

Docker

容器化技术

1 虚拟化技术

我们要讲解的是容器化技术,那么为什么在这里需要先了解一下容器化技术呢? 那么因为 Docker 的容器化技术是虚拟化的一种体现形式,因此我们要学习容器化技术之前,需要先来了解一下什么是虚拟化技术.

1.1 什么是虚拟化技术

在计算机中，虚拟化（英语：Virtualization）是一种资源管理技术，是将计算机的各种实体资源，如服务器、网络、内存及存储等，予以抽象、转换后呈现出来，打破实体结构间的不可切割的障碍，使用户可以比原本的组态更好的方式来应用这些资源。这些资源的新虚拟部份是不受现有资源的架设方式，地域或物理组态所限制。一般所指的虚拟化资源包括计算能力和资料存储。

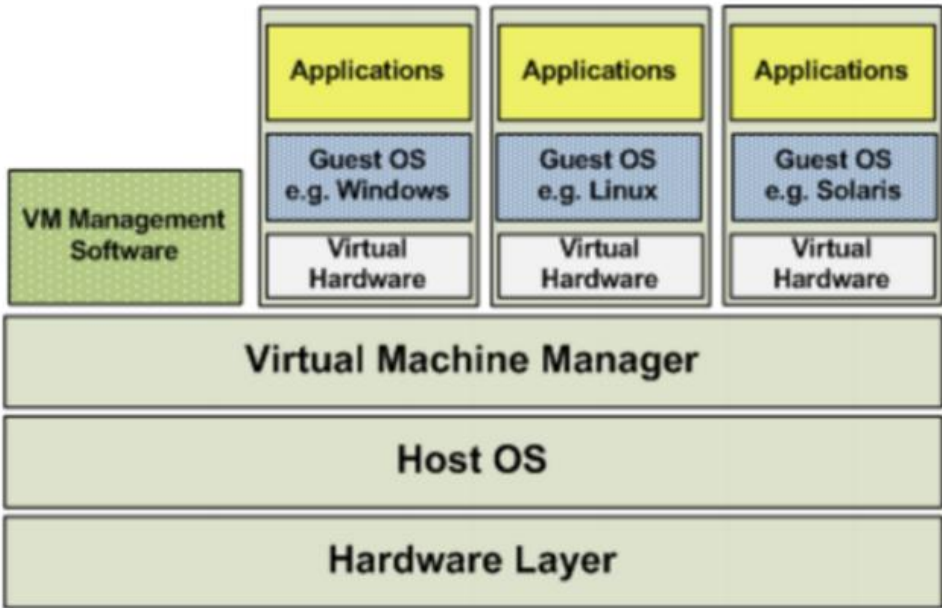
虚拟化技术种类很多，例如：软件虚拟化、硬件虚拟化、内存虚拟化、网络虚拟化(vip)、桌面虚拟化、服务虚拟化、虚拟机等等。

在实际的生产环境中，虚拟化技术主要用来解决高性能的物理硬件产能过剩和老的旧的硬件产能过低的重组重用，透明化底层物理硬件，从而最大化的利用物理硬件对资源充分利用

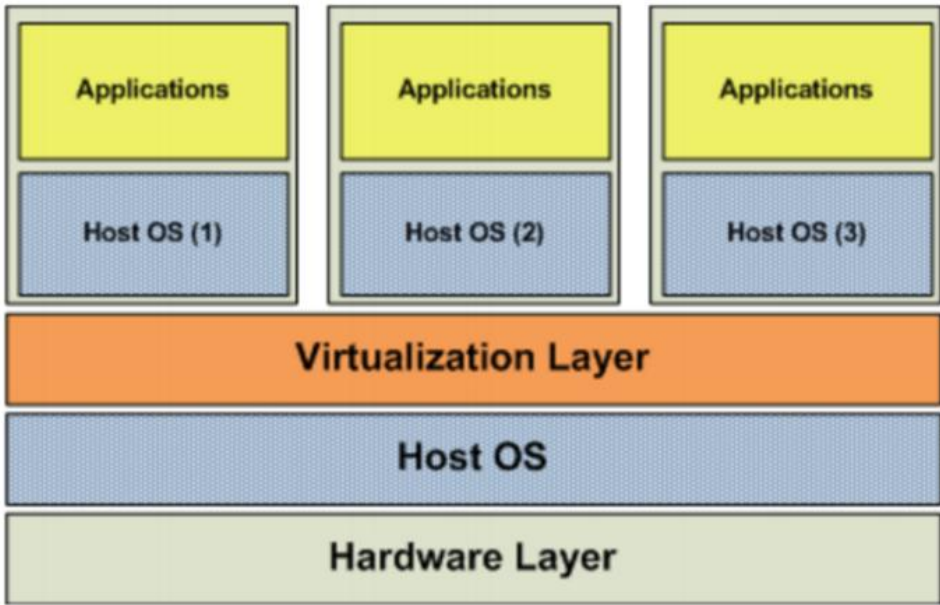
1.2 虚拟化常见架构

1.2.1 全虚拟化架构

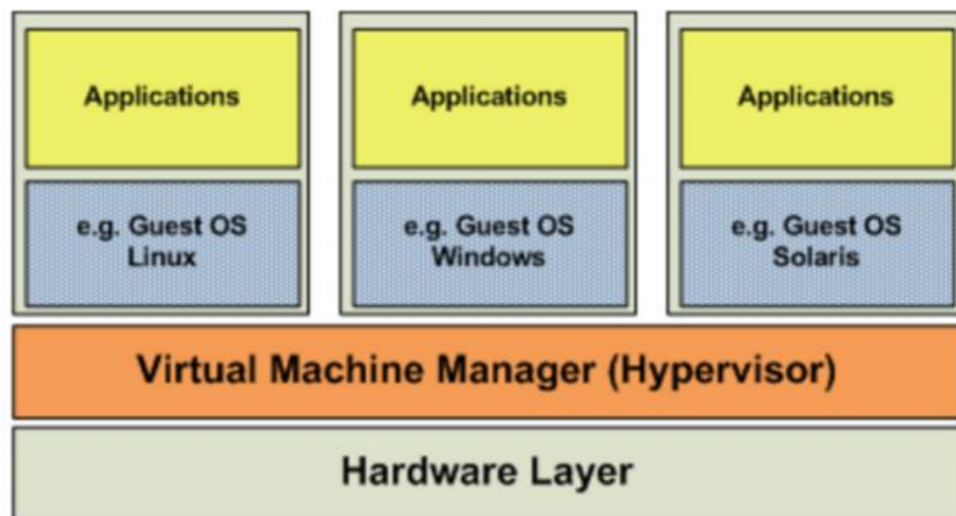
虚拟机的监视器（hypervisor）是类似于用户的应用程序运行在主机的 OS 之上，如 VMware 的 workstation，这种虚拟化产品提供了虚拟的硬件。



1.2.2 OS 层虚拟化架构



1.2.3 硬件层虚拟化



硬件层的虚拟化具有高性能和隔离性，因为 hypervisor 直接在硬件上运行，有利于控制 VM 的 OS 访问硬件资源，使用这种解决方案的产品有 VMware ESXi 和 Xen server

Hypervisor 是一种运行在物理服务器和操作系统之间的中间软件层，可允许多个操作系统和应用共享一套基础物理硬件，因此也可以看作是虚拟环境中的“元”操作系统，它可以协调访问服务器上的所有物理设备和虚拟机，也叫虚拟机监视器（Virtual Machine Monitor，VMM）。

