容器部署解决方案 Docker

课程目标

目标 1：了解 Docker 与虚拟机的不同点，相比的优势

目标 2：掌握 Docker 的启动方法

目标 3：掌握 Docker 镜像操作

目标 4：掌握 Docker 容器操作

目标 5：掌握 Mysql tomcat Nginx Redis 等容器的部署方法

目标 6：掌握 Docker 的备份与迁移方法

1.Docker 简介

1.1 虚拟化

1.1.1 什么是虚拟化

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。

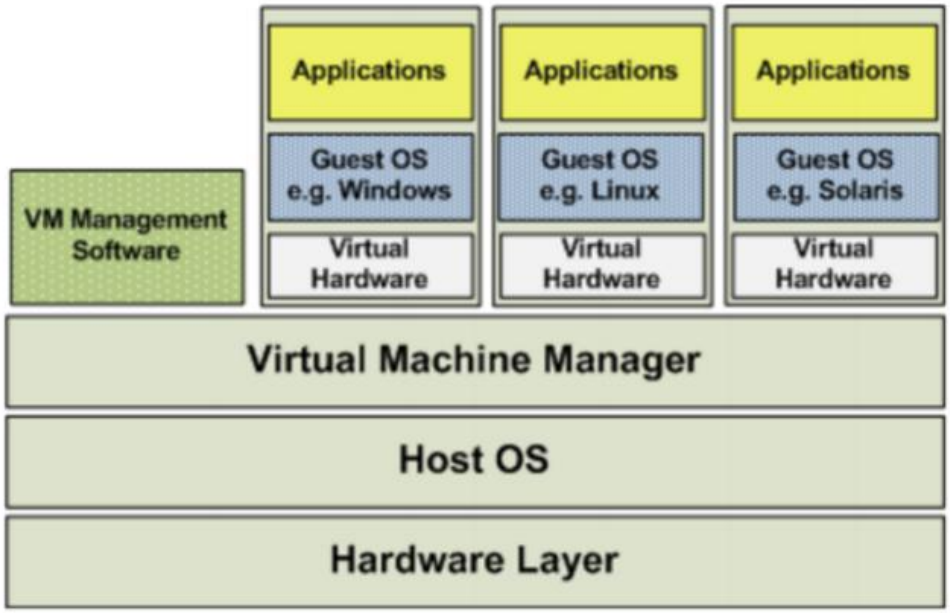
在实际的生产环境中，虚拟化技术主要用来解决高性能的物理硬件产能过剩和老的旧的硬件产能过低的重组重用，透明化底层物理硬件，从而最大化的利用物理硬件　对资源充分利用

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。

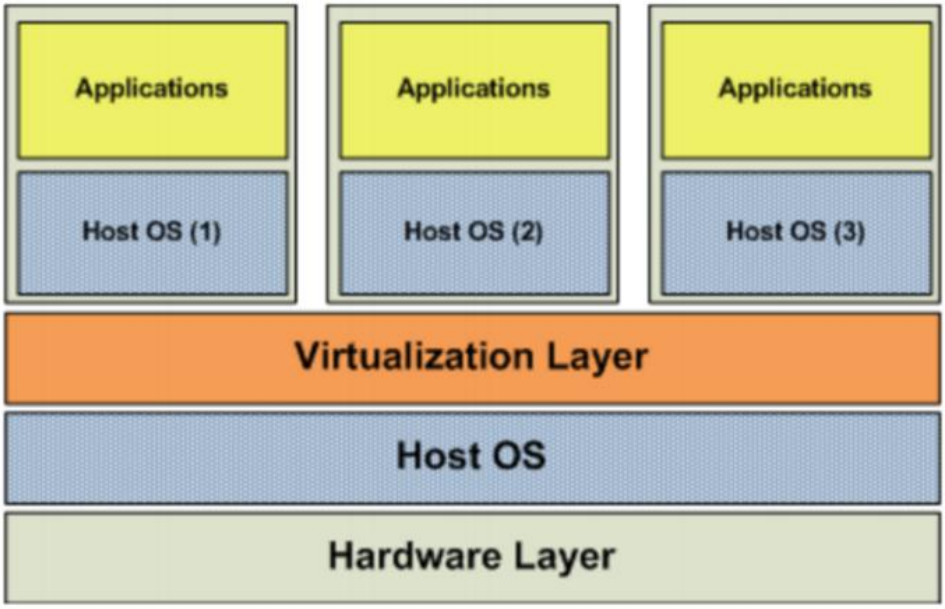
1.1.2 虚拟化种类

（1）全虚拟化架构

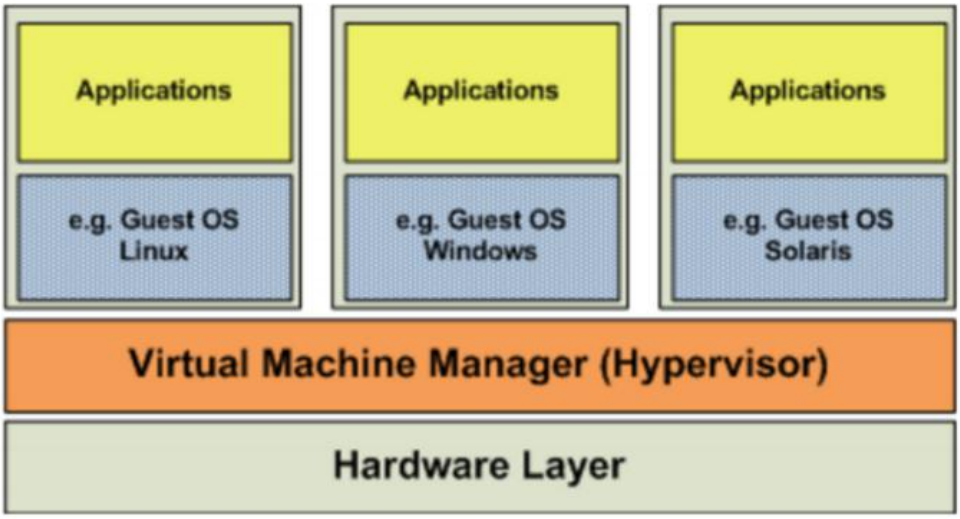
虚拟机的监视器（hypervisor）是类似于用户的应用程序运行在主机的 OS 之上，如 VMware 的 workstation，这种虚拟化产品提供了虚拟的硬件。



