法和格式有所不同，参看以下列表

Compose file format	Docker Engine release
3.7	18.06.0+
3.6	18.02.0+
3.5	17.12.0+
3.4	17.09.0+
3.3	17.06.0+
3.2	17.04.0+
3.1	1.13.1+
3.0	1.13.0+
2.4	17.12.0+
2.3	17.06.0+
2.2	1.13.0+
2.1	1.12.0+
2.0	1.10.0+
1.0	1.9.1.+

docker compose版本众多，以下通过具体示例说明docker compose的使用方法

6.3 从 docker compose 启动单个容器

注意：使用Docker compose之前，先要安装docker

6.3.1 创建 docker compose文件

docker compose 文件可在任意目录，创建文件名为docker-compose.yml 配置文件，要注意前后的缩进

```
[root@ubuntu1804 ~]#docker-compose --version
docker-compose version 1.25.4, build unknown
[root@ubuntu1804 ~]#mkdir /data/docker-compose
[root@ubuntu1804 ~]#cd /data/docker-compose
[root@ubuntu1804 docker-compose]#vim  docker-compose.yml
[root@ubuntu1804 docker-compose]#cat docker-compose.yml
service-nginx-web:
  image: 10.0.0.102/example/nginx-centos7-base:1.6.1
  container_name: nginx-web
  expose:
   - 80
   - 443
  ports:
    - "80:80"
    - "443:443"
```

6.3.2 查看配置和格式检查

```
[root@ubuntu1804 docker-compose]#docker-compose  config
services:
  service-nginx-web:
    container_name: nginx-web
    expose:
    - 80
    - 443
    image: 10.0.0.102/example/nginx-centos7-base:1.6.1
    network_mode: bridge
    ports:
    - 80:80/tcp
    - 443:443/tcp
version: '2.1'

[root@ubuntu1804 docker-compose]#docker-compose  config -q

#改错ocker-compose文件格式
[root@ubuntu1804 docker-compose]#vim docker-compose.yml
service-nginx-web   #改此行，最后的":"删除
  image: 10.0.0.102/example/nginx-centos7-base:1.6.1
  container_name: nginx-web
  expose:
   - 80
   - 443
  ports:
    - "80:80"
    - "443:443"

[root@ubuntu1804 docker-compose]#docker-compose  config -q
ERROR: yaml.scanner.ScannerError: mapping values are not allowed here
  in "./docker-compose.yml", line 2, column 8
[root@ubuntu1804 docker-compose]#
```

6.3.3 启动容器

注意：必须要在docker compose文件所在的目录执行

```
#前台启动
[root@ubuntu1804 docker-compose]#docker-compose up
Pulling service-nginx-web (10.0.0.102/example/nginx-centos7-base:1.6.1)...
ERROR: Get https://10.0.0.102/v2/: dial tcp 10.0.0.102:443: connect: connection
refused
[root@ubuntu1804 docker-compose]#vim /lib/systemd/system/docker.service
ExecStart=/usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
--insecure-registry 10.0.0.102
[root@ubuntu1804 docker-compose]#systemctl daemon-reload
[root@ubuntu1804 docker-compose]#systemctl restart docker
[root@ubuntu1804 docker-compose]#docker-compose up
Pulling service-nginx-web (10.0.0.102/example/nginx-centos7-base:1.6.1)...
1.6.1: Pulling from example/nginx-centos7-base
f34b00c7da20: Pull complete
544476d462f7: Pull complete
```

```
39345915aa1b: Pull complete
d5376f2bbd9e: Pull complete
4596aecee927: Pull complete
1617b995c379: Pull complete
d00df95be654: Pull complete
Digest: sha256:82e9e7d8bf65e160ba79a92bb25ae42cbbf791092d1e09fb7de25f91b31a21ff
Status: Downloaded newer image for 10.0.0.102/example/nginx-centos7-base:1.6.1
Creating nginx-web ... done
Attaching to nginx-web

#以上是前台执行不退出
```

6.3.4 验证docker compose执行结果

```
#上面命令是前台执行，所以要查看结果，可以再开一个终端窗口进行观察
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                                                 COMMAND
           CREATED             STATUS              PORTS
             NAMES
d71030504f6a        10.0.0.102/example/nginx-centos7-base:1.6.1
"/apps/nginx/sbin/ng…"   15 seconds ago      Up 13 seconds       0.0.0.0:80-
>80/tcp, 0.0.0.0:443->443/tcp   nginx-web
[root@ubuntu1804 ~]#docker-compose ps
ERROR:
        Can't find a suitable configuration file in this directory or any
        parent. Are you in the right directory?

        Supported filenames: docker-compose.yml, docker-compose.yaml

[root@ubuntu1804 ~]#cd /data/docker-compose/
[root@ubuntu1804 docker-compose]#docker-compose ps
  Name             Command           State                  Ports

------------------------------------------------------------------------------
-----
nginx-web   /apps/nginx/sbin/nginx   Up      0.0.0.0:443->443/tcp, 0.0.0.0:80-
>80/tcp

[root@ubuntu1804 docker-compose]#curl 127.0.0.1/app/
Test Page in app
[root@ubuntu1804 docker-compose]#docker-compose images
Container               Repository                  Tag      Image Id
Size
------------------------------------------------------------------------------
---
nginx-web   10.0.0.102/example/nginx-centos7-base   1.6.1   ea3840c349e5   413.4
MB

[root@ubuntu1804 docker-compose]#docker-compose exec service-nginx-web bash
[root@17c17ad30193 /]#
[root@17c17ad30193 /]#
[root@17c17ad30193 /]#
[root@17c17ad30193 /]# tail -f /apps/nginx/logs/access.log
172.17.0.1 - - [04/Feb/2020:16:01:42 +0800] "GET /app/ HTTP/1.1" 200 17 "-"
"curl/7.58.0"
10.0.0.101 - - [04/Feb/2020:16:06:29 +0800] "GET /app/ HTTP/1.1" 200 17 "-"
"curl/7.58.0"
```

```
10.0.0.102 - - [04/Feb/2020:16:08:22 +0800] "GET /app/ HTTP/1.1" 200 17 "-"
"curl/7.58.0"
```

6.3.5 结束前台执行

```
[root@ubuntu1804 docker-compose]#docker-compose up
Pulling service-nginx-web (10.0.0.102/example/nginx-centos7-base:1.6.1)...
1.6.1: Pulling from example/nginx-centos7-base
f34b00c7da20: Pull complete
544476d462f7: Pull complete
39345915aa1b: Pull complete
d5376f2bbd9e: Pull complete
4596aecee927: Pull complete
1617b995c379: Pull complete
d00df95be654: Pull complete
Digest: sha256:82e9e7d8bf65e160ba79a92bb25ae42cbbf791092d1e09fb7de25f91b31a21ff
Status: Downloaded newer image for 10.0.0.102/example/nginx-centos7-base:1.6.1
Creating nginx-web ... done
Attaching to nginx-web
^CGracefully stopping... (press Ctrl+C again to force)  #ctrl+c键，结束容器
Stopping nginx-web ... done

[root@ubuntu1804 docker-compose]#docker-compose ps
  Name              Command           State    Ports
------------------------------------------------------
nginx-web   /apps/nginx/sbin/nginx   Exit 0
[root@ubuntu1804 docker-compose]#docker ps -a
CONTAINER ID        IMAGE                                                COMMAND
           CREATED             STATUS                          PORTS
  NAMES
d71030504f6a        10.0.0.102/example/nginx-centos7-base:1.6.1
"/apps/nginx/sbin/ng…"   5 minutes ago       Exited (0) About a minute ago
                nginx-web

[root@ubuntu1804 docker-compose]#docker-compose start
Starting service-nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose ps
  Name              Command           State                    Ports

------------------------------------------------------------------------------
-----
nginx-web   /apps/nginx/sbin/nginx   Up      0.0.0.0:443->443/tcp, 0.0.0.0:80-
>80/tcp

#关闭容器
[root@ubuntu1804 docker-compose]#docker-compose kill
Killing nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose ps
  Name              Command            State     Ports
-------------------------------------------------------
nginx-web   /apps/nginx/sbin/nginx   Exit 137
```

6.3.6 删除容器

```
[root@ubuntu1804 docker-compose]#docker-compose ps
  Name              Command           State    Ports
```

```
------------------------------------------------
nginx-web   /apps/nginx/sbin/nginx   Exit 0

#只删除停止的容器
[root@ubuntu1804 docker-compose]#docker-compose rm
Going to remove nginx-web
Are you sure? [yN] y
Removing nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose up -d
Creating nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose rm
No stopped containers

#停止并删除容器及镜像
[root@ubuntu1804 docker-compose]#docker-compose down
Stopping nginx-web ... done
Removing nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose ps
Name   Command   State   Ports
------------------------------
[root@ubuntu1804 docker-compose]#docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED
STATUS              PORTS               NAMES
#也会自动删除镜像
[root@ubuntu1804 docker-compose]#docker-compose images
Container   Repository   Tag   Image Id   Size
------------------------------------------------
```

6.3.7 后台执行

```
[root@ubuntu1804 docker-compose]#docker-compose up -d
Creating nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose ps
  Name             Command          State                    Ports

-------------------------------------------------------------------------------
-----
nginx-web   /apps/nginx/sbin/nginx   Up      0.0.0.0:443->443/tcp, 0.0.0.0:80-
>80/tcp
[root@ubuntu1804 docker-compose]#curl 127.0.0.1/app/
Test Page in app
[root@ubuntu1804 docker-compose]#curl http://127.0.0.1/app/
Test Page in app
```

6.3.8 停止和启动与日志查看

```
[root@ubuntu1804 docker-compose]#docker-compose stop
Stopping nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose ps
  Name             Command          State    Ports
------------------------------------------------
nginx-web   /apps/nginx/sbin/nginx   Exit 0
[root@ubuntu1804 docker-compose]#docker-compose start
Starting service-nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose ps
```

```
  Name              Command           State                    Ports

-------------------------------------------------------------------------------
-----
nginx-web   /apps/nginx/sbin/nginx   Up      0.0.0.0:443->443/tcp, 0.0.0.0:80-
>80/tcp

[root@ubuntu1804 docker-compose]#docker-compose  restart
Restarting nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose  ps
  Name              Command           State                    Ports

-------------------------------------------------------------------------------
-----
nginx-web   /apps/nginx/sbin/nginx   Up      0.0.0.0:443->443/tcp, 0.0.0.0:80-
>80/tcp

#执行上面操作时，可以同时开一个终端，观察日事件
[root@ubuntu1804 docker-compose]#docker-compose events
2020-02-04 15:38:13.253822 container kill
5d92e4da8679a973145e5b4db364ae8cf8596a03c4fd0b3b6a28213a2f155be6
(image=10.0.0.102/example/nginx-centos7-base:1.6.1, name=nginx-web)
2020-02-04 15:38:13.531208 container die
5d92e4da8679a973145e5b4db364ae8cf8596a03c4fd0b3b6a28213a2f155be6
(image=10.0.0.102/example/nginx-centos7-base:1.6.1, name=nginx-web)
2020-02-04 15:38:13.631137 container stop
5d92e4da8679a973145e5b4db364ae8cf8596a03c4fd0b3b6a28213a2f155be6
(image=10.0.0.102/example/nginx-centos7-base:1.6.1, name=nginx-web)
2020-02-04 15:38:15.137495 container start
5d92e4da8679a973145e5b4db364ae8cf8596a03c4fd0b3b6a28213a2f155be6
(image=10.0.0.102/example/nginx-centos7-base:1.6.1, name=nginx-web)
2020-02-04 15:38:15.137546 container restart
5d92e4da8679a973145e5b4db364ae8cf8596a03c4fd0b3b6a28213a2f155be6
(image=10.0.0.102/example/nginx-centos7-base:1.6.1, name=nginx-web)

#以json格式显示日志
[root@ubuntu1804 docker-compose]#docker-compose events --json
{"time": "2020-02-04T15:48:22.423539", "type": "container", "action": "kill",
"id": "19d72e9bc85842d8879d7dcf2a3d2defd79a5a0c3c3d974ddfbbbc6e95bf910b",
"service": "service-nginx-web", "attributes": {"name": "nginx-web", "image":
"10.0.0.102/example/nginx-centos7-base:1.6.1"}}
{"time": "2020-02-04T15:48:22.537200", "type": "container", "action":
"exec_die", "id":
"19d72e9bc85842d8879d7dcf2a3d2defd79a5a0c3c3d974ddfbbbc6e95bf910b", "service":
"service-nginx-web", "attributes": {"name": "nginx-web", "image":
"10.0.0.102/example/nginx-centos7-base:1.6.1"}}
{"time": "2020-02-04T15:48:22.745670", "type": "container", "action": "die",
"id": "19d72e9bc85842d8879d7dcf2a3d2defd79a5a0c3c3d974ddfbbbc6e95bf910b",
"service": "service-nginx-web", "attributes": {"name": "nginx-web", "image":
"10.0.0.102/example/nginx-centos7-base:1.6.1"}}
{"time": "2020-02-04T15:48:22.863375", "type": "container", "action": "stop",
"id": "19d72e9bc85842d8879d7dcf2a3d2defd79a5a0c3c3d974ddfbbbc6e95bf910b",
"service": "service-nginx-web", "attributes": {"name": "nginx-web", "image":
"10.0.0.102/example/nginx-centos7-base:1.6.1"}}
{"time": "2020-02-04T15:48:23.979421", "type": "container", "action": "start",
"id": "19d72e9bc85842d8879d7dcf2a3d2defd79a5a0c3c3d974ddfbbbc6e95bf910b",
"service": "service-nginx-web", "attributes": {"name": "nginx-web", "image":
"10.0.0.102/example/nginx-centos7-base:1.6.1"}}
```

```
{"time": "2020-02-04T15:48:23.979468", "type": "container", "action": "restart",
"id": "19d72e9bc85842d8879d7dcf2a3d2defd79a5a0c3c3d974ddfbbbc6e95bf910b",
"service": "service-nginx-web", "attributes": {"name": "nginx-web", "image":
"10.0.0.102/example/nginx-centos7-base:1.6.1"}}
```

6.3.9 暂停和恢复

```
[root@ubuntu1804 docker-compose]#docker-compose  pause
Pausing nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose  ps
  Name              Command           State                    Ports

------------------------------------------------------------------------------
------
nginx-web   /apps/nginx/sbin/nginx   Paused   0.0.0.0:443->443/tcp, 0.0.0.0:80-
>80/tcp
[root@ubuntu1804 docker-compose]#curl -m 1  http://127.0.0.1/app/
curl: (28) Operation timed out after 1002 milliseconds with 0 bytes received
[root@ubuntu1804 docker-compose]#docker-compose  unpause
Unpausing nginx-web ... done
[root@ubuntu1804 docker-compose]#docker-compose  ps
  Name              Command           State                Ports

------------------------------------------------------------------------------
-----
nginx-web   /apps/nginx/sbin/nginx   Up      0.0.0.0:443->443/tcp, 0.0.0.0:80-
>80/tcp
[root@ubuntu1804 docker-compose]#curl -m 1  http://127.0.0.1/app/
Test Page in app
```

6.3.10 指定同时启动容器的数量

```
[root@ubuntu1804 docker-compose]#vim docker-compose.yml
[root@ubuntu1804 docker-compose]#cat docker-compose.yml
service-nginx-web:
  image: 10.0.0.102/example/nginx-centos7-base:1.6.1
#  container_name: nginx-web #同时启动多个同一镜像的容器，不要指定容器名称，否则会冲突
  expose:
   - 80
   - 443
#  ports:                    #同时启动多个同一镜像的容器，不要指定端口号，否则会冲突
#    - "80:80"
#    - "443:443"

#再加一个service
service-tomcat:
  image: 10.0.0.102/example/tomcat-base:v8.5.50

[root@ubuntu1804 docker-compose]#docker-compose  ps
Name   Command   State   Ports
------------------------------
[root@ubuntu1804 docker-compose]#docker-compose  up -d --scale service-nginx-
web=2
Creating docker-compose_service-tomcat_1    ... done
Creating docker-compose_service-nginx-web_1 ... done
```

```
Creating docker-compose_service-nginx-web_2 ... done
[root@ubuntu1804 docker-compose]#docker-compose  ps
               Name                             Command            State
Ports
-------------------------------------------------------------------------------
------
docker-compose_service-nginx-web_1   /apps/nginx/sbin/nginx   Up       443/tcp,
80/tcp
docker-compose_service-nginx-web_2   /apps/nginx/sbin/nginx   Up       443/tcp,
80/tcp
docker-compose_service-tomcat_1      /bin/bash                Exit 0

[root@ubuntu1804 docker-compose]#docker-compose  up -d --scale service-nginx-
web=3 --scale service-tomcat=2
Starting docker-compose_service-tomcat_1    ... done
Starting docker-compose_service-nginx-web_1 ... done
Starting docker-compose_service-nginx-web_2 ... done
Creating docker-compose_service-nginx-web_3 ... done
Creating docker-compose_service-tomcat_2    ... done
[root@ubuntu1804 docker-compose]#docker-compose  ps
               Name                             Command            State
Ports
-------------------------------------------------------------------------------
------
docker-compose_service-nginx-web_1   /apps/nginx/sbin/nginx   Up       443/tcp,
80/tcp
docker-compose_service-nginx-web_2   /apps/nginx/sbin/nginx   Up       443/tcp,
80/tcp
docker-compose_service-nginx-web_3   /apps/nginx/sbin/nginx   Up       443/tcp,
80/tcp
docker-compose_service-tomcat_1      /bin/bash                Exit 0

docker-compose_service-tomcat_2      /bin/bash                Exit 0

[root@ubuntu1804 docker-compose]#docker-compose  up -d
Stopping and removing docker-compose_service-nginx-web_2 ... done
Stopping and removing docker-compose_service-nginx-web_3 ... done
Stopping and removing docker-compose_service-tomcat_2    ... done
Starting docker-compose_service-tomcat_1                 ... done
Starting docker-compose_service-nginx-web_1              ... done
[root@ubuntu1804 docker-compose]#docker-compose  ps
               Name                             Command            State
Ports
-------------------------------------------------------------------------------
------
docker-compose_service-nginx-web_1   /apps/nginx/sbin/nginx   Up       443/tcp,
80/tcp
docker-compose_service-tomcat_1      /bin/bash                Exit 0
```