Hypervisor 是所有虚拟化技术的核心。当服务器启动并执行 Hypervisor 时，它会给每一台虚拟机分配适量的内存、CPU、网络 and 磁盘，并加载所有虚拟机的客户操作系统。宿主机

Hypervisor 是所有虚拟化技术的核心，软硬件架构和管理更高效、更灵活，硬件的效能能够更好地发挥出来。常见的产品有：VMware、KVM、Xen 等等。Openstack

2 Docker 简介

2.1 什么是 Docker



在计算机的世界中，容器拥有一段漫长且传奇的历史。容器与管理程序虚拟化（hypervisor virtualization，HV）有所不同，管理程序虚拟化通过中间层将一台或者多台独立的机器虚拟运行与物理硬件之上，而容器则是直接运行在操作系统内核之上的用户空间。因此，容器虚拟化也被称为“操作系统级虚拟化”，容器技术可以让多个独立的用户空间运行在同一台宿主机上。

由于“客居”于操作系统，容器只能运行与底层宿主机相同或者相似的操作系统，这看起来并不是非常灵活。例如：可以在 Ubuntu 服务中运行 Redhat Enterprise Linux，但无法再 Ubuntu 服务器上运行 Microsoft Windows。

相对于彻底隔离的管理程序虚拟化，容器被认为是不安全的。而反对这一观点的人则认为，由于虚拟容器所虚拟的是一个完整的操作系统，这无疑增大了攻击范围，而且还要考虑管理程序层潜在的暴露风险。

尽管有诸多局限性，容器还是被广泛部署于各种各样的应用场合。在超大规模的多租户服务部署、轻量级沙盒以及对安全要求不太高的隔离环境中，容器技术非常流行。最常见的一个例子就是“权限隔离监牢”（chroot jail），它创建一个隔离的目录环境来运行进程。如果权限隔离监牢正在运行的进程被入侵者攻破，入侵者便会发现自己“身陷囹圄”，因为权限不足被困在容器所创建的目录中，无法对宿主机进一步破坏。

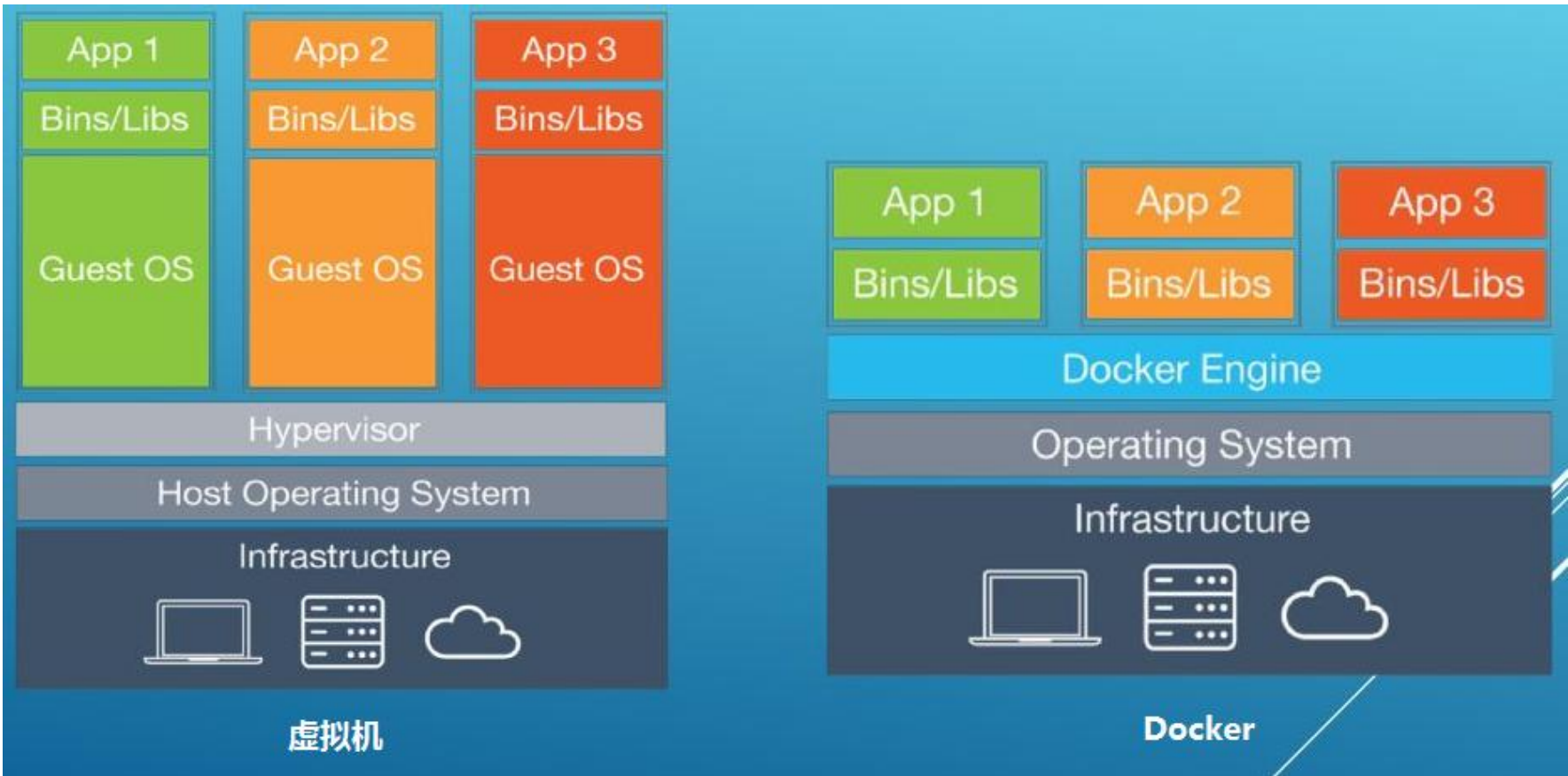
最新的容器技术引入了 OpenVZ、Solaris Zones 以及 Linux 容器（LXC）。使用这些新技术，容器不在仅仅是一个单纯的运行环境。在自己的权限类内，容器更像是一个完整的宿主机。对 Docker 来说，它得益于现代 Linux 特性，如控件组（control group）、命名空间（namespace）技术，容器和宿主机之间的隔离更加彻底，容器有独立的网络和存储栈，还拥有自己的资源管理能力，使得同一台宿主机中的多个容器可以友好的共存。

容器被认为是精益技术，因为容器需要的开销有限。和传统虚拟化以及半虚拟化相比，容器不需要模拟层（emulation layer）和管理层（hypervisor layer），而是使用操作系统的系统调用接口。这降低了运行单个容器所需的开销，也使得宿主机中可以运行更多的容器。