（2）OS 层虚拟化架构



（3）硬件层虚拟化



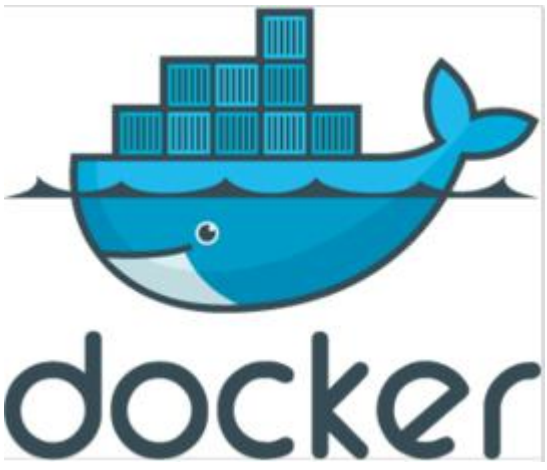
硬件层的虚拟化具有高性能和隔离性，因为 hypervisor 直接在硬件上运行，有利于控制 VM 的 OS 访问硬件资源，使用这种解决方案的产品有 VMware ESXi 和 Xen server

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、网络和磁盘，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

1.2 什么是 Docker



1.2.1 容器技术

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。

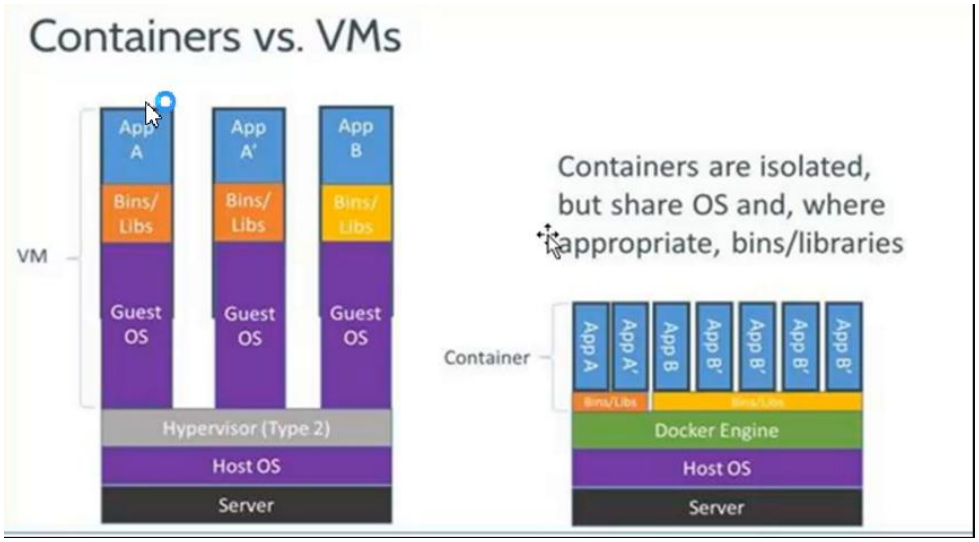
最新的容器技术引入了 OpenVZ、Solaris Zones 以及 Linux 容器（LXC）。使用这些新技术，容器不在仅仅是一个单纯的运行环境。在自己的权限类内，容器更像是一个完整的宿主机。对 Docker 来说，它得益于现代 Linux 特性，如控件组（control group）、命名空间（namespace）技术，容器和宿主机之间的隔离更加彻底，容器有独立的网络和存储栈，还拥有自己的资源管理能力，使得同一台宿主机中的多个容器可以友好的共存。

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。

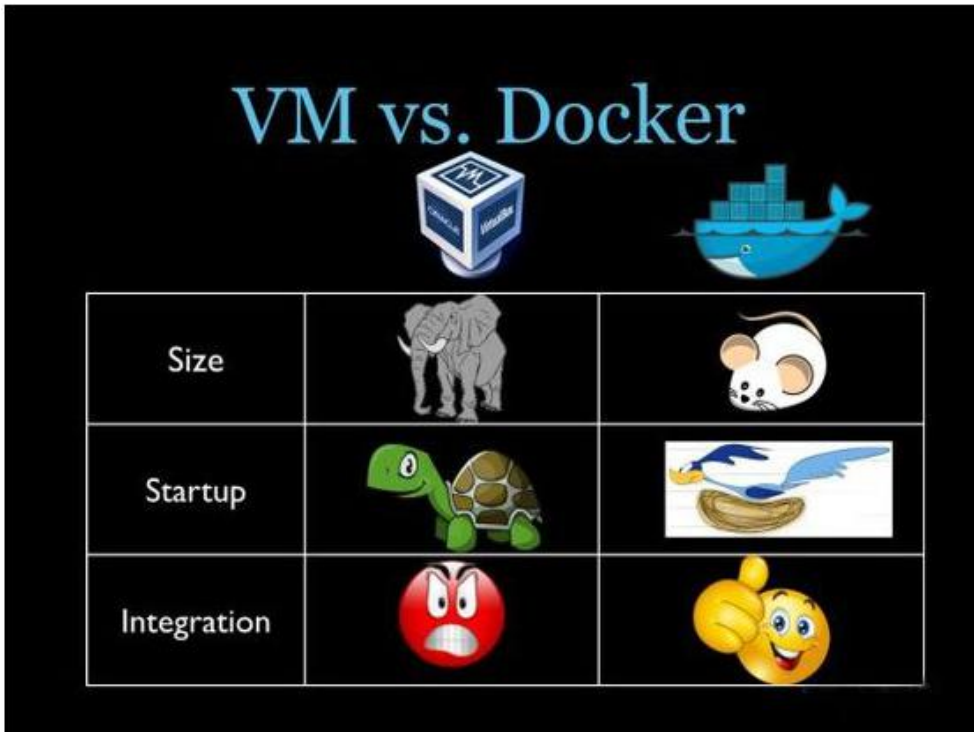
尽管有着光辉的历史，容器仍未得到广泛的认可。一个很重要的原因就是容器技术的复杂性：容器本身就比较复杂，不易安装，管理和自动化也很困难。而 Docker 就是为了改变这一切而生的。