6.4 从docker compose启动多个容器

6.4.1 编辑docker-compose文件并使用数据卷

注意: 同一个文件， 数据卷的优先级比镜像内的文件优先级高

```
[root@ubuntu1804 docker-compose]#vim docker-compose.yml
[root@ubuntu1804 docker-compose]#cat  docker-compose.yml
service-nginx-web:
  image: 10.0.0.102/example/nginx-centos7-base:1.6.1
  container_name: nginx-web
  volumes:
    - /data/nginx:/apps/nginx/html/#指定数据卷，将宿主机/data/nginx挂载到容
器/apps/nginx/html
  expose:
   - 80
   - 443
  ports:
    - "80:80"
    - "443:443"

service-tomcat-app1:
  image: 10.0.0.102/example/tomcat-web:app1
  container_name: tomcat-app1
  expose:
    - 8080
  ports:
    - "8081:8080"


service-tomcat-app2:
  image: 10.0.0.102/example/tomcat-web:app2
  container_name: tomcat-app2
  expose:
    - 8080
  ports:
    - "8082:8080"

#在宿主机准备nginx测试页面文件
[root@ubuntu1804 docker-compose]#mkdir /data/nginx
[root@ubuntu1804 docker-compose]#echo Docker compose test page >
/data/nginx/index.html
```

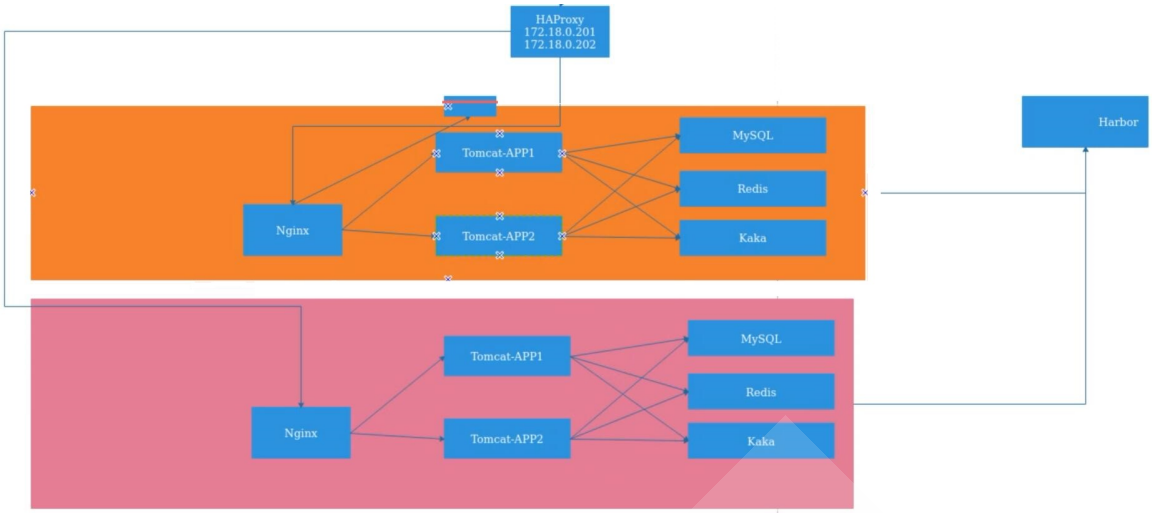
6.4.2 启动容器并验证结果

```
[root@ubuntu1804 docker-compose]#docker-compose up -d
Pulling service-tomcat-app1 (10.0.0.102/example/tomcat-web:app1)...
app1: Pulling from example/tomcat-web
f34b00c7da20: Already exists
544476d462f7: Already exists
39345915aa1b: Already exists
4b792f2bae38: Already exists
4439447a3522: Already exists
fe34d2ec1dd0: Already exists
b8487ca03126: Already exists
5a475b7d8b1a: Already exists
df8703d3d2dd: Already exists
f0da1ffa7aa7: Pull complete
80fd4c70e670: Pull complete
c2a0247d7bfa: Pull complete
b0977ed809cd: Pull complete
Digest: sha256:e0aba904df6095ea04c594d6906101f8e5f4a6ceb0a8f9b24432c47698d0caa8
```

```
Status: Downloaded newer image for 10.0.0.102/example/tomcat-web:app1
Pulling service-tomcat-app2 (10.0.0.102/example/tomcat-web:app2)...
app2: Pulling from example/tomcat-web
f34b00c7da20: Already exists
544476d462f7: Already exists
39345915aa1b: Already exists
4b792f2bae38: Already exists
4439447a3522: Already exists
fe34d2ec1dd0: Already exists
b8487ca03126: Already exists
5a475b7d8b1a: Already exists
df8703d3d2dd: Already exists
f0da1ffa7aa7: Already exists
80fd4c70e670: Already exists
1a55cb76a801: Pull complete
565ab795f82a: Pull complete
Digest: sha256:c4d6f166c3933f6c1ba59c84ea0518ed653af25f28b87981c242b0deff4209bb
Status: Downloaded newer image for 10.0.0.102/example/tomcat-web:app2
Creating tomcat-app1 ... done
Creating tomcat-app2 ... done
Creating nginx-web   ... done

[root@ubuntu1804 docker-compose]#docker-compose  ps
   Name                 Command               State                 Ports

-------------------------------------------------------------------------------
---------------
nginx-web     /apps/nginx/sbin/nginx           Up      0.0.0.0:443->443/tcp,
0.0.0.0:80->80/tcp
tomcat-app1   /apps/tomcat/bin/run_tomcat.sh   Up      8009/tcp, 0.0.0.0:8081-
>8080/tcp
tomcat-app2   /apps/tomcat/bin/run_tomcat.sh   Up      8009/tcp, 0.0.0.0:8082-
>8080/tcp
[root@ubuntu1804 docker-compose]#curl http://127.0.0.1/
Docker compose test page
[root@ubuntu1804 docker-compose]#curl http://127.0.0.1:8081/app/
Tomcat Page in app1
[root@ubuntu1804 docker-compose]#curl http://127.0.0.1:8082/app/
Tomcat Page in app2
```

6.5 实战案例: 实现单机版的Haproxy+Nginx+Tomcat

编写 docker-compose.yml 文件，实现单机版本的 nginx+tomcat 的动静分离 web站点，要求从 nginx 作为访问入口，当访问指定 URL 的时候转发至 tomcat 服务器响应



6.5.1 制作haproxy镜像

6.5.1.1 编辑Dockerfile文件

```
[root@ubuntu1804 ~]#cat /data/dockerfile/web/haproxy/2.1.2-centos7-
base/Dockerfile
#Haproxy Base Image
FROM centos7-base:v1
LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"

ADD haproxy-2.1.2.tar.gz /usr/local/src/

RUN cd /usr/local/src/haproxy-2.1.2 \
    && make ARCH=x86_64 TARGET=linux-glibc USE_PCRE=1 USE_OPENSSL=1 USE_ZLIB=1
USE_SYSTEMD=1 USE_CPU_AFFINITY=1 PREFIX=/apps/haproxy \
    && make install PREFIX=/apps/haproxy \
    && ln -s /apps/haproxy/sbin/haproxy  /usr/sbin/ \
    && mkdir /apps/haproxy/run \
    && rm -rf /usr/local/src/haproxy*

ADD haproxy.cfg /etc/haproxy/
ADD run_haproxy.sh /usr/bin

EXPOSE 80 9999
CMD ["run_haproxy.sh"]
```

6.5.1.2 前台启动脚本

```
[root@ubuntu1804 ~]#cat /data/dockerfile/web/haproxy/2.1.2-centos7-
base/run_haproxy.sh
#!/bin/bash
haproxy -f /etc/haproxy/haproxy.cfg
tail -f /etc/hosts
```

6.5.1.3 haproxy参数文件

```
[root@ubuntu1804 ~]#cat /data/dockerfile/web/haproxy/2.1.2-centos7-
base/haproxy.cfg
```

```
global
chroot /apps/haproxy
#stats socket /var/lib/haproxy/haproxy.sock mode 600 level admin
uid 99
gid 99
daemon
nbproc 1
pidfile /apps/haproxy/run/haproxy.pid
log 127.0.0.1 local3 info
defaults
option http-keep-alive
option forwardfor
mode http
timeout connect 300000ms
timeout client 300000ms
timeout server 300000ms
listen stats
 mode http
 bind 0.0.0.0:9999
 stats enable
 log global
 stats uri    /haproxy-status
 stats auth   haadmin:123456

listen web_port
 bind 0.0.0.0:80
 mode http
 log global
 balance roundrobin
 server nginx-web service-nginx-web:80 check inter 3000 fall 2 rise 5
 #service-nginx-web是docker-compose.yml文件中使用的地址
```

6.5.1.4 镜像build和上传harbor的脚本

```
[root@ubuntu1804 ~]#cat /data/dockerfile/web/haproxy/2.1.2-centos7-base/build.sh
#!/bin/bash
#
docker build -t 10.0.0.102/example/haproxy-centos7-base:2.1.2 .
docker push 10.0.0.102/example/haproxy-centos7-base:2.1.2
```

6.5.1.5 准备压缩包及其他文件

```
[root@ubuntu1804 ~]#tree /data/dockerfile/web/haproxy/2.1.2-centos7-base/
/data/dockerfile/web/haproxy/2.1.2-centos7-base/
├── build.sh
├── Dockerfile
├── haproxy-2.1.2.tar.gz
├── haproxy.cfg
└── run_haproxy.sh

0 directories, 5 files
```

6.5.1.6 执行构建镜像并上传 barbor

```
[root@ubuntu1804 ~]#cd /data/dockerfile/web/haproxy/2.1.2-centos7-base/
```

```
[root@ubuntu1804 2.1.2-centos7-base]#bash build.sh
Sending build context to Docker daemon   2.67MB
Step 1/8 : FROM centos7-base:v1
 ---> 34ab3afcd3b3
Step 2/8 : LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
 ---> Using cache
 ---> 6206742174bc
Step 3/8 : ADD haproxy-2.1.2.tar.gz /usr/local/src/
 ---> Using cache
 ---> 21a146f4e75d
Step 4/8 : RUN cd /usr/local/src/haproxy-2.1.2     && make ARCH=x86_64
TARGET=linux-glibc USE_PCRE=1 USE_OPENSSL=1 USE_ZLIB=1 USE_SYSTEMD=1
USE_CPU_AFFINITY=1 PREFIX=/apps/haproxy     && make install PREFIX=/apps/haproxy
    && ln -s /apps/haproxy/sbin/haproxy  /usr/sbin/     && mkdir
/apps/haproxy/run     && rm -rf /usr/local/src/haproxy*
 ---> Using cache
 ---> 014f0f0b3569
Step 5/8 : ADD haproxy.cfg /etc/haproxy/
 ---> Using cache
 ---> 878e8db33c8d
Step 6/8 : ADD run_haproxy.sh /usr/bin
 ---> Using cache
 ---> 1177ed360989
Step 7/8 : EXPOSE 80 9999
 ---> Using cache
 ---> 8c2c602c1a56
Step 8/8 : CMD ["run_haproxy.sh"]
 ---> Using cache
 ---> 03e8086d441d
Successfully built 03e8086d441d
Successfully tagged 10.0.0.102/example/haproxy-centos7-base:2.1.2
The push refers to repository [10.0.0.102/example/haproxy-centos7-base]
fea7d7539b84: Layer already exists
118f303dfb57: Layer already exists
2748cd88bb93: Layer already exists
a5dbda9ecfbf: Layer already exists
2073413aebd6: Layer already exists
6ec9af97c369: Layer already exists
034f282942cd: Layer already exists
2.1.2: digest:
sha256:5df06247a5bd187894dd1c705f374568ecb4996e8d38bea71fd0bc679ab6ae13 size:
1785
```

6.5.2 准备nginx镜像

6.5.2.1 准备Dockfile文件

```
[root@ubuntu1804 ~]#cat /data/dockerfile/web/nginx/1.16.1-centos7-
base/Dockerfile
FROM centos7-base:v1

LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"

ADD nginx-1.16.1.tar.gz /usr/local/src
```

```
RUN cd /usr/local/src/nginx-1.16.1 \
    && ./configure --prefix=/apps/nginx \
    && make && make install \
    && rm -rf /usr/local/src/nginx-1.16.1*     \
    && useradd -r -s /sbin/nologin nginx

COPY nginx.conf /apps/nginx/conf/

ADD  app.tar.gz /apps/nginx/html/

EXPOSE 80 443

CMD ["/apps/nginx/sbin/nginx"]
```

6.5.2.2 准备nginx配置文件

```
[root@ubuntu1804 ~]#cat /data/dockerfile/web/nginx/1.16.1-centos7-
base/nginx.conf
user  nginx;
worker_processes  1;
daemon off;
events {
    worker_connections  1024;
}
http {
    upstream tomcat {
        server service-tomcat-app1:8080;  #service-tomcat-app1为docker-
compse.yml指定的名称
        server service-tomcat-app2:8080;
    }
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;
    server {
        listen       80;
        server_name  localhost;
        location / {
            root   html;
            index  index.html index.htm;
        }
        location /app {
            proxy_pass http://tomcat ;
        }
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
```

6.5.2.3 镜像build和上传harbor的脚本

```
[root@ubuntu1804 ~]#cat  /data/dockerfile/web/nginx/1.16.1-centos7-base/build.sh
#!/bin/bash
#
docker build -t 10.0.0.102/example/nginx-centos7-base:1.6.1 .
docker push 10.0.0.102/example/nginx-centos7-base:1.6.1
```

6.5.2.4 准备软件包和其它文件

```
[root@ubuntu1804 ~]#tree /data/dockerfile/web/nginx/1.16.1-centos7-base/
/data/dockerfile/web/nginx/1.16.1-centos7-base/
├── app.tar.gz
├── build.sh
├── Dockerfile
├── nginx-1.16.1.tar.gz
└── nginx.conf

0 directories, 5 files
[root@ubuntu1804 ~]#
```

6.5.2.5 执行构建镜像并上传harbor

```
[root@ubuntu1804 ~]#cd /data/dockerfile/web/nginx/1.16.1-centos7-base/
[root@ubuntu1804 1.16.1-centos7-base]#bash build.sh
Sending build context to Docker daemon  1.041MB
Step 1/8 : FROM centos7-base:v1
 ---> 34ab3afcd3b3
Step 2/8 : LABEL maintainer="wangxiaochun <root@wangxiaochun.com>"
 ---> Using cache
 ---> 6206742174bc
Step 3/8 : ADD nginx-1.16.1.tar.gz /usr/local/src
 ---> Using cache
 ---> 1c2d1cbbf34d
Step 4/8 : RUN cd /usr/local/src/nginx-1.16.1     && ./configure --
prefix=/apps/nginx     && make && make install     && rm -rf
/usr/local/src/nginx-1.16.1*         && useradd -r -s /sbin/nologin nginx
 ---> Using cache
 ---> 019823fa30bf
Step 5/8 : COPY nginx.conf /apps/nginx/conf/
 ---> Using cache
 ---> 9583c76d6c9d
Step 6/8 : ADD  app.tar.gz /apps/nginx/html/
 ---> Using cache
 ---> 4bc0528bb32e
Step 7/8 : EXPOSE 80 443
 ---> Using cache
 ---> 547fb553f9d9
Step 8/8 : CMD ["/apps/nginx/sbin/nginx"]
 ---> Using cache
 ---> f6aa9591a2b0
Successfully built f6aa9591a2b0
Successfully tagged 10.0.0.102/example/nginx-centos7-base:1.6.1
The push refers to repository [10.0.0.102/example/nginx-centos7-base]
4591f99cd5a6: Layer already exists
27bdf47a2126: Layer already exists
6aa26a3d91e1: Layer already exists
93ae7d74d90e: Layer already exists
```

```
2073413aebd6: Layer already exists
6ec9af97c369: Layer already exists
034f282942cd: Layer already exists
1.6.1: digest:
sha256:2eb7525c623ecf34bc2046888029cdb0ef01d866a0687ca7afd4dfd36c16a863 size:
1786
[root@ubuntu1804 1.16.1-centos7-base]#
```