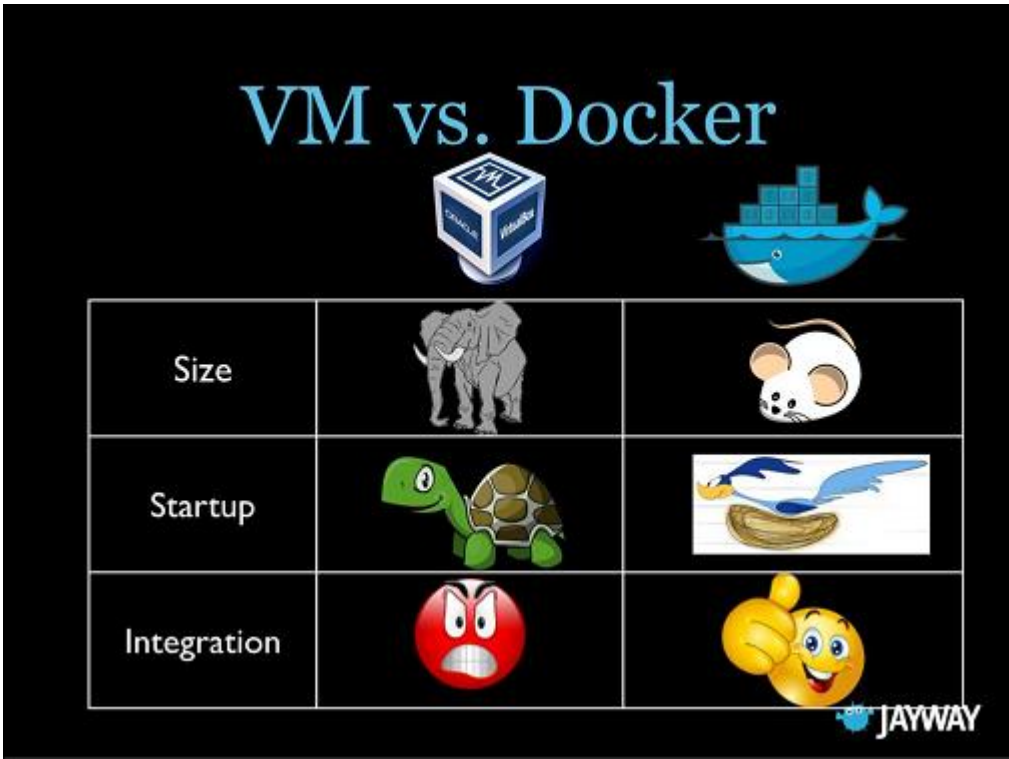
尽管有着光辉的历史，容器仍未得到广泛的认可。一个很重要的原因就是容器技术的复杂性：容器本身就比较复杂，不易安装，管理和自动化也很困难。而 Docker 就是为了改变这一切而生的。

2.2 虚拟化技术和 Docker 的区别

2.2.1 本质上区别



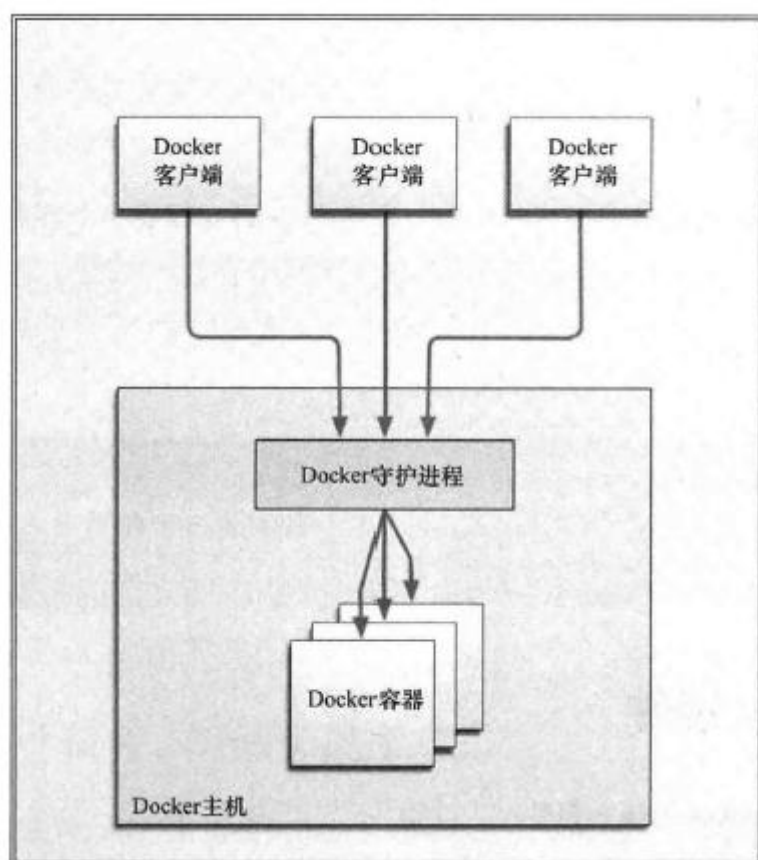
2.2.2 使用上的区别



3 Docker 组件

3.1 Docker 客户端和服务端

Docker 是一个客户端-服务器（C/S）架构程序。Docker 客户端只需要向 Docker 服务器或者守护进程发出请求，服务器或者守护进程将完成所有工作并返回结果。Docker 提供了一个命令行工具 Docker 以及一整套 RESTful API。你可以在同一台宿主机上运行 Docker 守护进程和客户端，也可以从本地的 Docker 客户端连接到运行在另一台宿主机上的远程 Docker 守护进程。



3.2 Docker 镜像

镜像是构建 Docker 的基石，其中包括了容器的文件系统结构与内容，它与 docker 的配置文件共同组成了 docker 容器的静态文件系统环境。用户基于镜像来运行自己的容器。

3.3 注册中心

我们从何处去获得镜像？如果我们是第一次通过某个镜像去启动容器，首先宿主机回去/var/lib/docker 目录下去找，如果没有找到，则会去 registry 中去下载镜像并且存放于虚拟机，然后完成启动。
registry 可以想象为一个镜像的仓库，默认的 registry 是 docker 官方提供的 registry 服务，名为 Docker Hub。当然，你也可以构建自己的镜像仓库。

3.4 Docker 容器

容器就是镜像的运行实例。
用户可以通过命令行或是 API 启动、停止、移动或删除容器。可以这么认为，对于应用软件，镜像是软件生命周期的构建和打包阶段，而容器则是启动和运行阶段。

4 Docker 的安装与启动

4.1 安装环境说明

Docker 官方建议在 Ubuntu 中安装，因为 Docker 是基于 Ubuntu 发布的，而且一般 Docker 出现的问题 Ubuntu 是最先更新或者打补丁的。在很多版本的 CentOS 中是不支持更新最新的一些补丁包的。
由于我们学习的环境都使用的是 CentOS，因此这里我们将 Docker 安装到 CentOS 上。注意：这里建议安装在 **CentOS7.x** 以上的版本，在 CentOS6.x 的版本中，安装前需要安装其他很多的环境而且 Docker 很多补丁不支持更新。

4.2 在 vm 中安装 Centos7

查看 ip 地址

```
ifconfig
```

```
ens33: flags=4163<UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
    inet 192.168.80.136 netmask 255.255.255.0 broadcast 192.168.80.255
    inet6 fe80::64ba:dea0:c4c3:6593 prefixlen 64 scopeid 0x20<link>
    ether 00:0c:29:e7:8d:e2 txqueuelen 1000 (Ethernet)
```

4.3 安装 Docker 与卸载

4.3.1 卸载 Docker

🔗 查看 docker 的安装包

```
yum list installed | grep docker
```

```
[root@pinyoyougou-docker docker]# yum list installed | grep docker
docker.x86_64                2:1.13.1-63.git94f4240.el7.centos @extras
docker-client.x86_64        2:1.13.1-63.git94f4240.el7.centos @extras
docker-common.x86_64        2:1.13.1-63.git94f4240.el7.centos @extras
```

🔗 删除安装包

```
yum -y remove docker.x86_64
yum -y remove docker-client.x86_64
yum -y remove docker-common.x86_64
```