1.2.2 容器与虚拟机比较

（1）本质上的区别



（2）使用上的区别



虚拟机已死，容器才是未来。

1.2.3 Docker 特点

（1）上手快。

用户只需要几分钟，就可以把自己的程序“Docker 化”。Docker 依赖于“写时复制”（copy-on-write）模型，使修改应用程序也非常迅速，可以说达到“随心所致，代码即改”的境界。

随后，就可以创建容器来运行应用程序了。大多数 Docker 容器只需要不到 1 秒中即可启动。由于去除了管理程序的开销，Docker 容器拥有很高的性能，同时同一台宿主机中也可以运行更多的容器，使用户尽可能的充分利用系统资源。

（2）职责的逻辑分类

使用 Docker，开发人员只需要关心容器中运行的应用程序，而运维人员只需要关心如何管理容器。Docker 设计的目的就是要加强开发人员写代码的开发环境与应用程序要部署的生产环境一致性。从而降低那种“开发时一切正常，肯定是运维的问题（测试环境都是正常的，上线后出了问题就归结为肯定是运维的问题）”

（3）快速高效的开发生命周期

Docker 的目标之一就是缩短代码从开发、测试到部署、上线运行的周期，让你的应用程序具备可移植性，易于构建，并易于协作。（通俗一点说，Docker 就像一个盒子，里面可以装很多物件，如果需要这些物件的可以直接将该大盒子拿走，而不需要从该盒子中一件件的取。）

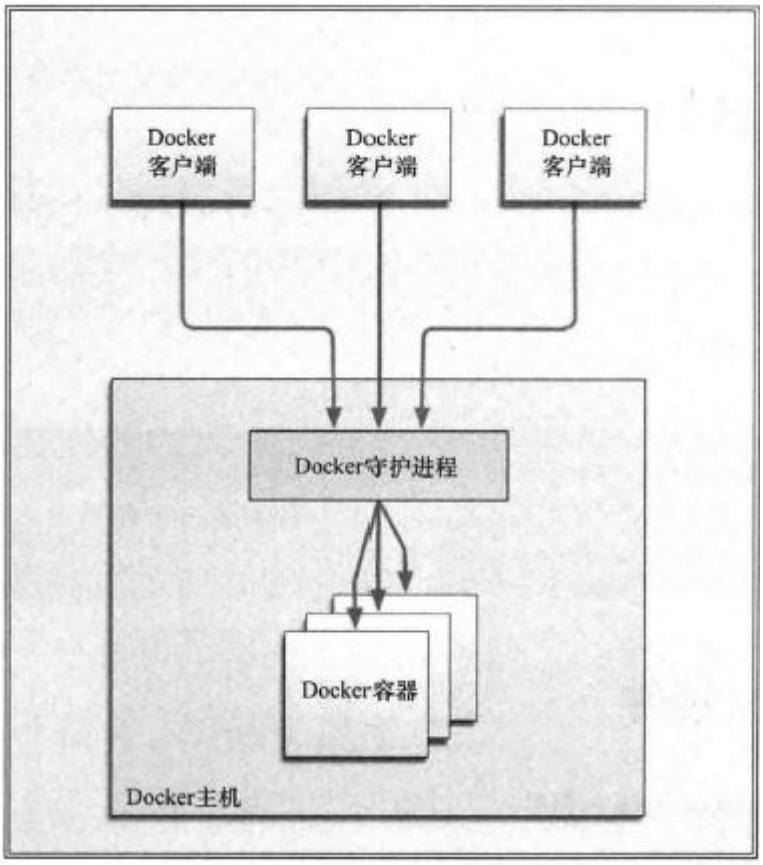
（4）鼓励使用面向服务的架构

Docker 还鼓励面向服务的体系结构和微服务架构。Docker 推荐单个容器只运行一个应用程序或进程，这样就形成了一个分布式的应用程序模型，在这种模型下，应用程序或者服务都可以表示为一系列内部互联的容器，从而使分布式部署应用程序，扩展或调试应用程序都变得非常简单，同时也提高了程序的内省性。（当然，可以在一个容器中运行多个应用程序）

1.3 Docker 组件

1.3.1 Docker 客户端和服务器

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。



1.3.2 Docker 镜像

镜像是构建 Docker 的基石。用户基于镜像来运行自己的容器。镜像也是 Docker 生命周期中的“构建”部分。镜像是基于联合文件系统的一种层式结构，由一系列指令一步一步构建出来。例如：

添加一个文件；

执行一个命令；

打开一个窗口。

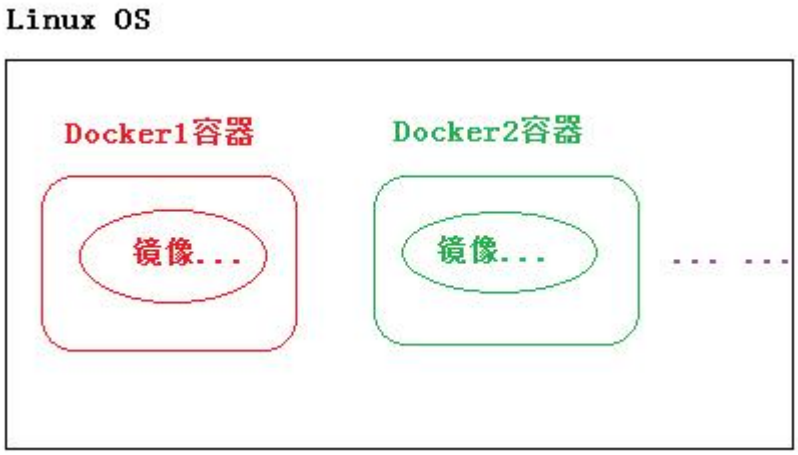
也可以将镜像当作容器的“源代码”。镜像体积很小，非常“便携”，易于分享、存储和更新。