6.5.3 准备tomcat镜像

此处略，参看前面的第2.4节

参考下面文件列表:

```
[root@ubuntu1804 ~]#tree /data/dockerfile/
/data/dockerfile/
├── system
│   ├── alpine
│   │   ├── build.sh
│   │   ├── Dockerfile
│   │   └── repositories
│   ├── centos
│   │   ├── build.sh
│   │   └── Dockerfile
│   ├── debian
│   └── ubuntu
└── web
    ├── apache
    ├── haproxy
    │   └── 2.1.2-centos7-base
    │       ├── build.sh
    │       ├── Dockerfile
    │       ├── haproxy-2.1.2.tar.gz
    │       ├── haproxy.cfg
    │       └── run_haproxy.sh
    ├── jdk
    │   ├── build.sh
    │   ├── Dockerfile
    │   ├── jdk-8u212-linux-x64.tar.gz
    │   └── profile
    ├── nginx
    │   ├── 1.16.1-alpine
    │   │   ├── build.sh
    │   │   ├── Dockerfile
    │   │   ├── index.html
    │   │   ├── nginx-1.16.1.tar.gz
    │   │   └── nginx.conf
    │   ├── 1.16.1-centos7
    │   │   ├── build.sh
    │   │   ├── Dockerfile
    │   │   ├── index.html
    │   │   ├── nginx-1.16.1.tar.gz
    │   │   └── nginx.conf
    │   ├── 1.16.1-centos7-base
    │   │   ├── app.tar.gz
    │   │   ├── build.sh
    │   │   ├── Dockerfile
```

```
|   |   ├── nginx-1.16.1.tar.gz
|   |   └── nginx.conf
|   └── 1.16.1-ubuntu1804
|       ├── build.sh
|       ├── Dockerfile
|       ├── index.html
|       ├── nginx-1.16.1.tar.gz
|       ├── nginx.conf
|       └── sources.list
└── tomcat
    ├── tomcat-app1
    |   ├── app
    |   |   └── index.jsp
    |   ├── app.tar.gz
    |   ├── build.sh
    |   ├── Dockerfile
    |   ├── run_tomcat.sh
    |   └── server.xml
    ├── tomcat-app2
    |   ├── app
    |   |   ├── index.html
    |   |   └── index.jsp
    |   ├── app.tar.gz
    |   ├── build.sh
    |   ├── Dockerfile
    |   ├── run_tomcat.sh
    |   └── server.xml
    └── tomcat-base-8.5.50
        ├── apache-tomcat-8.5.50.tar.gz
        ├── build.sh
        └── Dockerfile

21 directories, 51 files
[root@ubuntu1804 ~]#
```

6.5.4 查看harbor上的相关镜像

6.5.5 编辑docker compose文件及环境准备

6.5.5.1 编辑docker compose文件

```
[root@ubuntu1804 ~]#cat /data/docker-compose/docker-compose.yml
service-haproxy:
  image: 10.0.0.102/example/haproxy-centos7-base:2.1.2
  container_name: haproxy
  expose:
    - 80
    - 9999
  ports:
    - "80:80"
    - "9999:9999"
  links:
    - service-nginx-web

service-nginx-web:
  image: 10.0.0.102/example/nginx-centos7-base:1.6.1
  container_name: nginx-web
  volumes:
    - /data/nginx:/apps/nginx/html/
  links:
    - service-tomcat-app1
    - service-tomcat-app2

  expose:
   - 80
   - 443
#  ports:
#    - "80:80"
#    - "443:443"

service-tomcat-app1:
  image: 10.0.0.102/example/tomcat-web:app1
  container_name: tomcat-app1
  expose:
    - 8080
#  ports:
#    - "8081:8080"


service-tomcat-app2:
  image: 10.0.0.102/example/tomcat-web:app2
  container_name: tomcat-app2
  expose:
    - 8080
#  ports:
#    - "8082:8080"
[root@ubuntu1804 ~]#
```

6.5.5.2 启动容器

```
[root@ubuntu1804 ~]#cd /data/docker-compose/
[root@ubuntu1804 docker-compose]#docker-compose up -d
Pulling service-tomcat-app1 (10.0.0.102/example/tomcat-web:app1)...
app1: Pulling from example/tomcat-web
f34b00c7da20: Pull complete
544476d462f7: Pull complete
39345915aa1b: Pull complete
```

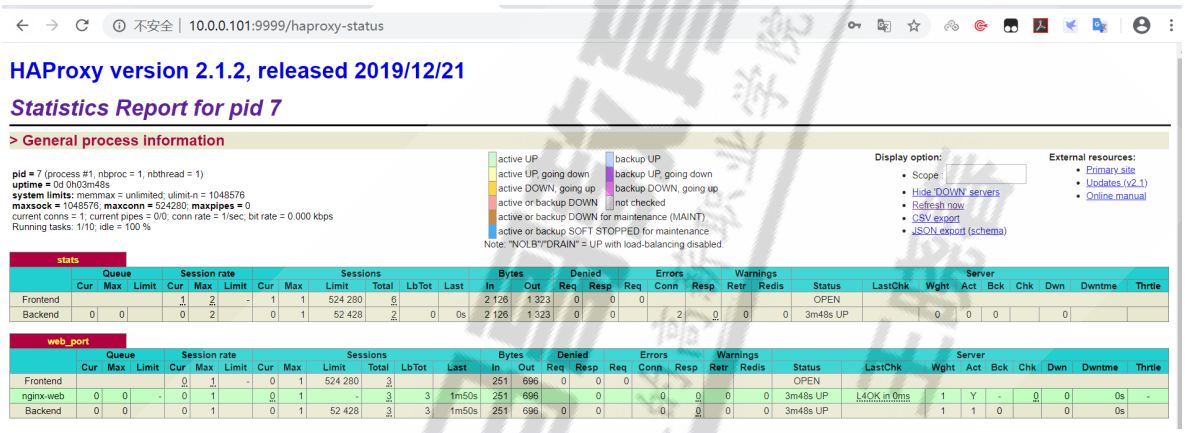
```
4b792f2bae38: Pull complete
4439447a3522: Pull complete
fe34d2ec1dd0: Pull complete
b8487ca03126: Pull complete
5a475b7d8b1a: Pull complete
df8703d3d2dd: Pull complete
f0da1ffa7aa7: Pull complete
80fd4c70e670: Pull complete
c2a0247d7bfa: Pull complete
b0977ed809cd: Pull complete
Digest: sha256:e0aba904df6095ea04c594d6906101f8e5f4a6ceb0a8f9b24432c47698d0caa8
Status: Downloaded newer image for 10.0.0.102/example/tomcat-web:app1
Pulling service-tomcat-app2 (10.0.0.102/example/tomcat-web:app2)...
app2: Pulling from example/tomcat-web
f34b00c7da20: Already exists
544476d462f7: Already exists
39345915aa1b: Already exists
4b792f2bae38: Already exists
4439447a3522: Already exists
fe34d2ec1dd0: Already exists
b8487ca03126: Already exists
5a475b7d8b1a: Already exists
df8703d3d2dd: Already exists
f0da1ffa7aa7: Already exists
80fd4c70e670: Already exists
1a55cb76a801: Pull complete
565ab795f82a: Pull complete
Digest: sha256:c4d6f166c3933f6c1ba59c84ea0518ed653af25f28b87981c242b0deff4209bb
Status: Downloaded newer image for 10.0.0.102/example/tomcat-web:app2
Pulling service-nginx-web (10.0.0.102/example/nginx-centos7-base:1.6.1)...
1.6.1: Pulling from example/nginx-centos7-base
f34b00c7da20: Already exists
544476d462f7: Already exists
39345915aa1b: Already exists
cf9a244574o2: Pull complete
15ef14db18ed: Pull complete
fc0bcf215997: Pull complete
2808e0476ed2: Pull complete
Digest: sha256:2eb7525c623ecf34bc2046888029cdb0ef01d866a0687ca7afd4dfd36c16a863
Status: Downloaded newer image for 10.0.0.102/example/nginx-centos7-base:1.6.1
Pulling service-haproxy (10.0.0.102/example/haproxy-centos7-base:2.1.2)...
2.1.2: Pulling from example/haproxy-centos7-base
f34b00c7da20: Already exists
544476d462f7: Already exists
39345915aa1b: Already exists
70a1e528112f: Pull complete
d240b62890cd: Pull complete
4223b7054cb3: Pull complete
7d2e1b713699: Pull complete
Digest: sha256:5df06247a5bd187894dd1c705f374568ecb4996e8d38bea71fd0bc679ab6ae13
Status: Downloaded newer image for 10.0.0.102/example/haproxy-centos7-base:2.1.2
Creating tomcat-app1 ... done
Creating tomcat-app2 ... done
Creating nginx-web   ... done
Creating haproxy     ... done
[root@ubuntu1804 docker-compose]#curl 127.0.0.1/app/
Tomcat Page in app1
[root@ubuntu1804 docker-compose]#curl 127.0.0.1/app/
```

```
Tomcat Page in app2
[root@ubuntu1804 docker-compose]#curl 127.0.0.1
Docker compose test page
```

6.5.5.3 验证容器启动成功

```
[root@ubuntu1804 ~]#curl http://10.0.0.101/
Docker compose test page
[root@ubuntu1804 ~]#curl http://10.0.0.101/app/
Tomcat Page in app1
[root@ubuntu1804 ~]#curl http://10.0.0.101/app/
Tomcat Page in app2
```

访问haroxy 管理界面:



7 docker 的资源限制

7.1 docker 资源限制

7.1.1 容器资源限制介绍

官方文档: https://docs.docker.com/config/containers/resource_constraints/

默认情况下，容器没有资源的使用限制，可以使用主机内核调度程序允许的尽可能多的资源

Docker 提供了控制容器使用资源的方法,可以限制容器使用多少内存或 CPU等， 在docker run 命令的运行时配置标志实现资源限制功能。

其中许多功能都要求宿主机的内核支持，要检查是否支持这些功能，可以使用docker info 命令 ，如果内核中的某项特性可能会在输出结尾处看到警告， 如下所示:

```
WARNING: No swap limit support
```

可通过修改内核参数消除以上警告

官方文档: https://docs.docker.com/install/linux/linux-postinstall/#your-kernel-does-not-support-cgroup-swap-limit-capabilities

范例: 修改内核参数消除以上警告

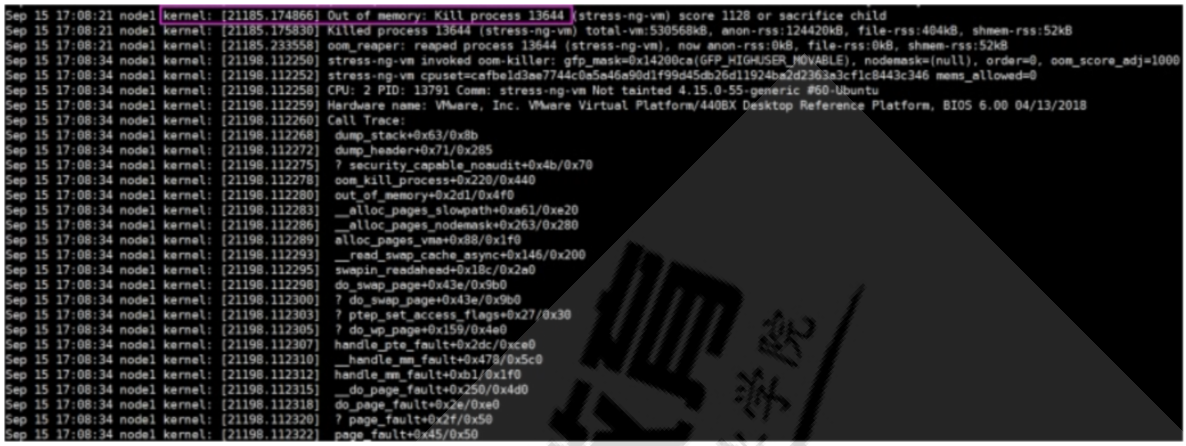
```
[root@ubuntu1804 ~]#docker info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 1
 Server Version: 19.03.5
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: b34a5c8af56e510852c35414db4c1f4fa6172339
 runc version: 3e425f80a8c931f88e6d94a8c831b9d5aa481657
 init version: fec3683
 Security Options:
  apparmor
  seccomp
   Profile: default
 Kernel Version: 4.15.0-29-generic
 Operating System: Ubuntu 18.04.1 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 962MiB
 Name: ubuntu1804.magedu.org
 ID: 4LUN:CS4Y:SGYQ:ZAXU:M7C5:JYSB:ZFTO:JZ6D:CD7Q:TOVD:JBFS:AQM3
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  10.0.0.102
  127.0.0.0/8
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/
 Live Restore Enabled: false

WARNING: No swap limit support

#修改内核参数
[root@ubuntu1804 ~]#vim /etc/default/grub
GRUB_CMDLINE_LINUX="cgroup_enable=memory net.ifnames=0 swapaccount=1"
```

```
[root@ubuntu1804 ~]#update-grub
[root@ubuntu1804 ~]#reboot
[root@ubuntu1804 ~]#docker info
Client:
 Debug Mode: false

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 1
 Server Version: 19.03.5
 Storage Driver: overlay2
  Backing Filesystem: extfs
  Supports d_type: true
  Native Overlay Diff: true
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk
syslog
 Swarm: inactive
 Runtimes: runc
 Default Runtime: runc
 Init Binary: docker-init
 containerd version: b34a5c8af56e510852c35414db4c1f4fa6172339
 runc version: 3e425f80a8c931f88e6d94a8c831b9d5aa481657
 init version: fec3683
 Security Options:
  apparmor
  seccomp
   Profile: default
 Kernel Version: 4.15.0-29-generic
 Operating System: Ubuntu 18.04.1 LTS
 OSType: linux
 Architecture: x86_64
 CPUs: 1
 Total Memory: 962MiB
 Name: ubuntu1804.magedu.org
 ID: 4LUN:CS4Y:SGYQ:ZAXU:M7C5:JYSB:ZFTO:JZ6D:CD7Q:TOVD:JBFS:AQM3
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  10.0.0.102
  127.0.0.0/8
 Registry Mirrors:
  https://si7y70hh.mirror.aliyuncs.com/
 Live Restore Enabled: false

[root@ubuntu1804 ~]#
```

7.1.2 OOM （Out of Memory Exception）

对于 Linux 主机，如果没有足够的内存来执行其他重要的系统任务，将会抛出OOM (Out of Memory Exception,内存溢出、内存泄漏、内存异常)，随后系统会开始杀死进程以释放内存， 凡是运行在宿主机的进程都有可能被 kill ，包括 Dockerd和其它的应用程序， 如果重要的系统进程被 Kill，会导致和该进程相关的服务全部宕机。通常越消耗内存比较大的应用越容易被kill，比如：MySQL数据库，Java程序等

范例：OOM发生后的日志信息



产生 OOM 异常时， Dockerd尝试通过调整 Docker 守护程序上的 OOM 优先级来减轻这些风险，以便它比系统上的其他进程更不可能被杀死但是容器 的 OOM 优先级未调整， 这使得单个容器被杀死的可能性比 Docker守护程序或其他系统进程被杀死的可能性更大，不推荐通过在守护程序或容器上手动设置-- oom -score-adj为极端负数，或通过在容器上设置 -- oom-kill-disable来绕过这些安全措施

OOM 优先级机制:

linux会为每个进程算一个分数，最终将分数最高的kill

```
/proc/PID/oom_score_adj
#范围为 -1000 到 1000，值越高容易被宿主机 kill掉，如果将该值设置为 -1000 ，则进程永远不会被宿主机 kernel kill

/proc/PID/oom_adj
#范围为 -17 到+15 ，取值越高越容易被干掉，如果是 -17 ， 则表示不能被 kill ，该设置参数的存在是为了和旧版本的 Linux 内核兼容。

/proc/PID/oom_score
#这个值是系统综合进程的内存消耗量、 CPU 时间 (utime + 、存活时间 (uptime - start time) 和 oom_adj 计算出的进程得分 ，消耗内存越多得分越高，容易被宿主机 kernel 强制杀死
```