🔗 删除 docker 镜像

```
rm -rf /var/lib/docker/
```

🔗 再次检查 Docker 是否已经卸载成功

```
[root@pinyoyougou-docker docker]# yum list installed | grep docker
[root@pinyoyougou-docker docker]#
```

4.3.2 Docker 的安装

🔗 使用 yum 命令进行在线安装 docker

```
yum install docker
```

```
[root@pinyoyougou-docker docker]# yum install docker
已加载插件: fastestmirror
Loading mirror speeds from cached hostfile
* base: mirrors.tuna.tsinghua.edu.cn
* extras: mirrors.huaweicloud.com
* updates: mirrors.tuna.tsinghua.edu.cn
```

🔗 提示信息

```
安装 1 软件包 (+2 依赖软件包)

总下载量: 20 M
安装大小: 69 M
Is this ok [y/d/N]:
```

🔗 输入 y 进行安装

```
Is this ok [y/d/N]: y
Downloading packages:
(1/3): docker-common-1.13.1-63.git94f4240.el7.centos.x86_64.rpm
(2/3): docker-client-1.13.1-63.git94f4240.el7.centos.x86_64.rpm
(3/3): docker-1.13.1-63.git94f4240.el7.centos.x86_64.rpm
```

4.4 检查 Docker 的版本

使用如下的命令查看 Docker 的版本

```
docker -v
```

```
[root@pinyoyougou-docker docker]# docker -v
Docker version 1.13.1, build 94f4240/1.13.1
```




4.5 启动与停止 Docker

systemctl 命令是系统服务管理器指令，它是 **service** 和 **chkconfig** 两个命令组合。

```
systemctl start    docker    启动 docker
systemctl status   docker    查看 docker 的启动状态
systemctl stop     docker    关闭 docker
systemctl restart  docker    重启 docker

docker info        查看 docker 的概要信息
docker -help       查看 docker 的帮助文档
```

启动 docker 的时候报错

```
[root@pinyougou-docker ~]# systemctl start docker
Job for docker.service failed because the control process exited with error code. See "systemctl status docker.service" and "journalctl -xe" for details.
```

查看 docker 的启动状态

```
[root@pinyougou-docker ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: failed (Result: exit-code) since 日 2018-06-24 11:28:30 CST; 1min 6s ago
     Docs: http://docs.docker.com
   Process: 2242 ExecStart=/usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default-runtime=docker-runc --exec-opt native.cgroupdriver=systemd --userland-proxy-path=/usr/libexec/docker/docker-proxy-current --init-path=/usr/libexec/docker/docker-init-current --seccomp-profile=/etc/docker/seccomp.json $OPTIONS $DOCKER_STORAGE_OPTIONS $DOCKER_NETWORK_OPTIONS $ADD_REGISTRY $BLOCK_REGISTRY $INSECURE_REGISTRY $REGISTRIES (code=exited, status=1/FAILURE)
 Main PID: 2242 (code=exited, status=1/FAILURE)

6月 24 11:28:28 pinyougou-docker systemd[1]: Starting Docker Application Container Engine...
6月 24 11:28:28 pinyougou-docker dockerd-current[2242]: time="2018-06-24T11:28:28.654527866+08:00" level=warning msg="could not change group /var/run/docker...ot found"
6月 24 11:28:28 pinyougou-docker dockerd-current[2242]: time="2018-06-24T11:28:28.664996575+08:00" level=info msg="libcontainerd: new containerd process, pid: 2246"
6月 24 11:28:30 pinyougou-docker dockerd-current[2242]: Error starting daemon: SELinux is not supported with the overlay2 graph driver on this kernel. Either...ed=false.
6月 24 11:28:30 pinyougou-docker systemd[1]: docker.service: main process exited, code=exited, status=1/FAILURE
6月 24 11:28:30 pinyougou-docker systemd[1]: Failed to start Docker Application Container Engine.
6月 24 11:28:30 pinyougou-docker systemd[1]: Unit docker.service entered failed state.
6月 24 11:28:30 pinyougou-docker systemd[1]: docker.service failed.
```

这里表示的意思是此 linux 的内核中的 SELinux 不支持 overlay2 graph driver，解决方法有两个，要么启动一个新内核，要么就在 docker 里禁用 selinux --selinux-enabled=false

我们的解决方案就是在 docker 里面禁用 selinux

```
vi /etc/sysconfig/docker
```

```
# Modify these options if you want to change the way the docker daemon runs
OPTIONS='--selinux-enabled=false --log-driver=journald --signature-verification=false'
if [ -z "${DOCKER_CERT_PATH}" ]; then
    DOCKER_CERT_PATH=/etc/docker
fi
```

再次启动 docker，并且查看 docker 的启动状态

```
[root@pinyougou-docker ~]# systemctl start docker
[root@pinyougou-docker ~]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since 日 2018-06-24 11:33:58 CST; 4s ago
     Docs: http://docs.docker.com
  Main PID: 2316 (dockerd-current)
    CGroup: /system.slice/docker.service
            └─2316 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/docker/docker-runc-current --default
               └─2321 /usr/bin/docker-containerd-current -l unix:///var/run/docker/libcontainerd/docker-containerd.sock --
```

我们看到 running 字样就说明 docker 启动成功了

5 Docker 镜像操作

5.1 列出镜像

列出宿主机上所存在的镜像

```
docker images
```

```
[root@pinyoyougou-docker ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
```

第一次安装 docker 的时候是没有镜像的,索引我们看到的镜像信息是空的. 给我们展示出来的每一列代表什么意思呢?

列名	含义	备注
REPOSITORY	镜像所在的仓库名称	
TAG	镜像标签	为了区分同一个仓库下的不同镜像，Docker 提供了一种称为标签（Tag）的功能。每个镜像在列出来时都带有一个标签，
IMAGE ID	镜像的 ID	
CREATED	创建时间	镜像的创建日期（不是获取该镜像的日期）
SIZE	镜像大小	

注: 这些镜像都是存储在 Docker 宿主机的/var/lib/docker 目录下

5.2 搜索镜像

如果你需要从网络中查找需要的镜像，可以通过以下命令搜索

```
docker search 镜像名称
```

搜索 tomcat 镜像

```
[root@pinyoyougou-docker docker]# docker search tomcat
INDEX      NAME                               DESCRIPTION                               STARS   OFFICIAL   AUTOMATED
docker.io  docker.io/tomcat                  Apache Tomcat is an open source implementa... 1891    [OK]
docker.io  docker.io/tomee                   Apache TomEE is an all-Apache Java EE cert... 51      [OK]
docker.io  docker.io/dordoka/tomcat          Ubuntu 14.04, Oracle JDK 8 and Tomcat 8 ba... 49
docker.io  docker.io/davidcaste/alpine-tomcat  Apache Tomcat 7/8 using Oracle Java 7/8 wi... 25
docker.io  docker.io/bitnami/tomcat          Bitnami Tomcat Docker Image                16
docker.io  docker.io/consol/tomcat-7.0       Tomcat 7.0.57, 8080, "admin/admin"          16
docker.io  docker.io/cloudesire/tomcat       Tomcat server, 6/7/8                        15
```



每一列的含义

列明	含义
NAME	仓库名称
DESCRIPTION	镜像描述
STARS	综合评分，反应一个镜像的受欢迎程度
OFFICIAL	是否官方
AUTOMATED	自动构建，表示该镜像由 Docker Hub 自动构建流程创建的