1.3.3 Registry（注册中心）

Docker 用 Registry 来保存用户构建的镜像。Registry 分为公共和私有两种。Docker 公司运营公共的 Registry 叫做 Docker Hub。用户可以在 Docker Hub 注册账号，分享并保存自己的镜像（说明：在 Docker Hub 下载镜像巨慢，可以自己构建私有的 Registry）。

1.3.4 Docker 容器

Docker 可以帮助你构建和部署容器，你只需要把自己的应用程序或者服务打包放进容器即可。容器是基于镜像启动起来的，容器中可以运行一个或多个进程。我们可以认为，镜像是 Docker 生命周期中的构建或者打包阶段，而容器则是启动或者执行阶段。 容器基于镜像启动，一旦容器启动完成后，我们就可以登录到容器中安装自己需要的软件或者服务。



所以 Docker 容器就是：

一个镜像格式；

一些列标准操作；

一个执行环境。

Docker 借鉴了标准集装箱的概念。标准集装箱将货物运往世界各地，Docker 将这个模型运用到自己的设计中，唯一不同的是：集装箱运输货物，而 Docker 运输软件。

和集装箱一样，Docker 在执行上述操作时，并不关心容器中到底装了什么，它不管是 web 服务器，还是数据库，或者是应用程序服务器什么的。所有的容器都按照相同的方式将内容“装载”进去。

Docker 也不关心你要把容器运到何方：我们可以在自己的笔记本中构建容器，上传到 Registry，然后下载到一个物理的或者虚拟的服务器来测试，在把容器部署到具体的主机中。像标准集装箱一样，Docker 容器方便替换，可以叠加，易于分发，并且尽量通用。

使用 Docker，我们可以快速的构建一个应用程序服务器、一个消息总线、一套实用工具、一个持续集成（CI）测试环境或者任意一种应用程序、服务或工具。我们可以在本地构建一个完整的测试环境，也可以为生产或开发快速复制一套复杂的应用程序栈。

2.Docker 安装与启动

2.1 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。

2.2 在 VMware Workstation 中安装 CentOS7

资料已经准备了安装好的镜像，直接挂载即可。

挂载后，使用 ip addr 命令查看本地 IP

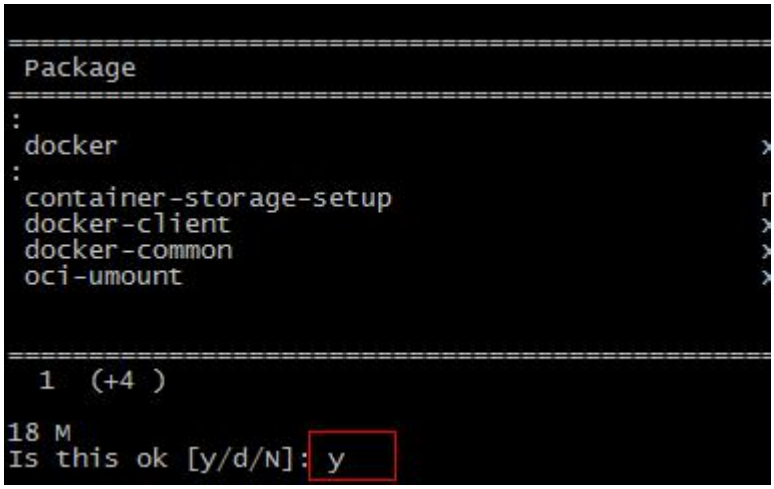
```
[root@pinyoyougou-docker ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:ed:85:fe brd ff:ff:ff:ff:ff:ff
    inet 192.168.247.135/24 brd 192.168.247.255 scope global dynamic ens33
       valid_lft 1770sec preferred_lft 1770sec
    inet6 fe80::64ba:dea0:c4c3:6593/64 scope link
       valid_lft forever preferred_lft forever
```

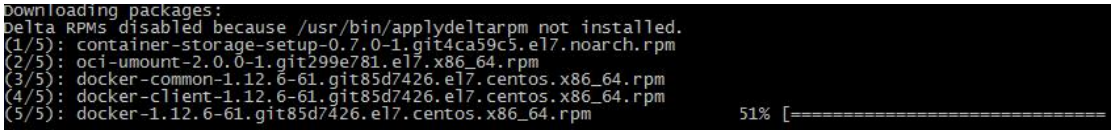
2.3 安装 Docker

使用 yum 命令在线安装----------01

```
yum install docker
```

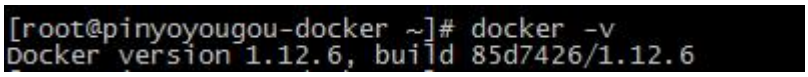
```
updates
(1/2): extras/7/x86_64/primary_db
(2/2): updates/7/x86_64/primary_db
Determining fastest mirrors
 * base: mirrors.tuna.tsinghua.edu.cn
 * extras: mirrors.tuna.tsinghua.edu.cn
 * updates: mirrors.tuna.tsinghua.edu.cn

-->
--->  docker.x86_64.2.1.12.6-55.gitc4618fb.el7.centos
--->  docker.x86_64.2.1.12.6-61.git85d7426.el7.centos
-->  docker-common = 2:1.12.6-61.git85d7426.el7.centos 2:docker-1.12.6-61.git8
-->  docker-client = 2:1.12.6-61.git85d7426.el7.centos 2:docker-1.12.6-61.git8
-->
```





2.4 安装后查看 Docker 版本

```
docker -v
```



2.5 启动与停止 Docker

这几个命令都要手敲上去为好~~~

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

- 启动 docker：systemctl start docker
- 停止 docker：systemctl stop docker
- 重启 docker：systemctl restart docker
- 查看 docker 状态：systemctl status docker
- 开机启动：systemctl enable docker