范例：查看OOM相关值

```
#按内存排序
[root@ubuntu1804 ~]#top
top - 20:15:38 up  5:53,  3 users,  load average: 0.00, 0.00, 0.00
Tasks: 191 total,   1 running, 116 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.3 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :   985104 total,   310592 free,   448296 used,   226216 buff/cache
KiB Swap:  1951740 total,  1892860 free,    58880 used.   384680 avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
```

```
 19674 2019      20   0 2241656  94684  12452 S  0.0  9.6   0:16.05 java

 19675 2019      20   0 2235512  74816  12440 S  0.0  7.6   0:14.89 java

 19860 99        20   0  183212  67748    960 S  0.0  6.9   0:01.15 haproxy

  4969 root      20   0  937880  49352  12612 S  0.0  5.0   0:46.07 dockerd

  2981 root      20   0  793072  13552   1808 S  0.0  1.4   0:13.78 containerd

   500 root      19  -1   78560   7552   7112 S  0.0  0.8   0:01.45 systemd-
journal
   798 root      20   0  170416   6604   4084 S  0.0  0.7   0:00.77 networkd-
dispat
     1 root      20   0   78036   6200   4416 S  0.0  0.6   0:05.39 systemd

  1011 root      20   0   24548   5496   3012 S  0.0  0.6   0:01.62 bash

   852 root      10 -10   25880   5264   4036 S  0.0  0.5   0:00.00 iscsid

   815 root      20   0  548292   4624   1224 S  0.0  0.5   0:01.89 snapd

 19586 root      20   0  109104   4532   3768 S  0.0  0.5   0:00.29 containerd-
shim
 19779 root      20   0  405532   4224   2828 S  0.0  0.4   0:00.01 docker-proxy

 19784 root      20   0  107696   4204   3652 S  0.0  0.4   0:00.29 containerd-
shim
 19424 root      20   0  109104   4084   3416 S  0.0  0.4   0:00.27 containerd-
shim
 20064 root      20   0   44076   4036   3360 R  0.7  0.4   0:00.20 top

 19768 root      20   0  405532   4024   2644 S  0.0  0.4   0:00.01 docker-proxy

 19423 root      20   0  109104   3792   3064 S  0.0  0.4   0:00.31 containerd-
shim
   490 root      20   0  193112   3316   2864 S  0.0  0.3   0:26.20 vmtoolsd

  7108 root      20   0  105688   3204   2504 S  0.0  0.3   0:00.11 sshd


[root@ubuntu1804 ~]#cat /proc/19674/oom_adj
0
[root@ubuntu1804 ~]#cat /proc/19674/oom_score
32
[root@ubuntu1804 ~]#cat /proc/19674/oom_score_adj
0
[root@ubuntu1804 ~]#cat /proc/7108/oom_adj
0
[root@ubuntu1804 ~]#cat /proc/7108/oom_score
1
[root@ubuntu1804 ~]#cat /proc/7108/oom_score_adj
0

#docker服务进程的OOM默认值
[root@ubuntu1804 ~]#cat /proc/`pidof dockerd`/oom_adj
-8
[root@ubuntu1804 ~]#cat /proc/`pidof dockerd`/oom_score
```

```
0
[root@ubuntu1804 ~]#cat /proc/`pidof dockerd`/oom_score_adj
-500
```

7.2 容器的内存限制

Docker 可以强制执行硬性内存限制，即只允许容器使用给定的内存大小。

Docker 也可以执行非硬性内存限制，即容器可以使用尽可能多的内存，除非内核检测到主机上的内存不够用了

7.2.1 内存相关选项

官文文档: https://docs.docker.com/config/containers/resource_constraints/

以下设置大部分的选项取正整数，跟着一个后缀 `b`，`k`，`m`，`g`，，表示字节，千字节，兆字节或千兆字节

选项	描述
`-m`，`--memory=`	容器可以使用的最大物理内存量，硬限制，此选项最小允许值为 `4m`（4 MB），此项较常用
--memory-swap	允许此容器交换到磁盘的内存量,必须先用-m 对内存限制才可以使用,详细说明如下
`--memory-swappiness`	设置容器使用交换分区的倾向性，值越高表示越倾向于使用swap分区，范围为0-100，0为能不用就不用，100为能用就用
`--memory-reservation`	允许指定小于 --memory 的软限制，当 Docker 检测到主机上的争用或内存不足时会激活该限制，如果使-- memory-reservation，则必须将其设置为低于 --memory 才能使其优先生效。 因为它是软限制，所以不能保证容器不超过限制
`--kernel-memory`	容器可以使用的最大内核内存量，最小为 4m，由于内核内存与用户空间内存隔离，因此无法与用户空间内存直接交换，因此内核内存不足的容器可能会阻塞宿主机资源，这会对主机和其他容器或者其他服务进程产生影响，因此不建议设置内核内存大小
`--oom-kill-disable`	默认情况下，如果发生内存不足（OOM）错误，则内核将终止容器中的进程。要更改此行为，请使用该 `--oom-kill-disable` 选项。仅在设置了该 `-m/--memory` 选项的容器上禁用OOM。如果 `-m` 未设置该标志，则主机可能会用完内存，内核可能需要终止主机系统的进程以释放内存

范例:

```
[root@ubuntu1804 ~]#docker run -e MYSQL_ROOT_PASSWORD=123456 -it --rm -m 1g --
oom-kill-disable mysql:5.7.29
2020-02-04 13:11:54+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL
Server 5.7.29-1debian9 started.
2020-02-04 13:11:54+00:00 [Note] [Entrypoint]: Switching to dedicated user
'mysql'
2020-02-04 13:11:54+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL
Server 5.7.29-1debian9 started.
2020-02-04 13:11:54+00:00 [Note] [Entrypoint]: Initializing database files
......
Version: '5.7.29'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL
Community Server (GPL)
```

范例:

```
[root@ubuntu1804 ~]#sysctl  -a |grep swappiness
sysctl: reading key "net.ipv6.conf.all.stable_secret"
sysctl: reading key "net.ipv6.conf.default.stable_secret"
sysctl: reading key "net.ipv6.conf.docker0.stable_secret"
sysctl: reading key "net.ipv6.conf.eth0.stable_secret"
sysctl: reading key "net.ipv6.conf.lo.stable_secret"
vm.swappiness = 60
```

7.2.2 swap限制

```
--memory-swap #只有在设置了 --memory 后才会有意义。使用 Swap,可以让容器将超出限制部分的内
存置换到磁盘上，WARNING:  经常将内存交换到磁盘的应用程序会降低性能
```

不同的--memory-swap 设置会产生不同的效果:

--memory-swap	--memory	功能
正数S	正数M	容器可用内存总空间为S,其中ram为M,swap为 S-M,若S=M,则无可用swap资源
0	正数M	相当于未设置swap(unset)
unset	正数M	若主机(Docker Host) 启用于swap , 则容器的可用swap 为2*M
-1	正数M	若主机(Docker Host)启用了swap ,则容器可使用最大至主机上所有swap空间

```
-memory-swap #值为正数， 那么--memory 和--memory-swap 都必须要设置，--memory-swap 表示
你能使用的内存和 swap 分区大小的总和，例如:  --memory=300m, --memory-swap=1g, 那么该容
器能够使用 300m 物理内存和 700m swap，即--memory 是实际物理内存大小值不变，而 swap 的实际
大小计算方式为(--memory-swap)-(--memory)=容器可用 swap
--memory-swap #如果设置为 0，则忽略该设置，并将该值视为未设置，即未设置交换分区
--memory-swap #如果等于--memory 的值，并且--memory 设置为正整数，容器无权访问 swap
-memory-swap #如果未设置，如果宿主机开启了 swap，则实际容器的swap 值最大为 2x( --
memory)，即两倍于物理内存大小，例如，如果--memory="300m"与--memory-swap没有设置，该容器可
以使用300m总的内存和600m交撒空间,但是并不准确(在容器中使用free 命令所看到的 swap 空间并不精
确，毕竟每个容器都可以看到具体大小，宿主机的 swap 是有上限的，而且不是所有容器看到的累计大小)
--memory-swap #如果设置为-1，如果宿主机开启了 swap，则容器可以使用主机上 swap 的最大空间
```

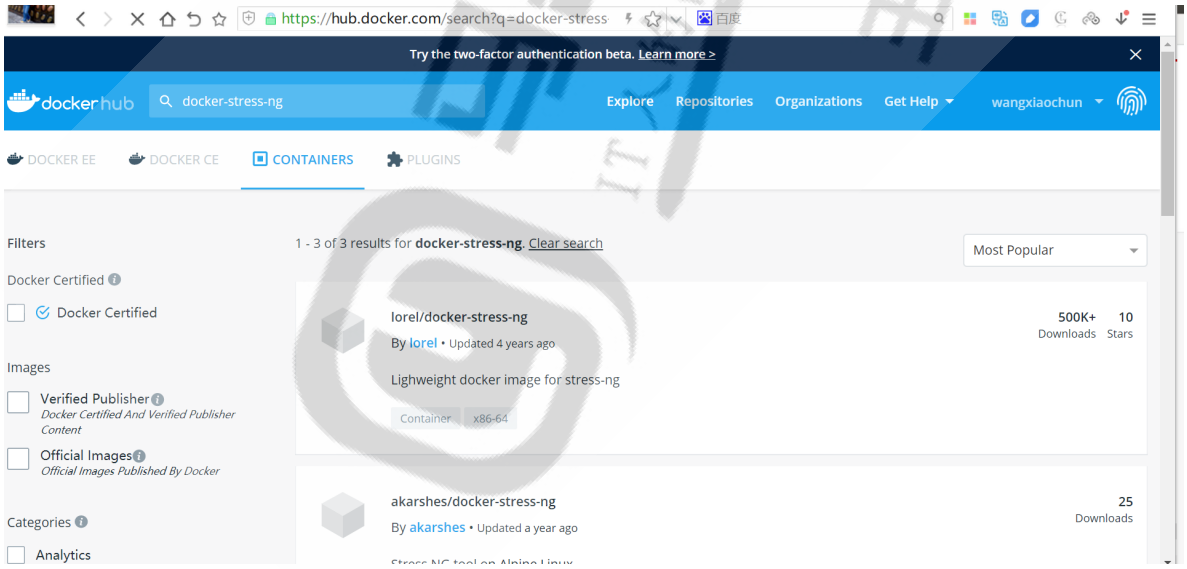
注意：在容器中执行free命令看到的是宿主机的内存和swap使用，而非容器自身的swap使用情况

范例：在容器中查看内存

```
[root@ubuntu1804 ~]#free
              total        used        free      shared  buff/cache   available
Mem:        3049484      278484     1352932       10384     1418068     2598932
Swap:       1951740           0     1951740
[root@ubuntu1804 ~]#docker run -it --rm -m 2G centos:centos7.7.1908 bash
[root@f5d387b5022f /]# free
              total        used        free      shared  buff/cache   available
Mem:        3049484      310312     1320884       10544     1418288     2566872
Swap:       1951740           0     1951740
[root@f5d387b5022f /]#
```

7.2.3 stress-ng 压力测试工具

stress-ng是一个压力测试工具，可以通过软件仓库进行安装，也提供了docker版本的容器



范例：软件包方式安装

```
[root@centos7 ~]#yum -y install stress-ng
[root@ubuntu1804 ~]#apt -y install stress-ng
```

范例：容器方式安装

```
[root@ubuntu1804 ~]#docker pull lorel/docker-stress-ng
Using default tag: latest
latest: Pulling from lorel/docker-stress-ng
Image docker.io/lorel/docker-stress-ng:latest uses outdated schema1 manifest
format. Please upgrade to a schema2 image for better future compatibility. More
information at https://docs.docker.com/registry/spec/deprecated-schema-v1/
c52e3ed763ff: Pull complete
a3ed95caeb02: Pull complete
7f831269c70e: Pull complete
Digest: sha256:c8776b750869e274b340f8e8eb9a7d8fb2472edd5b25ff5b7d55728bca681322
Status: Downloaded newer image for lorel/docker-stress-ng:latest
docker.io/lorel/docker-stress-ng:latest
[root@ubuntu1804 ~]#docker images
REPOSITORY               TAG                 IMAGE ID            CREATED
    SIZE
lorel/docker-stress-ng   latest              1ae56ccafe55        3 years ago
    8.1MB
```