5.3 拉取镜像

我们拉取镜像默认是从 Docker Hub 镜像仓库上进行获取的,在 Docker Hub 的官网 <https://hub.docker.com/> 上我们可以查看 Docker 镜像的一些信息

Explore Official Repositories

 alpine official	3.8K STARS	10M+ PULLS	> DETAILS
 nginx official	8.9K STARS	10M+ PULLS	> DETAILS
 httpd official	1.8K STARS	10M+ PULLS	> DETAILS
 busybox official	1.3K STARS	10M+ PULLS	> DETAILS
 redis official	5.3K STARS	10M+ PULLS	> DETAILS

国情的原因，国内下载 Docker HUB 官方的相关镜像比较慢，可以使用国内（docker.io）的一些镜像加速器，镜像保持和官方一致，关键是速度块，推荐使用。

常用的镜像加速器有: Mirror 与 Private Registry

区别如下图所示:

序号	Private Registry	Mirror
1	Private Registry（私有仓库）是开发者或者企业自建的镜像存储库，通常用来保存企业内部的 Docker 镜像，用于内部开发流程和产品的发布、版本控制。	Mirror 是一种代理中转服务，我们(比如 daocloud)提供的 Mirror 服务，直接对接 Docker Hub 的官方 Registry。Docker Hub 上有数以十万计的各类 Docker 镜像。
2	在使用 Private Registry 时，需要在 Docker Pull 或 Dockerfile 中直接键入 Private Registry 的地址，通常这样会导致与 Private Registry 的绑定，缺乏灵活性。	使用 Mirror 服务，只需要在 Docker 守护进程（Daemon）的配置文件中加入 Mirror 参数，即可在全局范围内透明的访问官方的 Docker Hub，避免了对 Dockerfile 镜像引用来源的修改。

目前国内访问 docker hub 速度上有点尴尬，使用 docker Mirror 势在必行。现有国内提供 docker 镜像加速服务的商家有不少，下面重点 **ustc** 镜像。

ustc 是老牌的 linux 镜像服务提供者了，还在遥远的 ubuntu 5.04 版本的时候就在用。ustc 的 docker 镜像加速器速度很快。ustc docker mirror 的优势之一就是不需要注册，是真正的公共服务。<https://lug.ustc.edu.cn/wiki/mirrors/help/docker>

配置 ustc 镜像加速器

✚ 编辑文件，注意在 centos7 中不存在 vim 命令，因此我们只能使用 vi 进行编辑

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

✚ 在配置文件中配置如下内容

```
{
  "registry-mirrors": ["https://docker.mirrors.ustc.edu.cn"]
}
```

✚ 重启 docker

```
systemctl restart docker
```

如果重启 docker 以后还是无法进行加速,可以重新 os 在进行尝试

✚ 拉取镜像

命令如下:

```
docker pull 镜像名称
```

拉取 tomcat 镜像

```
docker pull tomcat
```

```
[root@pinyoyougou-docker docker]# docker pull tomcat
Using default tag: latest
Trying to pull repository docker.io/library/tomcat ...
latest: Pulling from docker.io/library/tomcat
ccla78bfd46b: Pull complete
d2c05365ee2a: Pull complete
231cb0e216d3: Pull complete
e8912f9d0ce2: Pull complete
9bafe362f99b: Pull complete
28b0652112a6: Pull complete
da23e1e20eae: Pull complete
9d809d99b239: Pull complete
10c61bb6d245: Pull complete
50578eb745b9: Pull complete
85082b9ab294: Pull complete
d34391598837: Pull complete
Digest: sha256:22263c7c58e5397b29d02f88e39e7f2e18943f5bce6682be66f2298afee75769
Status: Downloaded newer image for docker.io/tomcat:latest
```

5.4 删除镜像

我们可以删除指定的镜像也可以删除所有的镜像

✚ 删除指定的镜像

docker rmi 镜像名称/镜像 ID

🔧 删除所有的镜像

docker rmi `docker images -q`: 删除所有镜像

注: `` 中间的字符可以被当做 linux 的命令进行解析执行

`docker images -q` 获取的所有镜像的 ID

[root@pinyoyougou-docker docker]# docker images -q
61205f6444f9

6 Docker 容器操作

6.1 查看容器

🔧 查看正在运行的容器

docker ps

[root@pinyoyougou-docker webapps]# docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
8e631baf38d2	redis	"docker-entrypoint..."	15 minutes ago	Up 15 minutes	0.0.0.0:6379->6379/tcp	itcast_docker_redis
4182b0c3218b	nginx	"nginx -g 'daemon ...'"	23 minutes ago	Up 23 minutes	0.0.0.0:80->80/tcp	itcast_docker_nginx
bec565f8147d	tomcat	"catalina.sh run"	30 minutes ago	Up 30 minutes	0.0.0.0:8099->8080/tcp	itcast_docker_tomcat

列的介绍

列名称	列含义
CONTAINER ID	容器的 ID
IMAGE	创建容器时所使用的镜像
COMMAND	运行容器中的软件执行的命令
CREATED	容器的创建时间
STATUS	容器的状态: UP 表示运行状态 Exited 表示关闭状态
PORTS	宿主机端口和容器中软件的端口的对应关系
NAMES	容器的名称

🔧 查看所有的容器(包含了正在运行的容器以及之前启动过的容器)

docker ps -a

[root@pinyoyougou-docker webapps]# docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d90a27ace8fa	pinyou_docker_tomcat	"catalina.sh run"	3 minutes ago	Exited (143) 59 seconds ago		pinyou_docker_tomcat
8e631baf38d2	redis	"docker-entrypoint..."	16 minutes ago	Up 16 minutes	0.0.0.0:6379->6379/tcp	itcast_docker_redis
4182b0c3218b	nginx	"nginx -g 'daemon ...'"	23 minutes ago	Up 23 minutes	0.0.0.0:80->80/tcp	itcast_docker_nginx
bec565f8147d	tomcat	"catalina.sh run"	30 minutes ago	Up 30 minutes	0.0.0.0:8099->8080/tcp	itcast_docker_tomcat

🔧 查看最后一次运行的容器

docker ps -l

[root@pinyoyougou-docker webapps]# docker ps -l

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d90a27ace8fa	pinyou_docker_tomcat	"catalina.sh run"	4 minutes ago	Exited (143) About a minute ago		pinyou_docker_tomcat

🔧 查看停止的容器

docker ps -f status=exited

[root@pinyoyougou-docker webapps]# docker ps -f status=exited

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
d90a27ace8fa	pinyou_docker_tomcat	"catalina.sh run"	4 minutes ago	Exited (143) 2 minutes ago		pinyou_docker_tomcat

6.2 创建与启动容器

6.2.1 创建容器参数介绍

创建容器的时候我们需要使用如下命令进行容器的创建

docker run

在创建容器的时候我需要使用一下参数.其中常用到的参数如下:

参数名称	参数含义
-i	运行容器
-t	表示容器启动后会进入其命令行。加入这两个参数后，容器创建就能登录进去。即分配一个伪终端。
-d	在 run 后面加上-d 参数,则会创建一个守护式容器在后台运行（这样创建容器后不会自动登录容器，如果只加-i -t 两个参数，创建后就会自动进去容器）
--name	为创建的容器命名
-v	表示目录映射关系（前者是宿主机目录，后者是映射到宿主机上的目录），可以使用多个-v 做多个目录或文件映射。注意：最好做目录映射，在宿主机上做修改，然后共享到容器上
-p	表示端口映射，前者是宿主机端口，后者是容器内的映射端口。可以使用多个-p 做多个端口映射