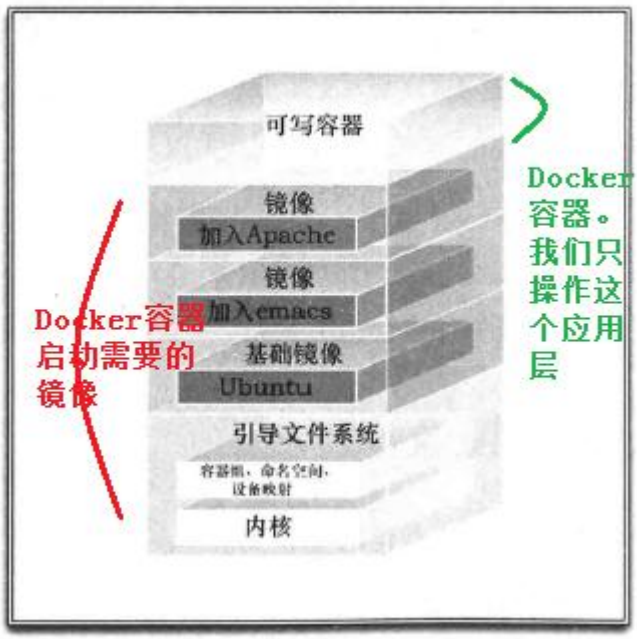
```
Hint: Some lines were ellipsized, use -l to show in full.
[root@localhost ~]# systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2017-04-10 18:19:47 PDT; 1h 7min ago
     Docs: http://docs.docker.com
 Main PID: 802 (dockerd-current)
   CGroup: /system.slice/docker.service
           ├─802 /usr/bin/dockerd-current --add-runtime docker-runc=/usr/libexec/docker/dock
           └─1972 /usr/bin/docker-containerd-current -l unix:///var/run/docker/libcontainerd/
```

- 查看 docker 概要信息：docker info
- 查看 docker 帮助文档：docker --help

3.Docker 镜像操作

3.1 什么是 Docker 镜像

Docker 镜像是由文件系统叠加而成（是一种文件的存储形式）。最底端是一个文件引导系统，即 bootfs，这很像典型的 Linux/Unix 的引导文件系统。Docker 用户几乎永远不会和引导系统有什么交互。实际上，当一个容器启动后，它将会被移动到内存中，而引导文件系统则会被卸载，以留出更多的内存供磁盘镜像使用。Docker 容器启动是需要的一些文件，而这些文件就可以称为 Docker 镜像。



3.2 列出镜像

列出 docker 下的所有镜像：docker images

```
See 'docker --help'.
[root@localhost ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
docker.io/solr      latest              a1012bc9e0af        16 months ago       562.7 MB
docker.io/nginx     latest              b24cded722db        16 months ago       132.7 MB
docker.io/mysql     latest              285498727bad        16 months ago       359.8 MB
docker.io/tomcat    latest              02cb33d2c49b        16 months ago       356.9 MB
docker.io/redis     latest              2675409c51d4        16 months ago       109.1 MB
docker.io/centos    latest              3635e344e8a0        17 months ago       172.3 MB
[root@localhost ~]#
```

- REPOSITORY：镜像所在的仓库名称
- TAG：镜像标签
- IMAGE ID：镜像 ID
- CREATED：镜像的创建日期（不是获取该镜像的日期）
- SIZE：镜像大小
- 这些镜像都是存储在 Docker 宿主机的/var/lib/docker 目录下

```
calhost ~]# cd /var/lib/docker/
calhost docker]# ll

--. 3 root root   78 Apr 10 19:41 containers
--. 5 root root   53 Mar  9 00:28 devicemapper
--. 3 root root   26 Feb 16 00:03 image
--. 3 root root   19 Feb 16 00:03 network
--. 2 root root    6 Feb 16 00:03 swarm
--. 2 root root    6 Mar  9 00:33 tmp
--. 2 root root    6 Feb 16 00:03 trust
--. 9 root root 4096 Mar 27 23:29 volumes
calhost docker]# cd containers/
calhost containers]# ll

--. 4 root root 149 Apr 10 19:41 e9f13b711034fb35ba1bbc7a3c13724d4809ca2936d5322de015fefd51621469
```

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，例如 12.10、12.04 等等。每个标签对组成特定镜像的一些镜像层进行标记（比如，标签 12.04 就是对所有 Ubuntu12.04 镜像层的标记）。这种机制使得同一个仓库中可以存储多个镜像。--- 版本号

我们在运行同一个仓库中的不同镜像时，可以通过在仓库名后面加上一个冒号和标签名来指定该仓库中的某一具体的镜像，例如 docker run --name custom_container_name –i –t docker.io/ubunto:12.04 /bin/bash，表明从镜像 Ubuntu:12.04 启动一个容器，而这个镜像的操作系统就是 Ubuntu:12.04。在构建容器时指定仓库的标签也是一个好习惯。