范例: 查看帮助

```
[root@ubuntu1804 ~]#docker run -it --rm lorel/docker-stress-ng
stress-ng, version 0.03.11

Usage: stress-ng [OPTION [ARG]]
 --h,  --help              show help
       --affinity N        start N workers that rapidly change CPU affinity
       --affinity-ops N    stop when N affinity bogo operations completed
       --affinity-rand     change affinity randomly rather than sequentially
       --aio N             start N workers that issue async I/O requests
       --aio-ops N         stop when N bogo async I/O requests completed
       --aio-requests N    number of async I/O requests per worker
 -a N, --all N             start N workers of each stress test
 -b N, --backoff N         wait of N microseconds before work starts
 -B N, --bigheap N         start N workers that grow the heap using calloc()
       --bigheap-ops N     stop when N bogo bigheap operations completed
       --bigheap-growth N  grow heap by N bytes per iteration
       --brk N             start N workers performing rapid brk calls
       --brk-ops N         stop when N brk bogo operations completed
       --brk-notouch       don't touch (page in) new data segment page
       --bsearch           start N workers that exercise a binary search
       --bsearch-ops       stop when N binary search bogo operations completed
       --bsearch-size      number of 32 bit integers to bsearch
 -C N, --cache N           start N CPU cache thrashing workers
       --cache-ops N       stop when N cache bogo operations completed (x86 only)
       --cache-flush       flush cache after every memory write (x86 only)
       --cache-fence       serialize stores
       --class name        specify a class of stressors, use with --sequential
       --chmod N           start N workers thrashing chmod file mode bits
       --chmod-ops N       stop chmod workers after N bogo operations
 -c N, --cpu N             start N workers spinning on sqrt(rand())
       --cpu-ops N         stop when N cpu bogo operations completed
 -l P, --cpu-load P        load CPU by P %%, 0=sleep, 100=full load (see -c)
       --cpu-method m      specify stress cpu method m, default is all
 -D N, --dentry N          start N dentry thrashing processes
       --dentry-ops N      stop when N dentry bogo operations completed
       --dentry-order O    specify dentry unlink order (reverse, forward, stride)
```

```
      --dentries N        create N dentries per iteration
      --dir N             start N directory thrashing processes
      --dir-ops N         stop when N directory bogo operations completed
-n,   --dry-run           do not run
      --dup N             start N workers exercising dup/close
      --dup-ops N         stop when N dup/close bogo operations completed
      --epoll N           start N workers doing epoll handled socket activity
      --epoll-ops N       stop when N epoll bogo operations completed
      --epoll-port P      use socket ports P upwards
      --epoll-domain D    specify socket domain, default is unix
      --eventfd N         start N workers stressing eventfd read/writes
      --eventfd-ops N     stop eventfd workers after N bogo operations
      --fault N           start N workers producing page faults
      --fault-ops N       stop when N page fault bogo operations completed
      --fifo N            start N workers exercising fifo I/O
      --fifo-ops N        stop when N fifo bogo operations completed
      --fifo-readers N    number of fifo reader processes to start
      --flock N           start N workers locking a single file
      --flock-ops N       stop when N flock bogo operations completed
-f N, --fork N            start N workers spinning on fork() and exit()
      --fork-ops N        stop when N fork bogo operations completed
      --fork-max P        create P processes per iteration, default is 1
      --fstat N           start N workers exercising fstat on files
      --fstat-ops N       stop when N fstat bogo operations completed
      --fstat-dir path    fstat files in the specified directory
      --futex N           start N workers exercising a fast mutex
      --futex-ops N       stop when N fast mutex bogo operations completed
      --get N             start N workers exercising the get*() system calls
      --get-ops N         stop when N get bogo operations completed
-d N, --hdd N             start N workers spinning on write()/unlink()
      --hdd-ops N         stop when N hdd bogo operations completed
      --hdd-bytes N       write N bytes per hdd worker (default is 1GB)
      --hdd-direct        minimize cache effects of the I/O
      --hdd-dsync         equivalent to a write followed by fdatasync
      --hdd-noatime       do not update the file last access time
      --hdd-sync          equivalent to a write followed by fsync
      --hdd-write-size N  set the default write size to N bytes
      --hsearch           start N workers that exercise a hash table search
      --hsearch-ops       stop when N hash search bogo operations completed
      --hsearch-size      number of integers to insert into hash table
      --inotify N         start N workers exercising inotify events
      --inotify-ops N     stop inotify workers after N bogo operations
-i N, --io N              start N workers spinning on sync()
      --io-ops N          stop when N io bogo operations completed
      --ionice-class C    specify ionice class (idle, besteffort, realtime)
      --ionice-level L    specify ionice level (0 max, 7 min)
-k,   --keep-name         keep stress process names to be 'stress-ng'
      --kill N            start N workers killing with SIGUSR1
      --kill-ops N        stop when N kill bogo operations completed
      --lease N           start N workers holding and breaking a lease
      --lease-ops N       stop when N lease bogo operations completed
      --lease-breakers N  number of lease breaking processes to start
      --link N            start N workers creating hard links
      --link-ops N        stop when N link bogo operations completed
      --lsearch           start N workers that exercise a linear search
      --lsearch-ops       stop when N linear search bogo operations completed
      --lsearch-size      number of 32 bit integers to lsearch
-M,   --metrics           print pseudo metrics of activity
```

```
       --metrics-brief     enable metrics and only show non-zero results
       --memcpy N          start N workers performing memory copies
       --memcpy-ops N      stop when N memcpy bogo operations completed
       --mmap N            start N workers stressing mmap and munmap
       --mmap-ops N        stop when N mmap bogo operations completed
       --mmap-async        using asynchronous msyncs for file based mmap
       --mmap-bytes N      mmap and munmap N bytes for each stress iteration
       --mmap-file         mmap onto a file using synchronous msyncs
       --mmap-mprotect     enable mmap mprotect stressing
       --msg N             start N workers passing messages using System V
messages
       --msg-ops N         stop msg workers after N bogo messages completed
       --mq N              start N workers passing messages using POSIX messages
       --mq-ops N          stop mq workers after N bogo messages completed
       --mq-size N         specify the size of the POSIX message queue
       --nice N            start N workers that randomly re-adjust nice levels
       --nice-ops N        stop when N nice bogo operations completed
       --no-madvise        don't use random madvise options for each mmap
       --null N            start N workers writing to /dev/null
       --null-ops N        stop when N /dev/null bogo write operations completed
 -o,   --open N            start N workers exercising open/close
       --open-ops N        stop when N open/close bogo operations completed
 -p N, --pipe N            start N workers exercising pipe I/O
       --pipe-ops N        stop when N pipe I/O bogo operations completed
 -P N, --poll N            start N workers exercising zero timeout polling
       --poll-ops N        stop when N poll bogo operations completed
       --procfs N          start N workers reading portions of /proc
       --procfs-ops N      stop procfs workers after N bogo read operations
       --pthread N         start N workers that create multiple threads
       --pthread-ops N     stop pthread workers after N bogo threads created
       --pthread-max P     create P threads at a time by each worker
 -Q,   --qsort N           start N workers exercising qsort on 32 bit random
integers
       --qsort-ops N       stop when N qsort bogo operations completed
       --qsort-size N      number of 32 bit integers to sort
 -q,   --quiet             quiet output
 -r,   --random N          start N random workers
       --rdrand N          start N workers exercising rdrand instruction (x86
only)
       --rdrand-ops N      stop when N rdrand bogo operations completed
 -R,   --rename N          start N workers exercising file renames
       --rename-ops N      stop when N rename bogo operations completed
       --sched type        set scheduler type
       --sched-prio N      set scheduler priority level N
       --seek N            start N workers performing random seek r/w IO
       --seek-ops N        stop when N seek bogo operations completed
       --seek-size N       length of file to do random I/O upon
       --sem N             start N workers doing semaphore operations
       --sem-ops N         stop when N semaphore bogo operations completed
       --sem-procs N       number of processes to start per worker
       --sendfile N        start N workers exercising sendfile
       --sendfile-ops N    stop after N bogo sendfile operations
       --sendfile-size N   size of data to be sent with sendfile
       --sequential N      run all stressors one by one, invoking N of them
       --sigfd N           start N workers reading signals via signalfd reads
       --sigfd-ops N       stop when N bogo signalfd reads completed
       --sigfpe N          start N workers generating floating point math faults
       --sigfpe-ops N      stop when N bogo floating point math faults completed
```

```
      --sigsegv N         start N workers generating segmentation faults
      --sigsegv-ops N     stop when N bogo segmentation faults completed
 -S N, --sock N           start N workers doing socket activity
      --sock-ops N        stop when N socket bogo operations completed
      --sock-port P       use socket ports P to P + number of workers - 1
      --sock-domain D     specify socket domain, default is ipv4
      --stack N           start N workers generating stack overflows
      --stack-ops N       stop when N bogo stack overflows completed
 -s N, --switch N         start N workers doing rapid context switches
      --switch-ops N      stop when N context switch bogo operations completed
      --symlink N         start N workers creating symbolic links
      --symlink-ops N     stop when N symbolic link bogo operations completed
      --sysinfo N         start N workers reading system information
      --sysinfo-ops N     stop when sysinfo bogo operations completed
 -t N, --timeout N        timeout after N seconds
 -T N, --timer N          start N workers producing timer events
      --timer-ops N       stop when N timer bogo events completed
      --timer-freq F      run timer(s) at F Hz, range 1000 to 1000000000
      --tsearch           start N workers that exercise a tree search
      --tsearch-ops       stop when N tree search bogo operations completed
      --tsearch-size      number of 32 bit integers to tsearch
      --times             show run time summary at end of the run
 -u N, --urandom N        start N workers reading /dev/urandom
      --urandom-ops N     stop when N urandom bogo read operations completed
      --utime N           start N workers updating file timestamps
      --utime-ops N       stop after N utime bogo operations completed
      --utime-fsync       force utime meta data sync to the file system
 -v,   --verbose          verbose output
      --verify            verify results (not available on all tests)
 -V,   --version          show version
 -m N, --vm N             start N workers spinning on anonymous mmap
      --vm-bytes N        allocate N bytes per vm worker (default 256MB)
      --vm-hang N         sleep N seconds before freeing memory
      --vm-keep           redirty memory instead of reallocating
      --vm-ops N          stop when N vm bogo operations completed
      --vm-locked         lock the pages of the mapped region into memory
      --vm-method m       specify stress vm method m, default is all
      --vm-populate       populate (prefault) page tables for a mapping
      --wait N            start N workers waiting on child being stop/resumed
      --wait-ops N        stop when N bogo wait operations completed
      --zero N            start N workers reading /dev/zero
      --zero-ops N        stop when N /dev/zero bogo read operations completed

Example: stress-ng --cpu 8 --io 4 --vm 2 --vm-bytes 128M --fork 4 --timeout 10s

Note: Sizes can be suffixed with B,K,M,G and times with s,m,h,d,y
```

假如一个容器未做内存使用限制，则该容器可以利用到系统内存最大空间，默认创建的容器没有做内存资源限制。

范例：默认一个workers 分配256M内存，2个即占512M内存

```
[root@ubuntu1804 ~]#docker run --name c1 -it --rm lorel/docker-stress-ng --vm 2
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

#因上一个命令是前台执行，下面在另一个终端窗口中执行，可以看到占用512M左右内存
[root@ubuntu1804 ~]#docker stats
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
fd184869ff7e        c1                  91.00%              524.3MiB / 962MiB
54.50%              766B / 0B           860kB / 0B          5
```

范例: 指定内存最大值

```
[root@ubuntu1804 ~]#docker run --name c1 -it --rm -m 300m lorel/docker-stress-ng
--vm 2
WARNING: Your kernel does not support swap limit capabilities or the cgroup is
not mounted. Memory limited without swap.
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

[root@ubuntu1804 ~]#vim /etc/default/grub
GRUB_CMDLINE_LINUX="cgroup_enable=memory swapaccount=1  net.ifnames=0"
[root@ubuntu1804 ~]#update-grub
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-4.15.0-29-generic
Found initrd image: /boot/initrd.img-4.15.0-29-generic
done
[root@ubuntu1804 ~]#reboot
[root@ubuntu1804 ~]#docker run --name c1 -it --rm -m 300m lorel/docker-stress-ng
--vm 2
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

#在另一个终端窗口执行
[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
6a93f6b22034        c1                  27.06%              297.2MiB / 300MiB
99.07%              1.45kB / 0B         4.98GB / 5.44GB     5
```

范例:

```
[root@ubuntu1804 ~]#docker run --name c2 -it --rm  lorel/docker-stress-ng --vm 4
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 4 vm

#一次性查看资源使用情况
[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
fd5fff3c04f7        c2                  21.20%              591.1MiB / 962MiB
61.45%              1.31kB / 0B         1.07GB / 46.6MB     9
[root@ubuntu1804 ~]#
```

范例: 容器占用内存造成OOM

```
[root@ubuntu1804 ~]#docker run -it --rm  lorel/docker-stress-ng --vm 6
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 6 vm

#另一个终端窗中同时执行下面命令
[root@ubuntu1804 ~]#docker run  -it --rm  lorel/docker-stress-ng --vm 6
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 6 vm


[root@ubuntu1804 ~]#docker stats
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
f33cebf5b55d        c2                  --                  -- / --
--                  --                  --                  --
b14b597c5a4f        cool_banach         --                  -- / --
--                  --                  --                  --

#观察日志出现OOM现象
[root@ubuntu1804 ~]#tail  /var/log/syslog
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928835] [ 2575]     0  2575    67104
 39218   544768    22906          1000 stress-ng-vm
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928836] [ 2594]     0  2594    67104
 37503   409600     7725          1000 stress-ng-vm
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928837] [ 2601]     0  2601    67104
 38815   438272     9779          1000 stress-ng-vm
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928838] [ 2602]     0  2602     1568
   861    49152        0          1000 stress-ng-vm
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928839] [ 2610]     0  2610     1568
   861    49152        0          1000 stress-ng-vm
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928840] [ 2614]     0  2614     1157
   174    53248        0             0 update-motd-hwe
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928841] [ 2615]     0  2615     3100
    15    61440        0             0 apt-config
Feb  4 22:59:40 ubuntu1804 kernel: [  785.928842] Out of memory: Kill process
2570 (stress-ng-vm) score 1090 or sacrifice child
Feb  4 22:59:40 ubuntu1804 kernel: [  785.929493] Killed process 2570 (stress-
ng-vm) total-vm:268416kB, anon-rss:170352kB, file-rss:632kB, shmem-rss:28kB
Feb  4 22:59:40 ubuntu1804 kernel: [  786.018319] oom_reaper: reaped process
2570 (stress-ng-vm), now anon-rss:0kB, file-rss:0kB, shmem-rss:28kB
```

范例: 查看内存限制

```
#启动两个工作进程，每个工作进程最大允许使用内存 256M，且宿主机不限制当前容器最大内存
[root@ubuntu1804 ~]#docker run  -it --rm  lorel/docker-stress-ng --vm 2
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

[root@ubuntu1804 ~]#docker ps -a
CONTAINER ID        IMAGE                      COMMAND                  CREATED
          STATUS              PORTS               NAMES
13e46172e1ae        lorel/docker-stress-ng   "/usr/bin/stress-ng …"   24 seconds
ago       Up 22 seconds                           gallant_moore

[root@ubuntu1804 ~]#ls /sys/fs/cgroup/memory/docker/
```

```
13e46172e1ae8593569f05a3bebc7b41b7839da44369d43b29102661364ac2cd
memory.kmem.tcp.limit_in_bytes      memory.numa_stat
cgroup.clone_children
memory.kmem.tcp.max_usage_in_bytes  memory.oom_control
cgroup.event_control
memory.kmem.tcp.usage_in_bytes      memory.pressure_level
cgroup.procs
memory.kmem.usage_in_bytes          memory.soft_limit_in_bytes
memory.failcnt
memory.limit_in_bytes               memory.stat
memory.force_empty
memory.max_usage_in_bytes           memory.swappiness
memory.kmem.failcnt
memory.memsw.failcnt                memory.usage_in_bytes
memory.kmem.limit_in_bytes
memory.memsw.limit_in_bytes         memory.use_hierarchy
memory.kmem.max_usage_in_bytes
memory.memsw.max_usage_in_bytes     notify_on_release
memory.kmem.slabinfo
memory.memsw.usage_in_bytes         tasks
memory.kmem.tcp.failcnt
memory.move_charge_at_immigrate

[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/13e46172e1ae8593569f05a3bebc7b41b7839da44369d43b291
02661364ac2cd/memory.limit_in_bytes
9223372036854771712
[root@ubuntu1804 ~]#echo 2^63|bc
9223372036854775808
```

范例: 内存限制200m

```
#宿主机限制容器最大内存使用:
[root@ubuntu1804 ~]#docker run  -it --rm  -m 200M lorel/docker-stress-ng --vm 2
--vm-bytes 256M
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
f69729b2acc1        sleepy_haibt        85.71%              198MiB / 200MiB
98.98%              1.05kB / 0B         697MB / 60.4GB      5
[root@ubuntu1804 ~]#

#查看宿主机基于 cgroup 对容器进行内存资源的大小限制
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/f69729b2acc16e032658a4efdab64d21ff97dcb6746d1cef451
ed82d5c98a81f/memory.limit_in_bytes
209715200

[root@ubuntu1804 ~]#echo 209715200/1024/1024|bc
200

#动态修改内存限制
[root@ubuntu1804 ~]#echo 300*1024*1024|bc
```

```
314572800
[root@ubuntu1804 ~]#echo 314572800 >
/sys/fs/cgroup/memory/docker/f69729b2acc16e032658a4efdab64d21ff97dcb6746d1cef451
ed82d5c98a81f/memory.limit_in_bytes
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/f69729b2acc16e032658a4efdab64d21ff97dcb6746d1cef451
ed82d5c98a81f/memory.limit_in_bytes
314572800
[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
f69729b2acc1        sleepy_haibt        76.69%              297.9MiB / 300MiB
99.31%              1.05kB / 0B         1.11GB / 89.1GB     5

#通过echo 命令可以改内存限制的值，但是可以在原基础之上增大内存限制，缩小内存限制会报错write
error: Device or resource busy
[root@ubuntu1804 ~]#echo 209715200 >
/sys/fs/cgroup/memory/docker/f69729b2acc16e032658a4efdab64d21ff97dcb6746d1cef451
ed82d5c98a81f/memory.limit_in_bytes
-bash: echo: write error: Device or resource busy
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/f69729b2acc16e032658a4efdab64d21ff97dcb6746d1cef451
ed82d5c98a81f/memory.limit_in_bytes
314572800
```

范例: 内存大小软限制

```
[root@ubuntu1804 ~]#docker run -it --rm  -m 256m --memory-reservation 128m --
name magedu-c1 lorel/docker-stress-ng --vm 2 --vm-bytes 256M
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
aeb38acde581        magedu-c1           72.45%              253.9MiB / 256MiB
99.20%              976B / 0B           9.47GB / 39.4GB     5

#查看硬限制
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/aeb38acde58155d421f998a54e9a99ab60635fe00c9070da050
cc49a2f62d274/memory.limit_in_bytes
268435456

#查看软限制
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/aeb38acde58155d421f998a54e9a99ab60635fe00c9070da050
cc49a2f62d274/memory.soft_limit_in_bytes
134217728

#软限制不能高于硬限制
[root@ubuntu1804 ~]#docker run -it --rm  -m 256m --memory-reservation 257m --
name magedu-c1 lorel/docker-stress-ng --vm 2 --vm-bytes 256M
docker: Error response from daemon: Minimum memory limit can not be less than
memory reservation limit, see usage.
See 'docker run --help'.
```

关闭OOM 机制:

```
# docker run -it --rm  -m 256m --oom-kill-disable --name magedu-c1
lorel/docker-stress-ng --vm 2 --vm-bytes 256M
# cat /sys/fs/cgroup/memory/docker/容器 ID/memory.oom_control
oom_kill_disable 1
under_oom 1
oom_kill 0
```

范例: 关闭OOM机制

```
#查看docker OOM机制默认值
[root@ubuntu1804 ~]#cat /sys/fs/cgroup/memory/docker/memory.oom_control
oom_kill_disable 0
under_oom 0
oom_kill 0

#启动容器时关闭OOM机制
[root@ubuntu1804 ~]#docker run -it --rm  -m 200m --oom-kill-disable
lorel/docker-stress-ng --vm 2 --vm-bytes 256M
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
MEM %               NET I/O             BLOCK I/O           PIDS
b655d88228c0        silly_borg          0.00%               197.2MiB / 200MiB
98.58%              1.31kB / 0B         1.84MB / 484MB      5
[root@ubuntu1804 ~]#c


[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/b655d88228c04d7db6a6ad833ed3d05d4cd596ef09834382e17
942db0295dc0c/memory.oom_control
oom_kill_disable 1
under_oom 1
oom_kill 0
[root@ubuntu1804 ~]#
```

交换分区限制:

```
# docker run -it --rm  -m 256m  --memory-swap 512m  --name magedu-c1  centos
bash

# cat /sys/fs/cgroup/memory/docker/容器 ID/memory.memsw.limit_in_bytes 536870912
#返回值
```

范例:

```
[root@ubuntu1804 ~]#docker run -it --rm  -m 200m --memory-swap 512m
lorel/docker-stress-ng --vm 2
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 2 vm

#宿主机cgroup验证:
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/memory/docker/23733a0cafa21f3e94ca8c96110978b12e53076261f1b92fd20
52bafe659c8ab/memory.memsw.limit_in_bytes
536870912
```

kubernets 对swap的要求

K8s 1.8.3更新日志:
宿主机开启交换分区，会在安装之前的预检查环节提示相应错误信息: https://github.com/kubernetes/kubernetes/blob/release-1.8/CHANGELOG-1.8.md

Changelog since v1.8.3

Other notable changes

- Cluster Autoscaler 1.0.3 (#55947, @aleksandra-malinowska)
- ◦ Add PodSecurityPolicies for cluster addons (#55509, @tallclair)
  - ▪ Remove SSL cert HostPath volumes from heapster addons
- Fix session affinity issue with external load balancer traffic when ExternalTrafficPolicy=Local. (#55519, @MrHohn)
- Addon manager supports HA masters. (#55782, @x13n)
- ScaleIO persistent volumes now support referencing a secret in a namespace other than the bound persistent volume claim's namespace; this is controlled during provisioning with the `secretNamespace` storage class parameter; StoragePool and ProtectionDomain attributes no longer defaults to the value `default` (#54013, @vladimirvivien)
- Allow HPA to read custom metrics. (#54854, @kawych)
- Add masquerading rules by default to GCE/GKE (#55178, @dnardo)
- kubeadm now produces error during preflight checks if swap is enabled. Users, who can setup kubelet to run in unsupported environment with enabled swap, will be able to skip that preflight check. (#55399, @kad)
- GCE: provide an option to disable docker's live-restore on COS/ubuntu (#55260, @yujuhong)
- Fix hyperkube kubelet --experimental-dockershim (#55250, @ivan4th)
- ScaleIO driver completely removes dependency on drv_cfg binary so a Kubernetes cluster can easily run a containerized kubelet. (#54956, @vladimirvivien)

7.3 容器的CPU限制

7.3.1 容器的CPU限制介绍

官方文档说明: https://docs.docker.com/config/containers/resource_constraints/

一个宿主机，有几十个核心的CPU，但是宿主机上可以同时运行成百上千个不同的进程用以处理不同的任务，多进程共用一个 CPU 的核心为可压缩资源，即一个核心的 CPU 可以通过调度而运行多个进程，但是同一个单位时间内只能有一个进程在 CPU 上运行，那么这么多的进程怎么在 CPU 上执行和调度的呢?