我们刚才在介绍参数的时候有一个-d 和-t. -d 表示让容器在后台运行起来, -t 表示创建好容器以后我们就指定进行到容器中,进入到容器中以后我们就可以输入命令和容器进行交互. 既然如此,那么也就是说容器我们可以分为两类: 1. 交互式容器 2. 守护式容器

6.2.2 创建交互式容器

创建一个交互式容器并取名为 mycentos

```
docker run -it --name=mycentos1 centos /bin/bash
```

```
[root@pinyoyougou-docker docker]# docker run -it --name=mycentos1 centos /bin/bash
[root@8916401b0097 /]#
```

使用 ps 命令查看容器的状态

```
[root@pinyoyougou-docker ~]# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS      PORTS      NAMES
8916401b0097   centos    "/bin/bash"             58 seconds ago Up 57 seconds mycentos1
```

使用 exit 推出容器

```
[root@8916401b0097 /]# exit
exit
[root@pinyoyougou-docker docker]#
```

然后在查看容器状态

```
[root@pinyoyougou-docker ~]# docker ps -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS      PORTS      NAMES
8916401b0097   centos    "/bin/bash"             2 minutes ago Exited (0) 42 seconds ago mycentos1
```

我们容器也关闭了,这就是交互式容器的特点: 当我们推出容器以后,容器就关闭了

6.2.3 创建守护式容器

创建一个守护式容器: 如果对于一个需要长期运行的容器来说, 我们可以创建一个守护式容器。命令如下 (**容器名称不能重复**):

```
docker run -di --name=mycentos2 centos:7
```

```
[root@pinyoyougou-docker ~]# docker run -di --name=mycentos2 centos
e3f8859e12bf0473d795e9caf8683047d7e1caa51f686f1b15731a5a2cdeb957
[root@pinyoyougou-docker ~]#
```

我们创建好容器以后,这个容器是以后台的方式运行的,那么我们需要操作容器就需要登录到容器中,可以使用如下命令进行登录

```
docker exec -it container_name (container_id) /bin/bash
```



```
[root@pinyoyougou-docker ~]# docker exec -it e3f8859e12bf /bin/bash
[root@e3f8859e12bf /]# ll
total 0
lrwxrwxrwx. 1 root root 7 May 31 18:02 bin -> usr/bin
drwxr-xr-x. 5 root root 340 Jun 24 06:34 dev
drwxr-xr-x. 1 root root 66 Jun 24 06:34 etc
drwxr-xr-x. 2 root root 6 Apr 11 04:59 home
lrwxrwxrwx. 1 root root 7 May 31 18:02 lib -> usr/lib
lrwxrwxrwx. 1 root root 9 May 31 18:02 lib64 -> usr/lib64
drwxr-xr-x. 2 root root 6 Apr 11 04:59 media
drwxr-xr-x. 2 root root 6 Apr 11 04:59 mnt
drwxr-xr-x. 2 root root 6 Apr 11 04:59 opt
dr-xr-xr-x. 115 root root 0 Jun 24 06:34 proc
dr-xr-x--. 2 root root 114 May 31 18:03 root
drwxr-xr-x. 1 root root 21 Jun 24 06:34 run
lrwxrwxrwx. 1 root root 8 May 31 18:02 sbin -> usr/sbin
drwxr-xr-x. 2 root root 6 Apr 11 04:59 srv
dr-xr-xr-x. 13 root root 0 Jun 24 03:27 sys
drwxrwxrwt. 7 root root 132 May 31 18:03 tmp
drwxr-xr-x. 13 root root 155 May 31 18:02 usr
drwxr-xr-x. 18 root root 238 May 31 18:02 var
```

我们退出以登录的容器,然后在查看容器的状态

```
[root@pinyoyougou-docker ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
e3f8859e12bf        centos             "/bin/bash"        5 minutes ago      Up 5 minutes                mycentos2
8916401b0097        centos             "/bin/bash"        12 minutes ago     Exited (0) 10 minutes ago    mycentos1
```

我们发现容器还是处于运行状态,因此我们可以总结出守护式容器的特点: 即使我们退出容器以后,容器还是处于运行状态

6.3 停止与启动容器

🔧 关闭容器

```
docker stop $CONTAINER_NAME/ID

[root@pinyoyougou-docker ~]# docker stop mycentos2
mycentos2
[root@pinyoyougou-docker ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
e3f8859e12bf        centos             "/bin/bash"        8 minutes ago      Exited (137) 5 seconds ago    mycentos2
8916401b0097        centos             "/bin/bash"        16 minutes ago     Exited (0) 14 minutes ago     mycentos1
[root@pinyoyougou-docker ~]#
```

🔧 启动已经关闭的容器

```
docker start $CONTAINER_NAME/ID

[root@pinyoyougou-docker ~]# docker start mycentos1
mycentos1
[root@pinyoyougou-docker ~]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
e3f8859e12bf        centos             "/bin/bash"        9 minutes ago      Exited (137) About a minute ago    mycentos2
8916401b0097        centos             "/bin/bash"        17 minutes ago     Up 2 seconds                mycentos1
```

6.4 文件拷贝

有的时候我们需要将某一个文件复制到容器中,然后在容器中进行时候,那么我们就需要学习一下如何进行文件的拷贝.

如果我们需要将文件拷贝到容器内可以使用 cp 命令

```
docker cp 宿主机对应的文件/或者目录 容器名称:容器目录

[root@pinyoyougou-docker ~]# docker cp mysql mycentos1:/root/
[root@pinyoyougou-docker ~]# docker exec -it mycentos1 /bin/bash
[root@8916401b0097 /]# ls /root/
anaconda-ks.cfg  mysql
```

我们也可以将容器中的某一个文件或者文件夹拷贝到宿主机上

```
docker cp 容器名称:容器目录 需要拷贝的文件或目录

[root@pinyoyougou-docker ~]# docker cp mycentos1:/a.txt .
[root@pinyoyougou-docker ~]# ll

总用量 4
-rw-----. 1 root root 1244 9月 3 2017 anaconda-ks.cfg
-rw-r--r--. 1 root root 0 6月 24 14:53 a.txt
drwxr-xr-x. 2 root root 105 6月 24 14:48 mysql
```


6.5 目录挂载

我们可以在创建容器的时候，将宿主机的目录与容器内的目录进行映射，这样我们就可以通过修改宿主机某个目录的文件从而去影响容器。

命令的格式为: 创建容器 添加 -v 参数 后边为 宿主机目录:容器目录

```
docker run -di -v /root/myhtml/:/root/myhtml --name=mycentos3 centos
```

```
[root@pinyoyougou-docker ~]# docker run -di -v /root/myhtml/:/root/myhtml --name=mycentos3 centos
42baf06c6214597467838ebdb8bb9b3237cd976e16845f227444f73b3ba6e31e
[root@pinyoyougou-docker ~]#
```

进入容器中查看 root 目录下是否存在一个 myhtml 目录

```
[root@pinyoyougou-docker ~]# docker exec -it mycentos3 /bin/bash
[root@42baf06c6214 /]# ll /root/
total 4
-rw-----. 1 root root 3302 May 31 18:03 anaconda-ks.cfg
drwxr-xr-x. 2 root root    6 Jun 24 06:57 myhtml
[root@42baf06c6214 /]#
```

拷贝文件到宿主机的 myhtml 目录下看看在容器中是否可以看到该文件

```
[root@42baf06c6214 myhtml]# ll
total 4
-rw-r--r--. 1 root root 31 Jun 24 07:02 b.txt
[root@42baf06c6214 myhtml]#
```