3.3 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
```

```
172.17.0.2
[root@localhost ~]# docker search tomcat
INDEX       NAME                                      DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
docker.io   docker.io/tomcat                          Apache Tomcat is an open source implementa...   1290      [OK]
docker.io   docker.io/dordoka/tomcat                  Ubuntu 14.04, Oracle JDK 8 and Tomcat 8 ba...   33                   [OK]
docker.io   docker.io/davidcaste/alpine-tomcat        Apache Tomcat 7/8 using Oracle Java 7/8 wi...   17                   [OK]
docker.io   docker.io/cloudesire/tomcat               Tomcat server, 6/7/8                            14                   [OK]
docker.io   docker.io/andreptb/tomcat                 Debian Jessie based image with Apache Tomc...   6                    [OK]
docker.io   docker.io/openweb/oracle-tomcat           A fork off of official tomcat image with O...   5                    [OK]
docker.io   docker.io/fbrx/tomcat                     Minimal Tomcat image based on Alpine Linux      4                    [OK]
docker.io   docker.io/abzcoding/tomcat-redis          a tomcat container with redis as session m...   2                    [OK]
docker.io   docker.io/kieker/tomcat                                                                   2                    [OK]
docker.io   docker.io/antoineco/tomcat-mod_cluster    Apache Tomcat with JBoss mod_cluster            1                    [OK]
docker.io   docker.io/bitnami/tomcat                  Bitnami Tomcat Docker Image                     1                    [OK]
docker.io   docker.io/camptocamp/tomcat-logback       Docker image for tomcat with logback integ...   1                    [OK]
docker.io   docker.io/cloudunit/tomcat-9              tomcat-9                                        1                    [OK]
docker.io   docker.io/picoded/tomcat                  tomcat 8 with java 8, and MANAGER_USER / M...   1                    [OK]
docker.io   docker.io/antoineco/tomcat                Extra OS variants for the official Tomcat ...   0                    [OK]
docker.io   docker.io/blueur/tomcat                   Tomcat with the possibility to set the use...   0                    [OK]
docker.io   docker.io/charlycoste/tomcat             Debian based Tomcat (don't use it, this is...   0                    [OK]
docker.io   docker.io/cheewai/tomcat                  Tomcat and Oracle JRE in docker                 0                    [OK]
docker.io   docker.io/dianplus/tomcat                 Tomcat base image maintained by dianjia.io.     0                    [OK]
docker.io   docker.io/inspectit/tomcat               Tomcat with inspectIT                           0                    [OK]
docker.io   docker.io/maxird/tomcat                  Tomcat                                          0                    [OK]
docker.io   docker.io/muicoder/tomcat                Tomcat 6/7/8/9                                  0                    [OK]
docker.io   docker.io/oriaks/tomcat                  Tomcat                                          0                    [OK]
docker.io   docker.io/phpmentors/tomcat-app          Tomcat application image using Maven            0                    [OK]
docker.io   docker.io/steigr/tomcat                  Apache Tomcat based on Alpine Linux with J...   0                    [OK]
[root@localhost ~]#
```

- NAME：仓库名称
- DESCRIPTION：镜像描述
- STARS：用户评价，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

3.4 拉取镜像

3.4.1 从 Docker Hub 拉取

Docker 镜像首页，包括官方镜像和其它公开镜像。Docker Hub 上最受欢迎的 10 大镜像（通过 Docker registry API 获取不了镜像被 pull 的个数，只能通过镜像的 stars 数量，来衡量镜像的流行度。毫无疑问，拥有最高 stars 数量的库都是官方库）。

Rank	Repository	Stars
1	Ubuntu	2,007
2	Centos	1,164
3	Nginx	1,163
4	Redis	957
5	Node	891
6	Postgres	889
7	Mysql	885
8	Mongo	796
9	Debian	573
10	Jenkins	508

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。Mirror 与 Private Registry

的区别：

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。

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。

简单来说，Mirror 类似 CDN，本质是官方的 cache；Private Registry 类似私服，跟官方没什么关系。对用户来说，由于用户是要拖 docker hub 上的 image，对应的是 Mirror。yum/apt-get 的 Mirror 又有点不一样，它其实是把官方的库文件整个拖到自己的服务器上做镜像，并定时与官方做同步；而 Docker Mirror 只会缓存曾经使用过的 image。

使用命令拉取：

```
docker pull centos:7
```

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。

3.4.2 ustc 的镜像

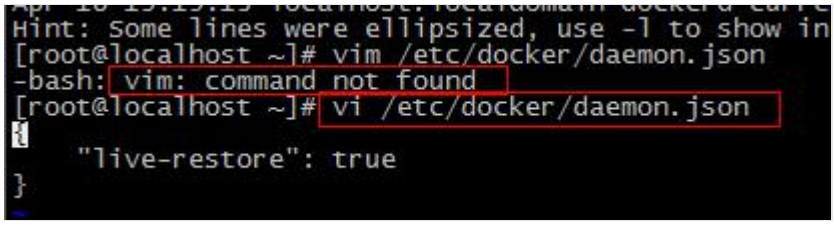
ustc 是老牌的 linux 镜像服务提供者了，还在遥远的 ubuntu 5.04 版本的时候就在用。ustc 的 docker 镜像加速器速度很快。ustc docker mirror 的优势之一就是不需要注册，是真正的公共服务。

https://lug.ustc.edu.cn/wiki/mirrors/help/docker

步骤：

（1）编辑该文件：vi /etc/docker/daemon.json // 如果该文件不存在就手动创建；说明：在 centos7.x 下，通过 vi。

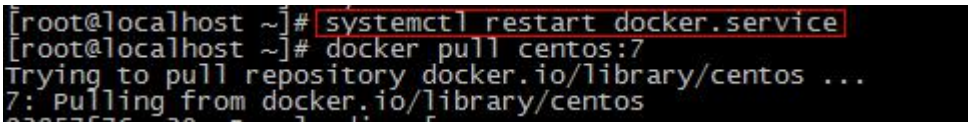
-- 创建文件 touch deamon.json



（2）在该文件中输入如下内容：