Linux kernel 进程的调度基于CFS(Completely Fair Scheduler)，完全公平调度

服务器资源密集型

- CPU 密集型的场景: 优先级越低越好，计算密集型任务的特点是要进行大量的计算，消耗CPU 资源，比如计算圆周率、数据处理、对视频进行高清解码等等，全靠CPU 的运算能力。
- IO 密集型的场景: 优先级值高点，涉及到网络、磁盘IO 的任务都是IO 密集型任务，这类任务的特点是 CPU 消耗很少，任务的大部分时间都在等待 IO 操作完成（因为 IO 的速度远远低于 CPU 和内存的速度），比如 Web 应用，高并发，数据量大的动态网站来说，数据库应该为IO 密集型

CFS原理

cfs定义了进程调度的新模型，它给cfs_rq（cfs的run queue）中的每一个进程安排一个虚拟时钟vruntime。如果一个进程得以执行，随着时间的增长，其vruntime将不断增大。没有得到执行的进程vruntime不变, 而调度器总是选择vruntime跑得最慢的那个进程来执行。这就是所谓的“完全公平”。为了区别不同优先级的进程，优先级高的进程vruntime增长得慢，以至于它可能得到更多的运行机会。CFS的意义在于， 在一个混杂着大量计算型进程和IO交互进程的系统中，CFS调度器相对其它调度器在对待IO交互进程要更加友善和公平。

7.3.2 配置默认的CFS调度程序

默认情况下，每个容器对主机的CPU周期的访问都是不受限制的。可以设置各种约束，以限制给定容器对主机CPU周期的访问。大多数用户使用并配置 默认的CFS调度程序。在Docker 1.13及更高版本中，还可以配置 realtime scheduler。

CFS是用于常规Linux进程的Linux内核CPU调度程序。通过几个运行时标志,可以配置对容器拥有的CPU资源的访问量。使用这些设置时，Docker会在主机上修改容器cgroup的设置。

选项	描述
`--cpus=`	指定一个容器可以使用多少个可用的CPU核心资源。例如，如果主机有两个CPU，如果设置了`--cpus="1.5"`，则可以保证容器最多使用1.5个的CPU(如果是4核CPU，那么还可以是4核心上每核用一点，但是总计是1.5核心的CPU)。这相当于设置`--cpu-period="100000"`和`--cpu-quota="150000"`。此设置可在Docker 1.13及更高版本中可用，目的是替代--cpu-period和--cpu-quota两个参数，从而使配置更简单，但是最大不能超出宿主机的CPU总核心数(在操作系统看到的CPU超线程后的数值)，此项较常用
`--cpu-period=`	过时选项,指定CPU CFS调度程序周期，必须与`--cpu-quota`一起使用 。默认为100微秒。大多数用户不会更改默认设置。如果使用Docker 1.13或更高版本，请改用`--cpus`
`--cpu-quota=`	过时选项,在容器上添加 CPU CFS 配额，计算方式为 cpu-quota / cpu-period的结果值，docker1.13 及以上版本通常使用--cpus 设置此值
`--cpuset-cpus`	用于指定容器运行的 CPU 编号，也就是所谓的CPU绑定。如果一个或多个CPU，则容器可以使用逗号分隔的列表或用连字符分隔的CPU范围。第一个CPU的编号为0。有效值可能是`0-3`（使用第一，第二，第三和第四CPU）或`1,3`（使用第二和第四CPU）
`--cpu-shares`	用于设置 cfs 中调度的相对最大比例权重,cpu-share 的值越高的容器，将会分得更多的时间片(宿主机多核 CPU 总数为 100%，假如容器 A 为1024，容器 B 为 2048，那么容器 B 将最大是容器 A 的可用 CPU 的两倍)，默认的时间片1024，最大 262144。这是一个软限制。

7.3.3 使用stress-ng测试cpu配置

范例: 查看 stress-n 关于cpu的帮助

```
[root@ubuntu1804 ~]#docker run -it --rm  --name magedu-c1  lorel/docker-stress-ng  |grep cpu
 -c N, --cpu N             start N workers spinning on sqrt(rand())
       --cpu-ops N         stop when N cpu bogo operations completed
 -l P, --cpu-load P        load CPU by P %%, 0=sleep, 100=full load (see -c)
       --cpu-method m      specify stress cpu method m, default is all
Example: stress-ng --cpu 8 --io 4 --vm 2 --vm-bytes 128M --fork 4 --timeout 10s
[root@ubuntu1804 ~]#
```

范例: 不限制容器CPU

```
[root@ubuntu1804 ~]#lscpu |grep CPU
CPU op-mode(s):      32-bit, 64-bit
CPU(s):              6
On-line CPU(s) list: 0-5
CPU family:          6
Model name:          Intel(R) Core(TM) i7-4710HQ CPU @ 2.50GHz
CPU MHz:             2494.236
NUMA node0 CPU(s):   0-5

#占用4个CPU资源.但只是平均的使用CPU资源
[root@ubuntu1804 ~]#docker run -it --rm   lorel/docker-stress-ng  --cpu 4
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 4 cpu, 4 vm

[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
  MEM %               NET I/O             BLOCK I/O           PIDS
818a85e1da2f        frosty_taussig      595.57%             1.037GiB / 2.908GiB
  35.64%              1.12kB / 0B         0B / 0B             13
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/cpuset/docker/818a85e1da2f9a4ef297178a9dc09b338b2308108195ad8d4197a1c47febcbff/cpuset.cpus
0-5
[root@ubuntu1804 ~]#top
```

```
top - 12:44:32 up 39 min,  2 users,  load average: 7.56, 3.65, 1.84
Tasks: 231 total,   9 running, 127 sleeping,   0 stopped,   0 zombie
%Cpu0  : 99.7 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  : 99.7 us,  0.3 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 99.3 us,  0.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3049520 total,  1148964 free,  1418420 used,   482136 buff/cache
KiB Swap:  1951740 total,  1951740 free,        0 used.  1458912 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 2971 root      20   0  268412 265872    544 R  83.3  8.7   2:09.65 stress-ng-vm
 2964 root      20   0    6908   6860   2812 R  80.7  0.2   2:08.47 stress-ng-cpu
 2962 root      20   0    6908   6860   2812 R  77.3  0.2   2:07.97 stress-ng-cpu
 2973 root      20   0  268412 265872    544 R  75.0  8.7   2:10.57 stress-ng-vm
 2972 root      20   0  268412 265872    544 R  73.7  8.7   2:08.50 stress-ng-vm
 2958 root      20   0    6908   6860   2812 R  73.3  0.2   2:08.97 stress-ng-cpu
 2960 root      20   0    6908   6860   2812 R  72.0  0.2   2:07.81 stress-ng-cpu
 2970 root      20   0  268412 265872    544 R  69.7  8.7   2:05.37 stress-ng-vm
 3037 root      20   0   44212   4084   3392 R   1.0  0.1   0:00.29 top
    8 root      20   0       0      0      0 I   0.3  0.0   0:00.70 rcu_sched
  561 root      20   0  192984  10656   9392 S   0.3  0.3   0:02.81 vmtoolsd
 2727 root      20   0       0      0      0 I   0.3  0.0   0:00.39 kworker/1:3
    1 root      20   0   77696   8916   6716 S   0.0  0.3   0:02.85 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    4 root       0 -20       0      0      0 I   0.0  0.0   0:00.00 kworker/0:0H
```

范例：限制使用CPU

```
[root@ubuntu1804 ~]#docker run -it --rm  --cpus 1.5  lorel/docker-stress-ng  --cpu 4
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 4 cpu, 4 vm


[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
  MEM %               NET I/O             BLOCK I/O           PIDS
9f8b2e693113        busy_hodgkin        147.71%             786.8MiB / 2.908GiB
  26.42%              836B / 0B           0B / 0B             13
[root@ubuntu1804 ~]#top
```

```
top - 12:55:48 up 51 min,  2 users,  load average: 4.55, 6.53, 4.69
Tasks: 233 total,   9 running, 127 sleeping,   0 stopped,   0 zombie
%Cpu0  : 19.2 us,  5.6 sy,  0.0 ni, 75.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 23.3 us,  1.7 sy,  0.0 ni, 75.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 25.9 us,  0.0 sy,  0.0 ni, 74.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  : 24.5 us,  2.0 sy,  0.0 ni, 73.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 25.2 us,  0.3 sy,  0.0 ni, 74.4 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 24.9 us,  0.0 sy,  0.0 ni, 75.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3049520 total,  1192040 free,  1374788 used,   482692 buff/cache
KiB Swap:  1951740 total,  1951740 free,        0 used.  1502540 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3257 root      20   0    6908   6860   2812 R  25.9  0.2   0:13.94 stress-ng-cpu
 3261 root      20   0    6908   6848   2812 R  25.2  0.2   0:15.93 stress-ng-cpu
 3263 root      20   0    6908   6860   2812 R  24.9  0.2   0:16.06 stress-ng-cpu
 3270 root      20   0  301184 298880    552 R  21.6  9.8   0:10.04 stress-ng-vm
 3268 root      20   0  301184 298880    552 R  16.9  9.8   0:09.57 stress-ng-vm
 3269 root      20   0  268412 265880    552 R  12.6  8.7   0:07.82 stress-ng-vm
 3271 root      20   0  268412 160808    552 R  12.6  5.3   0:07.75 stress-ng-vm
 3259 root      20   0    6908   6860   2812 R  12.3  0.2   0:14.01 stress-ng-cpu
 3320 root      20   0   44212   4112   3420 R   0.3  0.1   0:00.07 top
```

范例：限制CPU

```
[root@ubuntu1804 ~]#docker run -it --rm  --cpu-quota 2000 --cpu-period 1000
lorel/docker-stress-ng  --cpu 4
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 4 cpu, 4 vm

[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                      CPU %               MEM USAGE /
LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
bd949bb6698e        affectionate_chebyshev    185.03%             1.037GiB /
2.908GiB   35.64%              836B / 0B           0B / 0B             13
[root@ubuntu1804 ~]#
```

范例: 绑定CPU

```
#一般不建议绑在0号CPU上，因0号CPU一般会较忙
[root@ubuntu1804 ~]#docker run -it --rm  --cpus 1.5  --cpuset-cpus 2,4-5
lorel/docker-stress-ng  --cpu 4
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 4 cpu, 4 vm

[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
  MEM %               NET I/O             BLOCK I/O           PIDS
585879094e73        hungry_albattani    154.35%             1.099GiB / 2.908GiB
  37.79%              906B / 0B           0B / 0B             13
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/cpuset/docker/585879094e7382d2ef700947b4454426eee7f943f8d1438fe42
ce34df789227b/cpuset.cpus
2,4-5
[root@ubuntu1804 ~]#top
```

```
top - 13:00:22 up 55 min,  2 users,  load average: 0.77, 3.21, 3.76
Tasks: 231 total,   9 running, 127 sleeping,   0 stopped,   0 zombie
%Cpu0  :  0.0 us,  0.0 sy,  0.0 ni, 99.7 id,  0.0 wa,  0.0 hi,  0.3 si,  0.0 st
%Cpu1  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  : 46.0 us,  3.7 sy,  0.0 ni, 50.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu4  : 46.5 us,  4.0 sy,  0.0 ni, 49.5 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu5  : 43.9 us,  6.3 sy,  0.0 ni, 49.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3049520 total,  1227892 free,  1338752 used,   482876 buff/cache
KiB Swap:  1951740 total,  1951740 free,        0 used.  1537404 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 3482 root      20   0  268412 265912    584 R  25.2  8.7   0:04.30 stress-ng-vm
 3484 root      20   0  301184 298912    584 R  24.9  9.8   0:07.41 stress-ng-vm
 3485 root      20   0  268412 183016    584 R  16.9  6.0   0:02.79 stress-ng-vm
 3469 root      20   0    6916   6860   2812 R  16.6  0.2   0:03.33 stress-ng-cpu
 3471 root      20   0    6916   6860   2812 R  16.6  0.2   0:02.80 stress-ng-cpu
 3473 root      20   0    6916   6860   2812 R  16.6  0.2   0:02.79 stress-ng-cpu
 3483 root      20   0  268412 238192    584 R  16.6  7.8   0:03.56 stress-ng-vm
 3475 root      20   0    6916   6860   2812 R  16.3  0.2   0:02.79 stress-ng-cpu
 3378 root      20   0  859560  62156  34732 S   0.3  2.0   0:00.07 docker
    1 root      20   0   77696   8916   6716 S   0.0  0.3   0:02.90 systemd
```

范例: 多个容器的CPU利用率比例

```
#同时开两个容器
[root@ubuntu1804 ~]#docker run -it --rm  --name c1 --cpu-shares 1000
lorel/docker-stress-ng  --cpu 4
stress-ng: info: [1] defaulting to a 86400 second run per stressor
```

```
stress-ng: info: [1] dispatching hogs: 4 cpu, 4 vm
[root@ubuntu1804 ~]#docker run -it --rm  --name c2 --cpu-shares 500
lorel/docker-stress-ng  --cpu 4
stress-ng: info: [1] defaulting to a 86400 second run per stressor
stress-ng: info: [1] dispatching hogs: 4 cpu, 4 vm



[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
  MEM %               NET I/O             BLOCK I/O           PIDS
a1d4c6e6802d        c2                  195.88%             925.3MiB / 2.908GiB
  31.07%              726B / 0B           0B / 0B             13
d5944104aff4        c1                  398.20%             1.036GiB / 2.908GiB
  35.64%              906B / 0B           0B / 0B             13
[root@ubuntu1804 ~]#

#查看c1容器的cpu利用比例
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/cpu,cpuacct/docker/d5944104aff40b7b76f536c45a68cd4b98ce466a73416b
68819b9643e3f49da7/cpu.shares
1000

#查看c2容器的cpu利用比例
[root@ubuntu1804 ~]#cat
/sys/fs/cgroup/cpu,cpuacct/docker/a1d4c6e6802d1b846b33075f3c1e1696376009e85d9ff8
756f9a8d93d3da3ca6/cpu.shares
500

#再打开新的容器，cpu分配比例会动态调整
[root@ubuntu1804 ~]#docker run -it --rm  --name c3 --cpu-shares 2000
lorel/docker-stress-ng  --cpu 4

[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
  MEM %               NET I/O             BLOCK I/O           PIDS
c2d54818e1fe        c3                  360.15%             664.5MiB / 2.908GiB
  22.31%              726B / 0B           1.64GB / 150MB      13
a1d4c6e6802d        c2                  82.94%              845.2MiB / 2.908GiB
  28.38%              936B / 0B           103MB / 4.54MB      13
d5944104aff4        c1                  181.18%             930.1MiB / 2.908GiB
  31.23%              1.12kB / 0B         303MB / 19.8MB      13
```

范例：动态调整cpu shares值

```
[root@ubuntu1804 ~]#echo 2000 >
/sys/fs/cgroup/cpu,cpuacct/docker/a1d4c6e6802d1b846b33075f3c1e1696376009e85d9ff8
756f9a8d93d3da3ca6/cpu.shares
[root@ubuntu1804 ~]#docker stats --no-stream
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT
  MEM %               NET I/O             BLOCK I/O           PIDS
a1d4c6e6802d        c2                  389.31%             1.037GiB / 2.908GiB
  35.64%              1.01kB / 0B         1.16GB / 14MB       13
d5944104aff4        c1                  200.28%             1.036GiB / 2.908GiB
  35.63%              1.19kB / 0B         2.66GB / 26.7MB     13
[root@ubuntu1804 ~]#
```