6.6 查看容器的 IP 地址

我们可以通过以下命令查看容器运行的各种数据

```
docker inspect mycentos3
```

通过上述命令可以查看 docker 容器中很多的信息,其中就包含了 ip 地址

```
{
  "NetworkSettings": {
    "Bridge": "",
    "SandboxID": "adb9c376a7423d0cb6952139695c63a4401a459ffde8ed224b0408f3effdff7f",
    "HairpinMode": false,
    "LinkLocalIPv6Address": "",
    "LinkLocalIPv6PrefixLen": 0,
    "Ports": {},
    "SandboxKey": "/var/run/docker/netns/adb9c376a742",
    "SecondaryIPAddresses": null,
    "SecondaryIPv6Addresses": null,
    "EndpointID": "aa0520c8895ab8e8edce82b9a22ec1c9860b3dc779a69e931091d9804559fd22",
    "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": "c83458c8b06811c9a98b48182440b3e220c582c89eb83f31a8dfef3bd3161e26",
        "EndpointID": "aa0520c8895ab8e8edce82b9a22ec1c9860b3dc779a69e931091d9804559fd22",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.3",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:03"
      }
    }
  }
}
```

我们也可以使用如下命令过滤 ip 地址

```
docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos3
```

```
[root@pinyoyougou-docker myhtml]# docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos3
172.17.0.3
[root@pinyoyougou-docker myhtml]#
```

6.7 删除容器

🔧 删除指定的容器(只能删除关闭的容器)

```
docker rm $CONTAINER_ID/NAME
```



```
[root@pinyoyougou-docker myhtml]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
42baf06c6214        centos              "/bin/bash"         11 minutes ago      Up 11 minutes                               mycentos3
e3f8859e12bf        centos              "/bin/bash"         35 minutes ago      Exited (137) 26 minutes ago                mycentos2
8916401b0097        centos              "/bin/bash"         42 minutes ago      Up 25 minutes                               mycentos1
[root@pinyoyougou-docker myhtml]# docker rm mycentos1
Error response from daemon: You cannot remove a running container 8916401b0097a835d3ebb5d348c0960d873e79342ed9d94e83072b422c82520a. Stop the container
oval or use -f
[root@pinyoyougou-docker myhtml]# docker rm mycentos2
mycentos2
[root@pinyoyougou-docker myhtml]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
42baf06c6214        centos              "/bin/bash"         11 minutes ago      Up 11 minutes                               mycentos3
8916401b0097        centos              "/bin/bash"         43 minutes ago      Up 26 minutes                               mycentos1
```

删除所有的容器

```
docker rm `docker ps -a -q`
```

```
[root@pinyoyougou-docker myhtml]# docker ps -a -q
42baf06c6214
8916401b0097
[root@pinyoyougou-docker myhtml]# docker rm `docker ps -a -q`
42baf06c6214
8916401b0097
[root@pinyoyougou-docker myhtml]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
[root@pinyoyougou-docker myhtml]#
```

7 部署应用

7.1 MySql 部署

拉取镜像

```
docker pull mysql
```

```
[root@pinyoyougou-docker myhtml]# docker pull mysql
Using default tag: latest
Trying to pull repository docker.io/library/mysql ...
latest: Pulling from docker.io/library/mysql
f2aa67a397c4: Pull complete
1accf44cb7e0: Pull complete
2d830ea9fa68: Pull complete
740584693b89: Pull complete
4d620357ec48: Pull complete
ac3b7158d73d: Pull complete
a48d784ee503: Pull complete
f122eadb2640: Pull complete
3df40c552a96: Pull complete
da7d77a8ed28: Pull complete
f03c5af3b206: Pull complete
54dd1949fa0f: Pull complete
Digest: sha256:d60c13a2bfdbbeb9cf1c84fd3cb0a1577b2bbaeec11e44bf345f4da90586e9e1
Status: Downloaded newer image for docker.io/mysql:latest
```

查看拉取下拉的镜像

```
[root@pinyoyougou-docker myhtml]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker.io/tomcat     latest             61205f6444f9       2 weeks ago        467 MB
docker.io/centos     latest             49f7960eb7e4       2 weeks ago        200 MB
docker.io/mysql      latest             a8a59477268d       7 weeks ago        445 MB
```

通过镜像创建容器

```
docker run -di --name=itcast_docker_mysql -p 33306:3306 -e MYSQL_ROOT_PASSWORD=1234 mysql
```

```
[root@pinyoyougou-docker myhtml]# docker run -di --name=itcast_docker_mysql -p 33306:3306 -e MYSQL_ROOT_PASSWORD=1234 mysql
4db98fd23fd8d42e6a7c27d3f3a87775c751d6bea03dab650926c37150c02376
[root@pinyoyougou-docker myhtml]# docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS              NAMES
4db98fd23fd8        mysql              "docker-entrypoint.." 7 seconds ago      Up 6 seconds       0.0.0.0:33306->3306/tcp  itcast_docker_mysql
[root@pinyoyougou-docker myhtml]#
```

-p 代表端口映射，格式为 宿主机映射端口:容器运行端口
-e 代表添加环境变量 MYSQL_ROOT_PASSWORD 是 root 用户的登陆密码

登录容器

```
docker exec -it itcast_docker_mysql /bin/bash
```

```
[root@pinyougou-docker myhtml]# docker exec -it itcast_docker_mysql /bin/bash
root@4db98fd23fd8:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.11 MySQL Community Server - GPL

Copyright (c) 2000, 2018, 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 (0.04 sec)
```

同样我们也可以使用 mysql 的客户端软件去链接 mysql

注意 docker 中安装的最新的 mysql 版本是 8.0 版本

```
mysql> select version();
+-----+
| version() |
+-----+
| 8.0.11 |
+-----+
1 row in set (0.00 sec)
```

MySQL8.0 版本 和 5.0 的加密规则不一样，而现在的可视化工具只支持旧的加密方式。因此我们需要修改 MySQL 用户登录的加密规则修改为 mysql_native_password

修改加密规则

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'password' PASSWORD EXPIRE NEVER;
```

修改密码

```
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'password';
```

开启远程访问权限

```
GRANT ALL ON *.* TO 'root'@'%';
```

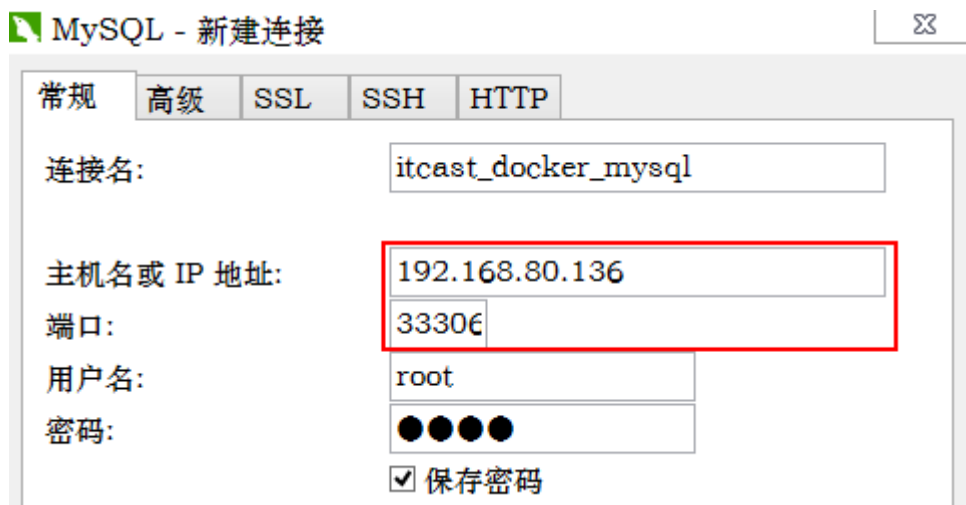
刷新权限