```
{

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

}
```

（3）注意：一定要重启 docker 服务，如果重启 docker 后无法加速，可以重新启动 OS



然后通过 docker pull 命令下载镜像：速度杠杠的。

3.5 删除镜像

1、 docker rmi $IMAGE_ID：删除指定镜像

2、 docker rmi `docker images -q`：删除所有镜像

4.Docker 容器操作

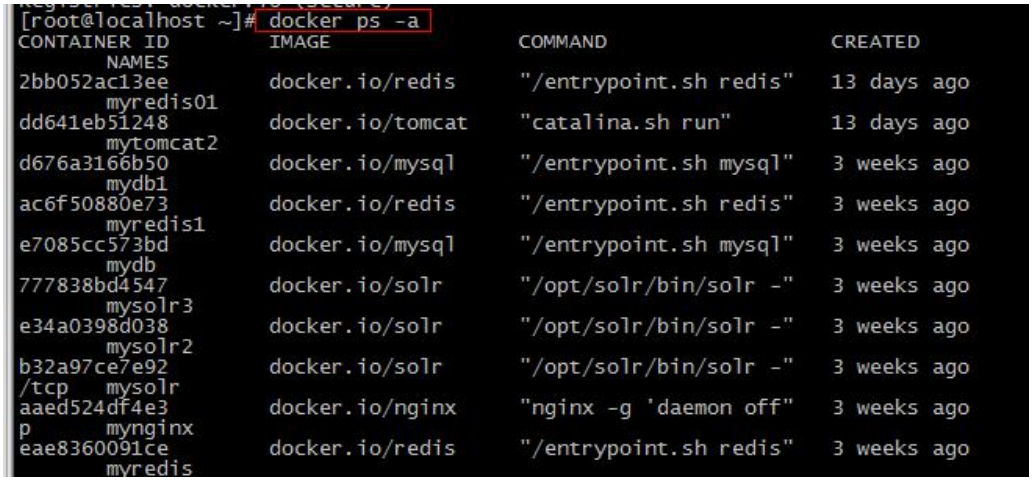
4.1 查看容器

- 查看正在运行容器：

```
docker ps
```

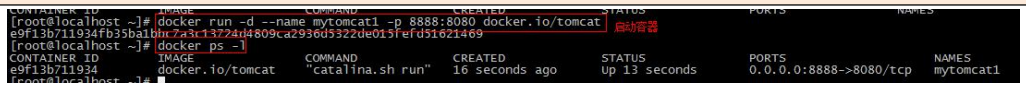
- 查看所有的容器（启动过的历史容器）：

```
docker ps –a
```



- 查看最后一次运行的容器：

```
docker ps –l
```



- 查看停止的容器

```
docker ps -f status=exited
```

4.2 创建与启动容器

```

```

- 创建容器常用的参数说明：

- 创建容器命令：docker run
- -i：表示运行容器
- -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。
- --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。
- -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）。
- -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

4.1.1 交互式容器

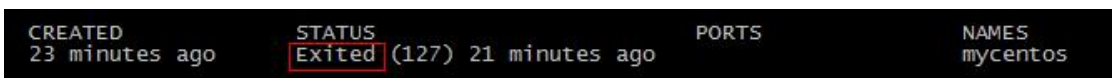
创建一个交互式容器并取名为 mycentos

```
docker run -it --name=mycentos centos:7 /bin/bash
```

这时我们通过 ps 命令查看，发现可以看到启动的容器，状态为启动状态

使用 exit 命令 退出当前容器

然后用 ps -a 命令查看发现该容器也随之停止：



4.1.2 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
```

- 登录守护式容器方式：

docker exec -it container_name (或者 container_id) /bin/bash（exit 退出时，容器不会停止）

```
[root@pinyoyougou-docker ~]# docker exec -it  mycentos4 /bin/bash
[root@704db36e6871 /]#
```

4.3 停止与启动容器

- 停止正在运行的容器：docker stop $CONTAINER_NAME/ID

```
[root@localhost ~]# docker stop mytomcat1
mytomcat1
```

- 启动已运行过的容器：docker start $CONTAINER_NAME/ID

```
563a4d9f8894        docker.io/tomcat    "ca
[root@localhost ~]# docker start mytomcat1
mytomcat1
[root@localhost ~]#
```

4.4 文件拷贝

如果我们需要将文件拷贝到容器内可以使用 cp 命令

```
docker cp  需要拷贝的文件或目录  容器名称:容器目录
```

也可以将文件从容器内拷贝出来

```
docker cp  容器名称:容器目录  需要拷贝的文件或目录
```

4.5 目录挂载

```
----    创建守护式容器  命令

    docker  run  -di  -v/usr/local/apps/centos:/usr/local/apps/centos  --name=a_centos  centos:7
/bin/bash
```

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。

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 /usr/local/myhtml:/usr/local/myhtml --name=mycentos2 centos:7
```

如果你共享的是多级的目录，可能会出现权限不足的提示。

```
Permission denied
```

这是因为 CentOS7 中的安全模块 selinux 把权限禁掉了，我们需要添加参数 --privileged=true 来解决挂载的目录没有权限的问题

4.6 查看容器 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 mycentos2
```

也可以直接执行下面的命令直接输出 IP 地址

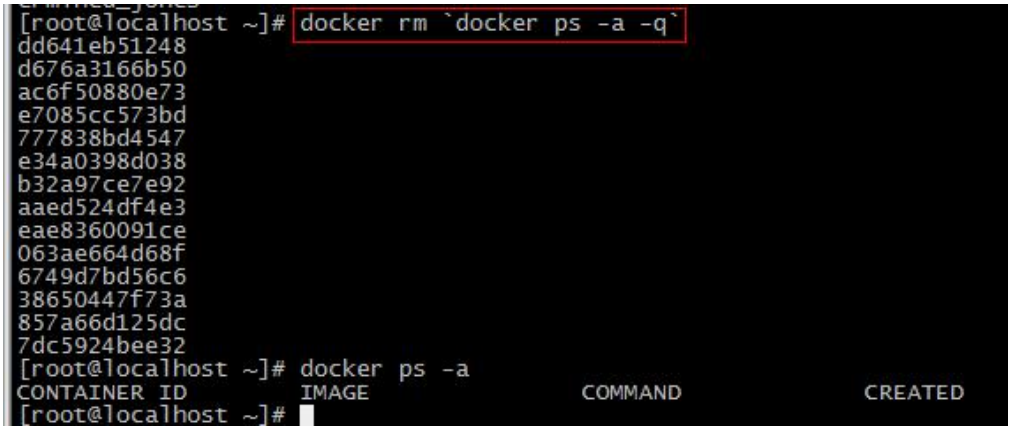
```
docker inspect --format='{{.NetworkSettings.IPAddress}}' mycentos2
```