8 可视化图形工具Portainer

8.1 Portainer介绍

Portainer是一个可视化的容器镜像的图形管理工具，利用Portainer可以轻松构建，管理和维护Docker环境。 而且完全免费，基于容器化的安装方式，方便高效部署。

官方站点: https://www.portainer.io/

8.2 安装 Portainer

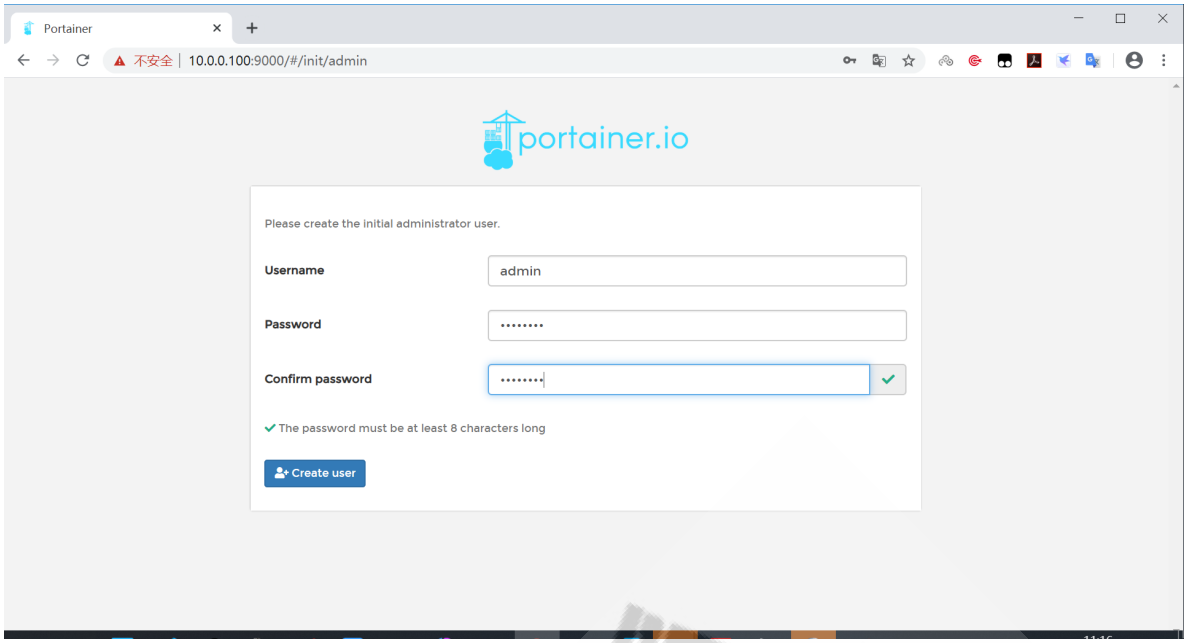
官方安装说明: https://www.portainer.io/installation/

```
[root@ubuntu1804 ~]#docker search portainer |head -n 3
NAME                  DESCRIPTION                     STARS       OFFICIAL
AUTOMATED
portainer/portainer   Making Docker management easy. https://porta…   1569

portainer/agent       An agent used to manage all the resources in…   54
     0

#portainer项目废弃
[root@ubuntu1804 ~]#docker pull  portainer/portainer

#portainer-ce项目代替portainer
[root@ubuntu1804 ~]#docker pull  portainer/portainer-ce

[root@ubuntu1804 ~]#docker volume create portainer_data
portainer_data
[root@ubuntu1804 ~]#docker run -d -p 8000:8000 -p 9000:9000 --name=portainer --
restart=always -v /var/run/docker.sock:/var/run/docker.sock -v
portainer_data:/data portainer/portainer-ce
20db26b67b791648c2ef6aee444a5226a9c897ebcf0160050e722dbf4a4906e3
[root@ubuntu1804 ~]#docker ps
CONTAINER ID        IMAGE                  COMMAND             CREATED
  STATUS              PORTS                                              NAMES
20db26b67b79        portainer/portainer    "/portainer"        5 seconds ago
  Up 4 seconds        0.0.0.0:8000->8000/tcp, 0.0.0.0:9000->9000/tcp   portainer
```

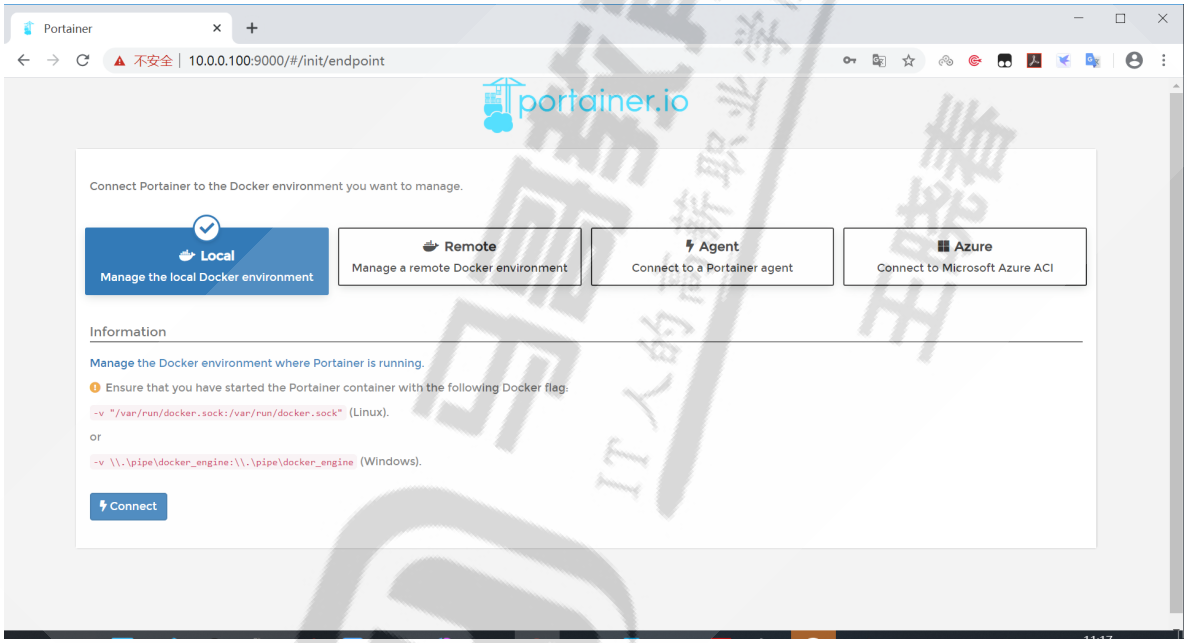
8.3 登录和使用Portainer

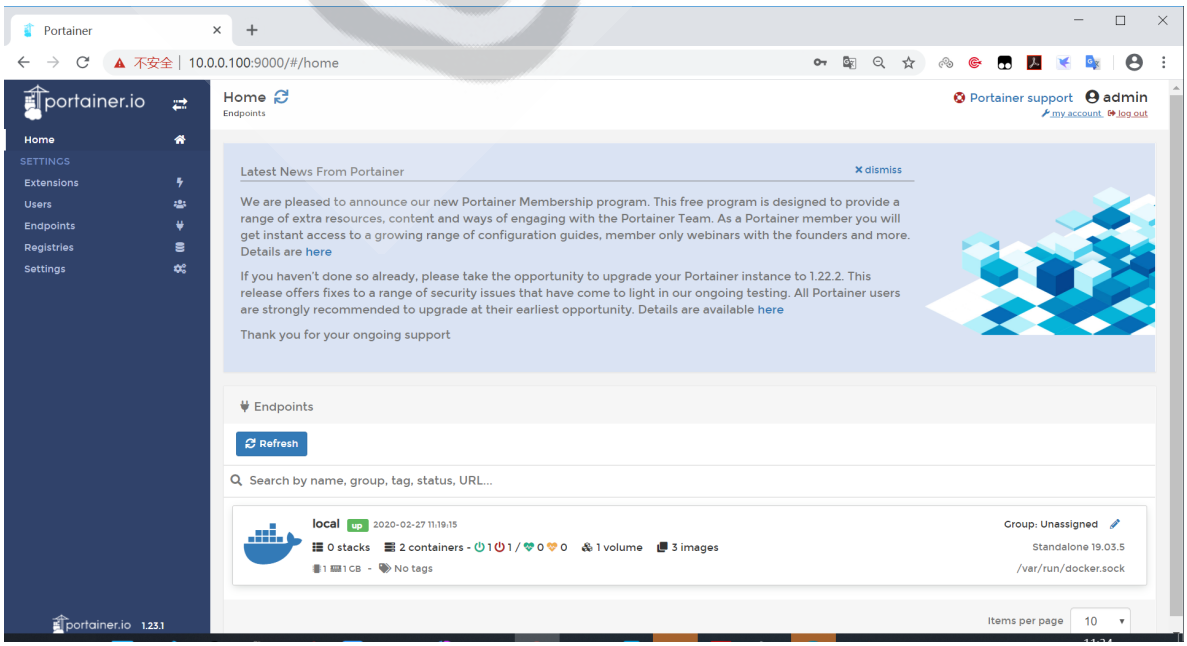
用浏览器访问: `http://localhost:9000` 可以看到以下界面