```
flush privileges;
```

有必要时关闭 linux 防火墙

```
systemctl stop firewalld
```

通过远程工具链接



注：如果我想使用 mysql5.x 的版本,那么在进行镜像下载的时候可以指定 mysql 的版本
具体的版本信息可以在 docker hub 上进行查看

Full Description

Supported tags and respective Dockerfile links

- 8.0.11, 8.0, 8, latest (8.0/Dockerfile)
- 5.7.22, 5.7, 5 (5.7/Dockerfile)
- 5.6.40, 5.6 (5.6/Dockerfile)
- 5.5.60, 5.5 (5.5/Dockerfile)

```
[root@pinyoyougou-docker ~]# docker pull mysql:5.6
Trying to pull repository docker.io/library/mysql ...
5.6: Pulling from docker.io/library/mysql
f2aa67a397c4: Already exists
1accf44cb7e0: Already exists
2d830ea9fa68: Already exists
740584693b89: Already exists
```

7.2 Tomcat 部署

拉取镜像

```
docker pull tomcat
```

创建容器

```
docker run -di --name=itcast_docker_tomcat -p 8099:8080 -v /usr/local/webapps/:/usr/local/tomcat/webapps tomcat
```

-p: 指定端口映射
-v: 指定目录映射

部署应用

将我们的应用程序部署到宿主机的/usr/local/webapps/目录下，然后进行访问

如果我们的应用向链接 docker 容器中的 mysql,那么就需要查询 docker 的 mysql 容器的 ip 地址,然后将应用的数据库连接信息做响应的修改

7.3 Nginx 部署

拉取镜像

```
docker pull nginx
```

```
[root@pinyoyougou-docker webapps]# docker pull nginx
Using default tag: latest
Trying to pull repository docker.io/library/nginx ...
latest: Pulling from docker.io/library/nginx
f2aa67a397c4: Already exists
1cd0975d4f45: Pull complete
72fd2d3be09a: Pull complete
Digest: sha256:3e2ffcf0edca2a4e9b24ca442d227baea7b7f0e33ad654ef1eb806fbd9bedcf0
Status: Downloaded newer image for docker.io/nginx:latest
```

创建容器

```
docker run -di --name=itcast_docker_nginx -p 80:80 nginx
```

```
[root@pinyoyougou-docker webapps]# docker run -di --name=itcast_docker_nginx -p 80:80 nginx
4182b0c3218be35337bc9f3055e3ca07135c320d61b0d86dc694d8dcb7bcc828
```

访问容器 <http://192.168.80.136/>

注: 容器中的 nginx 的配置文件存储于/etc/nginx 目录下,所以要进行反向代理以及负载均衡的配置可以上/etc/nginx 目录下找到对应的配置文件然后做更改. 如果在容器中没有办法进行修改,那么我们可以将其容器中的配置文件拷贝到宿主主机上进行修改,然后在将修改后的文件拷贝到容器中

7.4 Redis 部署

拉取镜像

```
docker pull redis
```



```
[root@pinyougou-docker webapps]# docker pull redis
Using default tag: latest
Trying to pull repository docker.io/library/redis ...
latest: Pulling from docker.io/library/redis
f2aa67a397c4: Already exists
298c0b953ff5: Pull complete
54481933a13d: Pull complete
e236faec6ff6: Pull complete
93540029cb59: Pull complete
1c88aa70d0e2: Pull complete
Digest: sha256:c389a35832492c42f4719776471f9a42d2fc5a8ba0077ba25746626b09eab880
Status: Downloaded newer image for docker.io/redis:latest
```

创建容器

```
docker run -di --name=itcast_docker_redis -p 6379:6379 redis
```

```
[root@pinyougou-docker webapps]# docker run -di --name=itcast_docker_redis -p 6379:6379 redis
8e631baf38d26c703f7deb78c3db6f10bf42185f55abe944d31fdf62647f6a60
[root@pinyougou-docker webapps]#
```

通过客户端连接容器

```
D:\redis2.8win32>redis-cli.exe -h 192.168.80.136 -p 6379
192.168.80.136:6379> set a zhansgan
OK
192.168.80.136:6379> get a
"zhansgan"
192.168.80.136:6379>
```

8 Docker 备份与迁移

8.1 容器保存为镜像

我们可以通过以下命令将容器保存为镜像

```
docker commit 容器名称 保存的新镜像的名称
docker commit itcast_docker_tomcat pinyou_docker_tomcat
```

```
[root@pinyougou-docker webapps]# docker commit itcast_docker_tomcat pinyou_docker_tomcat
sha256:e206ecac6dc9c5bb216854a8f1aceb233ae47edce990fb2273cb8b7d3deba010
[root@pinyougou-docker webapps]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
pinyou_docker_tomcat latest             e206ecac6dc9       28 seconds ago     467 MB
docker.io/redis      latest             55cb7014c24f       3 days ago         83.4 MB
docker.io/tomcat     latest             61205f6444f9       2 weeks ago        467 MB
docker.io/nginx      latest             cd5239a0906a       2 weeks ago        109 MB
docker.io/centos     latest             49f7960eb7e4       2 weeks ago        200 MB
docker.io/mysql      5.6                5f5ccdc8aedc       7 weeks ago        256 MB
docker.io/mysql      latest             a8a59477268d       7 weeks ago        445 MB
```

8.2 镜像备份

当我们把容器保存为镜像了以后,那么接下来我们就需要将镜像打包成一个文件

```
docker save -o 打包以后的文件名称 镜像名称
docker save -o pinyougou_docker_tomcat.tar pinyou_docker_tomcat
```

-o: 表示的意思是输出

```
[root@pinyougou-docker webapps]# docker save -o pinyougou_docker_tomcat.tar pinyou_docker_tomcat
[root@pinyougou-docker webapps]# ls
docker  pinyougou_docker_tomcat.tar
```

8.3 镜像恢复与迁移

首先我们先删除掉 pinyou_docker_tomcat 镜像

```
[root@pinyougou-docker webapps]# docker rmi pinyou_docker_tomcat
Untagged: pinyou_docker_tomcat:latest
Deleted: sha256:e206ecac6dc9c5bb216854a8f1aceb233ae47edce990fb2273cb8b7d3deba010
Deleted: sha256:10e51cc236c57d273759ce9a43caac45dc4b453b62b3074f6427528afb1ba0a1
[root@pinyougou-docker webapps]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker.io/redis      latest             55cb7014c24f       3 days ago         83.4 MB
docker.io/tomcat     latest             61205f6444f9       2 weeks ago        467 MB
docker.io/nginx      latest             cd5239a0906a       2 weeks ago        109 MB
docker.io/centos     latest             49f7960eb7e4       2 weeks ago        200 MB
docker.io/mysql      5.6                5f5ccdc8aedc       7 weeks ago        256 MB
docker.io/mysql      latest             a8a59477268d       7 weeks ago        445 MB
```

然后执行此命令进行恢复

```
docker load -i pinyougou_docker_tomcat.tar
```

-i: 表示输入