4.7 删除容器

- 删除指定的容器：docker rm $CONTAINER_ID/NAME

  注意，只能删除停止的容器

- 删除所有容器：docker rm `docker ps -a -q`



5.部署应用

5.1 MySQL 部署

5.1.1 拉取 MySQL 镜像

```
docker pull mysql
```

```
[root@pinyougou-server usr]# docker pull mysql
Using default tag: latest
Trying to pull repository docker.io/library/mysql ...
latest: Pulling from docker.io/library/mysql
9f0706ba7422: Pull complete
2290e155d2d0: Pull complete
547981b8269f: Pull complete
2c9d42ed2f48: Pull complete
55e3122f1297: Pull complete
abc10bd84060: Pull complete
aa37081010bb: Pull complete
aadaa7b95bc6: Pull complete
8781ef2786a7: Pull complete
b5c96613e09e: Pull complete
3eac97813dda: Pull complete
Digest: sha256:75c563c474f1adc149978011fedfe2e6670483d133b22b07ee32789b626f8de3
```

查看镜像

```
[root@pinyougou-server usr]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED
SIZE
docker.io/tomcat    latest              7856f1f03e2d        2 days ago
292.3 MB
docker.io/mysql     latest              ec161391b8c3        6 days ago
412.4 MB
docker.io/redis     latest              a1b99da73d05        10 days ago
105.9 MB
docker.io/centos    7                   36540f359ca3        2 weeks ago
192.5 MB
docker.io/centos    latest              36540f359ca3        2 weeks ago
192.5 MB
docker.io/tomcat    7.0.77-jre7         edd4e9cbd03a        10 weeks ago
357.3 MB
```

5.1.2 创建 MySQL 容器

```
docker run -di --name my_mysql -p 33306:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql
```

-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

5.1.3 进入 MySQL 容器,登陆 MySQL

进入 mysql 容器

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

登陆 mysql

```
mysql -u root -p
```

5.1.4 远程登陆 MySQL

（1）我们在我们本机的电脑上去连接虚拟机 Centos 中的 Docker 容器，这里 192.168.247.130 是虚拟机操作系统的 IP

（2）在本地客户端执行建库脚本

执行“资源/建库语句/mydb.sql”

5.1.5 查看容器 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 my_mysql
```

运行效果如下：


```
"Networks": {
    "bridge": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": null,
        "NetworkID": "9eff60f333fc41a97696a59b2d9c1644fedcea6ba94a1a1fde1a55857b653ab9",
        "EndpointID": "5b450d905c7b01f98356d2110654cfb753c6cf691fe430e205f26b0e17f5b017",
        "Gateway": "172.17.0.1",
        "IPAddress": "172.17.0.2",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:ac:11:00:02"
    }
}
```

我们可以看到我们的数据库服务器的 IP 是 172.17.0.2

5.2 tomcat 部署

5.2.1 拉取 tomcat 镜像

```
docker pull tomcat:7-jre7
```

5.2.2 创建 tomcat 容器

创建容器用于部署单点登录系统（CAS） -p 表示地址映射

```
docker run -di --name=my_tomcat -p 9000:8080
    -v /usr/local/myhtml:/usr/local/tomcat/webapps --privileged=true tomcat:7-jre7
```

```
docker run -di --name=my_tomcat -p 8090:8080 -v /usr/local/apps/tomcat:/usr/local/apps/tomcat --privileged=true tomcat:7-jre7
```

5.2.3 部署 web 应用

（1）修改 cas 系统的配置文件，修改数据库连接的 url

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
        p:driverClass="com.mysql.jdbc.Driver"
        p:jdbcUrl="jdbc:mysql://172.17.0.6:3306/mydb?characterEncoding=utf8"
        p:user="root"
        p:password="123456" />
```

测试：地址栏输入：http://192.168.247.135:9000/cas/login

5.3 Nginx 部署

5.3.1 拉取 Nginx 镜像

```
docker pull nginx
```

5.3.2 创建 Nginx 容器

```
docker run -di --name=my_nginx -p 80:80    nginx/bin/bash
```

5.3.3 测试 Nginx

浏览器地址栏输入： http://192.168.247.135

5.3.4 配置反向代理

官方的 nginx 镜像,nginx 配置文件 nginx.conf 在/etc/nginx/目录下。

在容器内编辑配置文件不方便，我们可以先将配置文件从容器内拷贝到宿主机，编辑修改后再拷贝回去。

（1）从容器拷贝配置文件到宿主机

```
docker cp my_nginx:/etc/nginx/nginx.conf nginx.conf
```

（2）编辑 nginx.conf，添加反向代理配置

```
upstream tomcat-cas {
    server 172.17.0.7:8080;
}
server {
    listen 80;
    server_name passport.pinyougou.com;
    location / {
        proxy_pass http://tomcat-cas;
        index index.html index.htm;
    }
}
```

（3）将修改后的配置文件拷贝到容器

```
docker cp nginx.conf    my_nginx:/etc/nginx/nginx.conf
```

（4）重新启动容器

```
docker restart my_nginx
```

（5）设置域名指向

```
192.168.247.135 passport.pinyougou.com
```

浏览器测试：http://passport.pinyougou.com/cas/login

5.4 Redis 部署

5.4.1 拉取 Redis 镜像

```
docker pull redis
```

5.4.2 创建 Redis 容器

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

5.4.3 客户端测试

在你的本地电脑命令提示符下，用 window 版本 redis 测试

```
redis-cli -h 192.168.247.135
```

6.备份与迁移

6.1 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 my_nginx mynginx
```

my_nginx 是容器名称

mynginx 是新的镜像名称

此镜像的内容就是你当前容器的内容，接下来你可以用此镜像再次运行新的容器

6.2 镜像备份

```
docker    save -o mynginx.tar mynginx
```

-o 输出到的文件

执行后，运行 ls 命令即可看到打成的 tar 包

6.3 镜像恢复与迁移

首先我们先删除掉 mynginx 镜像

然后执行此命令进行恢复

```
docker load -i mynginx.tar
```

-i 输入的文件

执行后再次查看镜像，可以看到镜像已经恢复