设置admin用户密码，需要输入两次超过8个字符的相同的密码

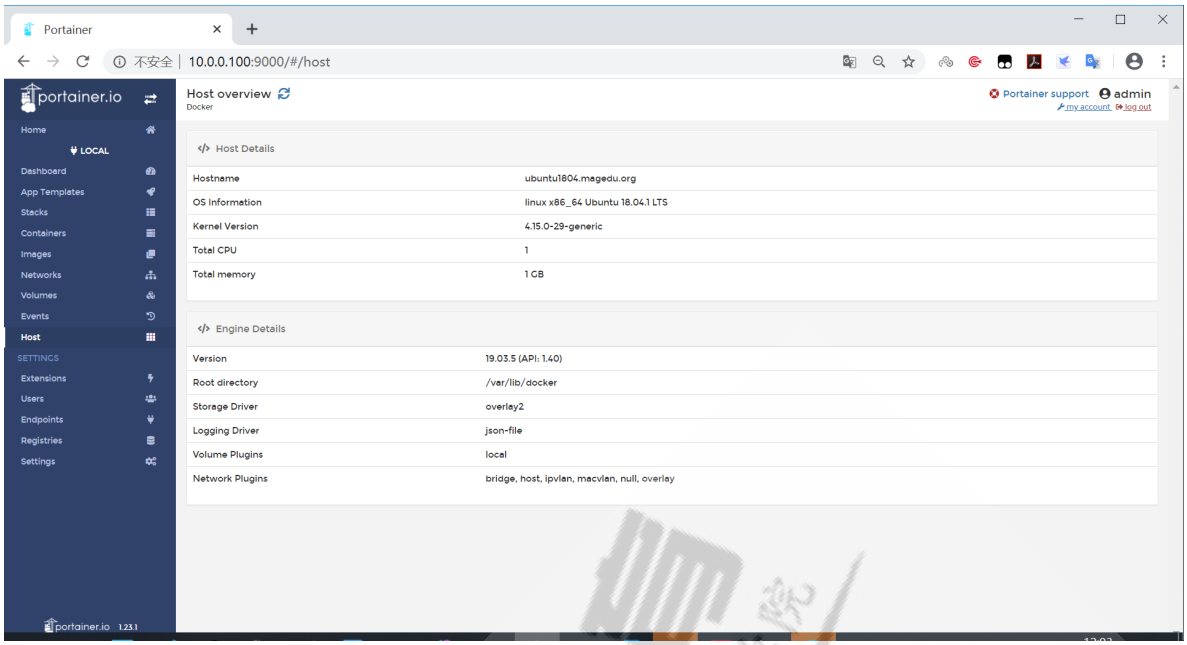


以下界面中，选择`local`，再点击`Connect`

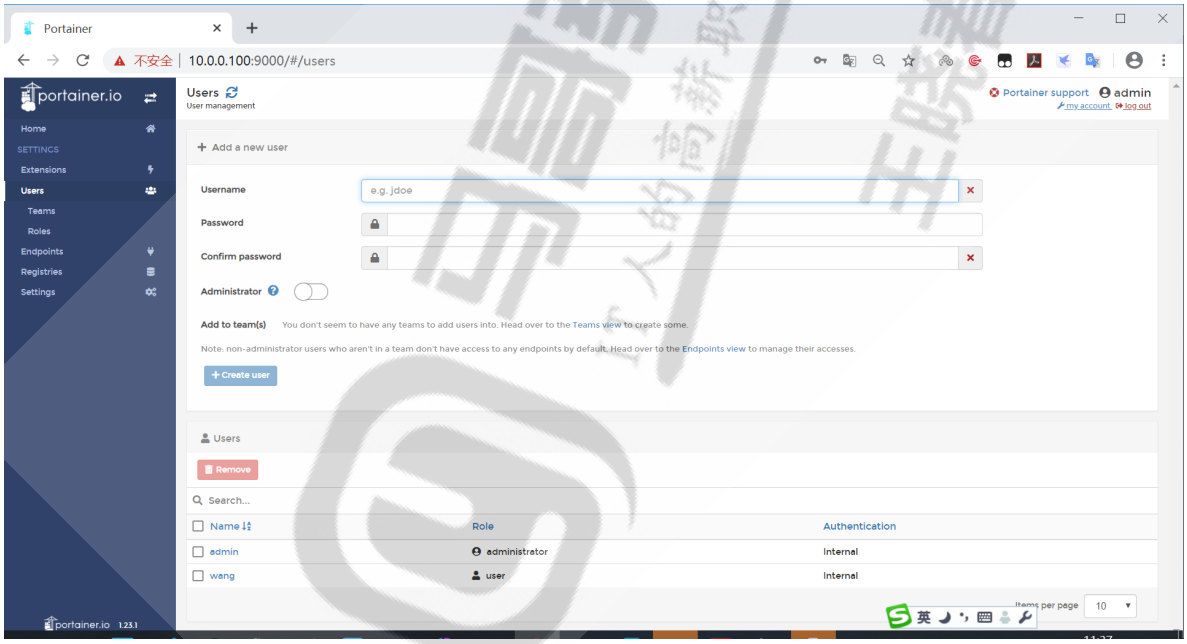




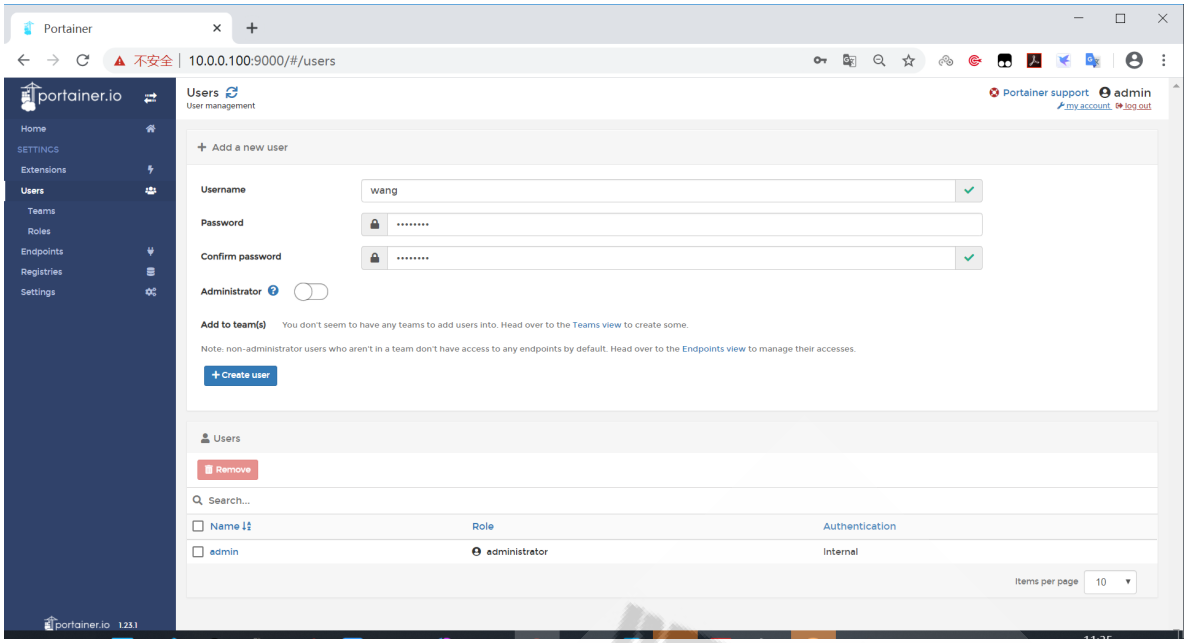
8.4 查看主机信息



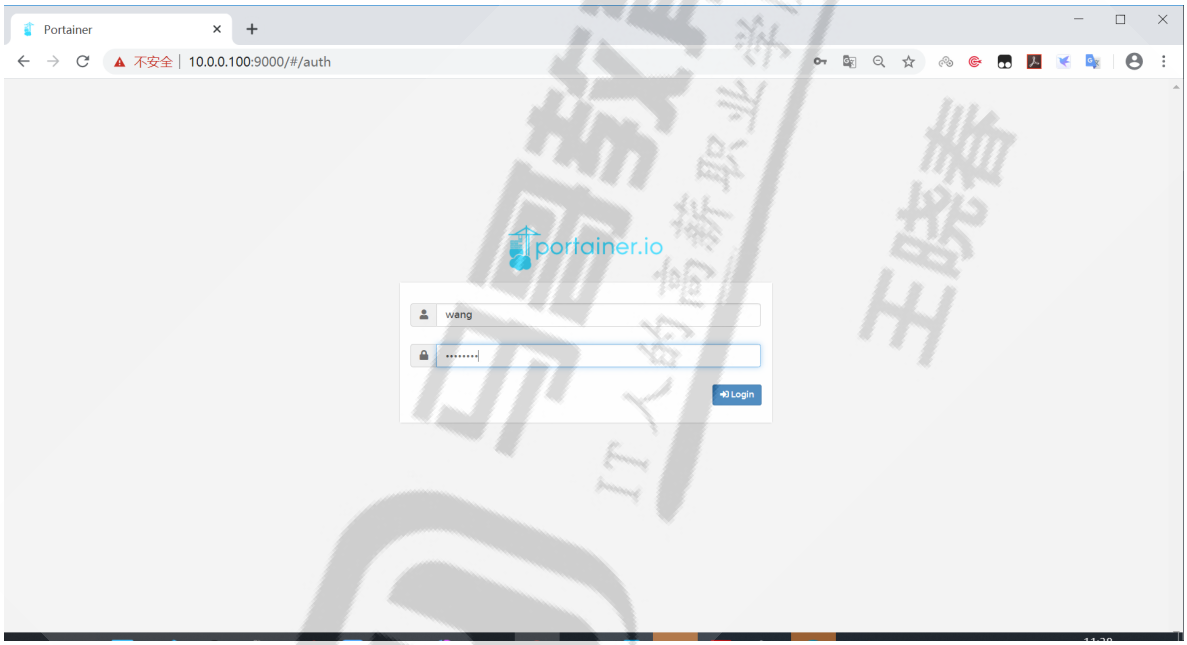
8.5 创建portainer用户



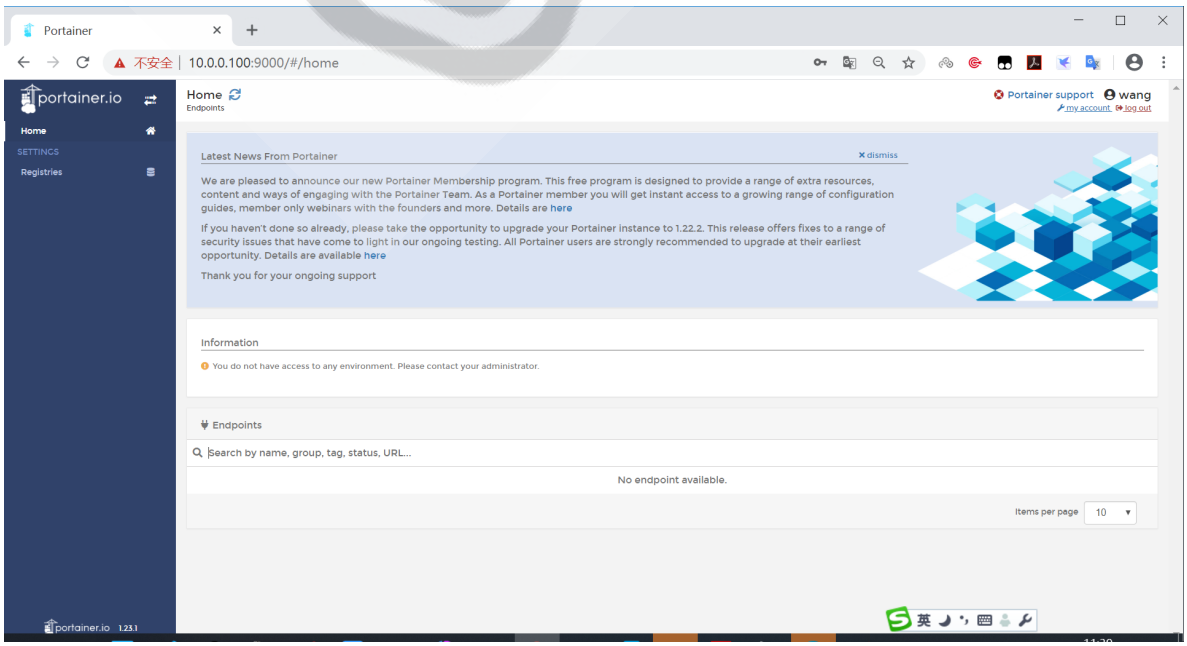
输入新用户信息



用新建的用户登录

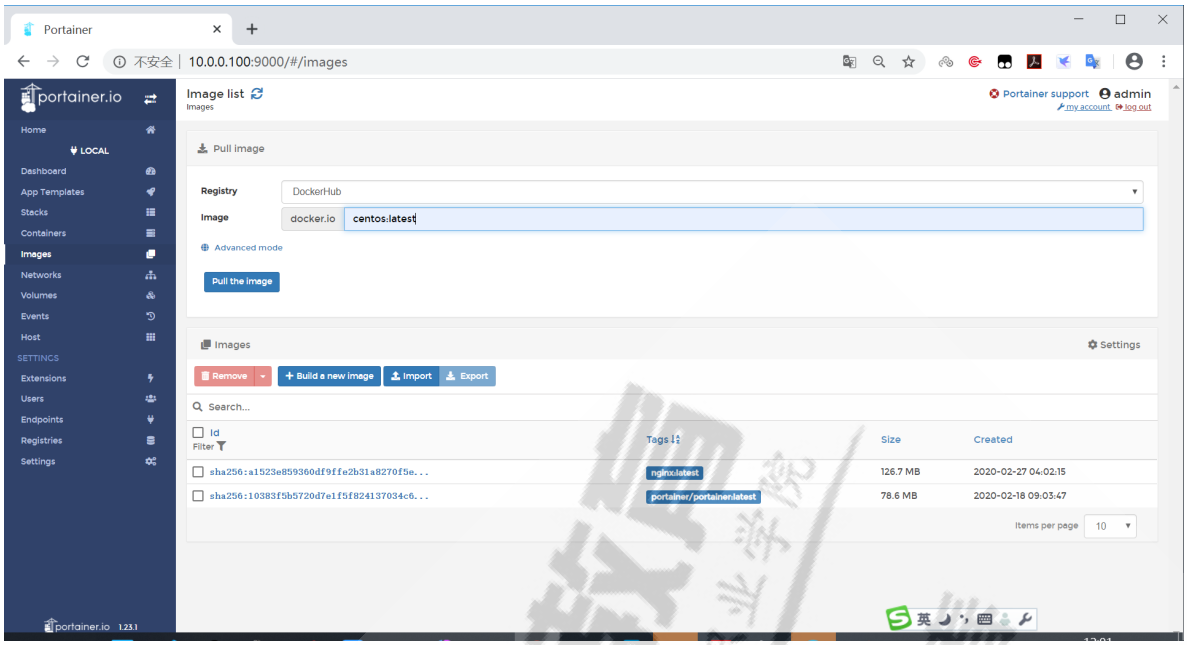


普通用户权限较小，无法管理容器



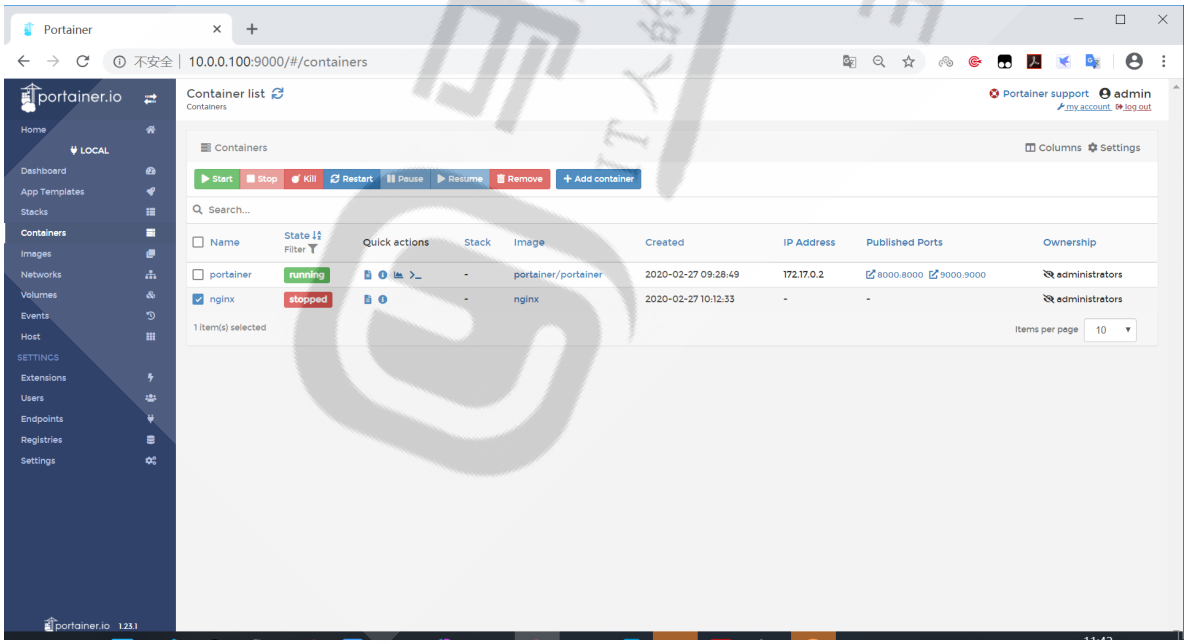
8.6 管理镜像

可以拉取，上传，构建等管理镜像

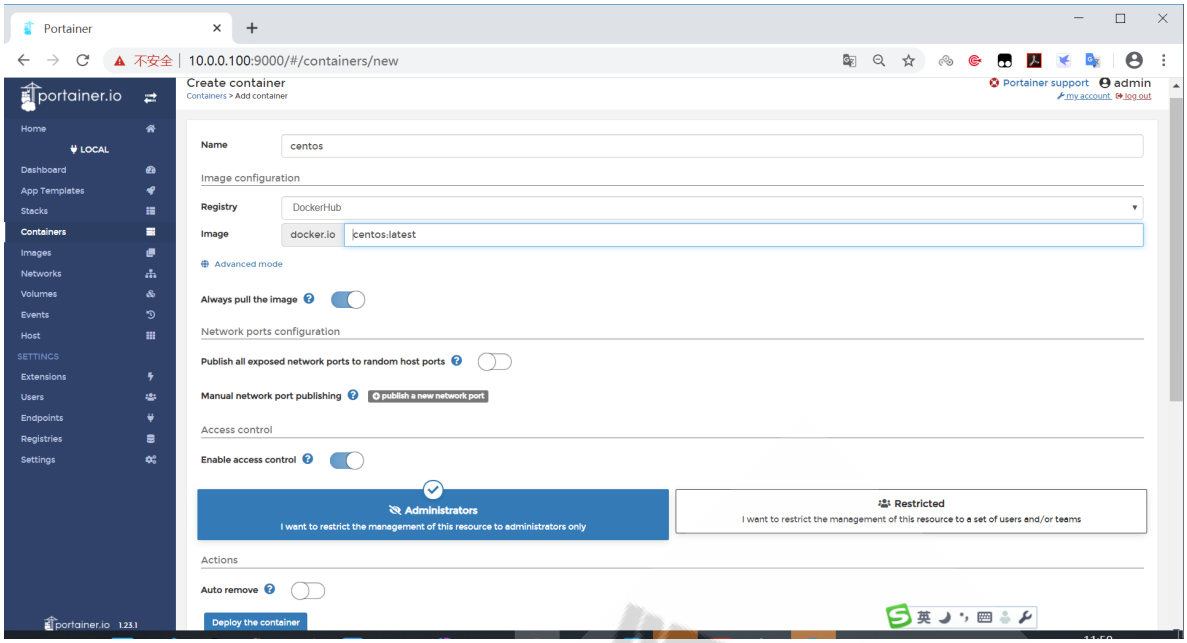


8.7 管理容器

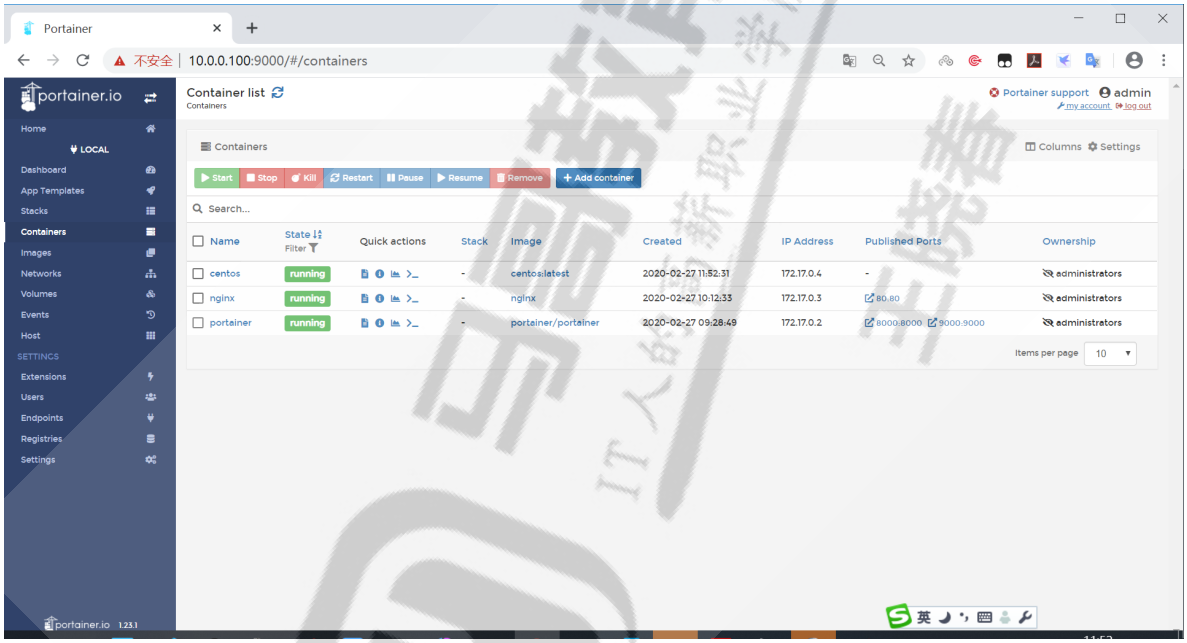
可以创建，删除启动和停止容器等



创建新容器



查看创建的容器



docker 命令总结

```
attach                  # 当前shell下attach连接指定运行镜像
build                   # 通过dockerfile定制镜像
commit                  # 提交当前容器为新的镜像
cp                      # 从容器中拷贝指定文件或者目录到宿主机中
create                  # 创建一个新的容器，同run 但不启动容器
diff                    # 查看docker 容器变化
events                  # 从docker 服务获取容器实时事件
exec                    # 在已存在的容器上运行命令
export                  # 导出容器的内容作为一个 tar 归档文件[对应import]
history                 # 展示一个镜像形成历史
images                  # 列出系统当前镜像
import                  # 从tar包中的内容创建一个新的文件系统映像[对应export]
info                    # 显示系统相关信息
inspect                 # 查看容器详细信息
kill                    # kill 指定容器
```

```
load                    # 从一个tar 包中加载一个镜像[对应save]
login                   # 注册或者登陆一个docker源服务器
logout                  # 从当前docker registry退出
logs                    # 输出当前容器日志信息
port                    # 查看映射端口对应的容器内部源端口
pause                   # 暂停容器
ps                      # 列出容器列表
pull                    # 从docker镜像源服务器拉取指定镜像或者库镜像
push                    # 推送指定镜像或者库镜像至docker源服务器
restart                 # 重启运行的容器
rm                      # 移除一个或者多个容器
rmi                     # 移除一个或多个镜像[无容器使用该镜像才可删除，否则需要删除相关容器
才可继续或 -f 强制删除]
run                     # 创建一个新的容器并运行一个命令
save                    # 保存一个镜像为一个tar包[对应load]
search                  # 在docker hub 中搜索镜像
start                   # 启动容器
stop                    # 停止容器
tag                     # 给源中镜像打标签
top                     # 查看容器中运行的进程信息
unpause                 # 取消暂停容器
version                 # 查看docker版本号
wait                    # 截取容器停止时的退出状态值
```