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

注册服务器可以理解为 GitHub 这样的托管服务。

按照提示信息，安装所需的工具和库：

[root@ssli-centos7 ~]# yum install gcc make perl

[root@ssli-centos7 ~]# yum install kernel-devel-$(uname -r)

[root@ssli-centos7 ~]# rpm -qa kernel\\*

kernel-3.10.0-693.el7.x86\_64

kernel-tools-3.10.0-693.el7.x86\_64

kernel-headers-3.10.0-693.el7.x86\_64

kernel-devel-3.10.0-693.el7.x86\_64

kernel-tools-libs-3.10.0-693.el7.x86\_64

5）安装docker：

root@ssli-centos7:~$ curl -fsSL https://get.docker.com/ | sh

root@ssli-centos7:~$ systemctl start docker 启动

root@ssli-centos7:~$ systemctl restart docker 重启动

root@ssli-centos7:~$ systemctl status docker

root@ssli-centos7:~$ systemctl enable docker

docker中国官方的镜像源

大致上操作都是一样的，修改/etc/docker/daemon.json文件

vi /etc/docker/daemon.json  
然后加入代码

{

"registry-mirrors": ["https://registry.docker-cn.com"]  
}

然后重启docker

2：从 Docker Hub 仓库下载一个 hello-world的镜像：docker pull hello-world；

有时候官方仓库注册服务器下载较慢，可以从其他仓库下载。 从其它仓库下载时需要指定完整的仓库注册服务器地址：sudo docker pull dl.dockerpool.com:5000/ubuntu:12.04

国内镜像源：docker pull hub.c.163.com/library/tomcat:latest

3：使用 docker images 显示本地已有的镜像、

4：使用该镜像了，例如创建一个容器，让其中运行 bash 应用：docker run hello-world 5：镜像的ID唯一标识了镜像；TAG信息用来标记来自同一个仓库的不同镜像

6：使用docker commit 命令来提交更新后的副本；先使用exit退出容器。

Eg： docker commit 25c03f45a199 centos/java8 25c03f45a199为容器id

7：使用 docker build 来创建一个新的镜像：sudo docker build -t="ouruser/sinatra:v2" .

其中 -t 标记来添加 tag，指定新的镜像的用户信息。 “.” 是 Dockerfile 所在的路径（当前目录），也可以替换为一个具体的 Dockerfile 的路径。

8：还可以用 docker tag 命令来修改镜像的标签；还可以利用 ADD 命令复制本地文件到镜像；用 EXPOSE 命令来向外部开放端口；用 CMD 命令来描述容器启动后运行的程序等；

9：要从本地文件系统导入一个镜像，可以使用 openvz（容器虚拟化的先锋技术）的模板来创建，比如，先下载了一个 ubuntu-14.04 的镜像，之后使用以下命令导入：

sudo cat ubuntu-14.04-x86\_64-minimal.tar.gz |docker import - ubuntu:14.04

12：用户可以通过 docker push 命令，把自己创建的镜像上传到仓库中来共享

13：如果要导出镜像到本地文件，可以使用 docker save 命令

14：可以使用 docker load 从导出的本地文件中再导入到本地镜像库

15：如果要移除本地的镜像，可以使用 docker rmi 命令。注意 docker rm 命令是移除容器。注意：在删除镜像之前要先用 docker rm 删掉依赖于这个镜像的所有容器

16：新建并启动：命令主要为 docker run 。

docker run -t -i ubuntu:14.04 /bin/bash启动一个 bash 终端，允许用户进行交互。其中， -t 选项让Docker分配一个伪终端（pseudo-tty）并绑定到容器的标准输入上， -i 则让容器的标准输入保持打开。

17：当利用 docker run 来创建容器时，Docker 在后台运行的标准操作包括：

检查本地是否存在指定的镜像，不存在就从公有仓库下载

利用镜像创建并启动一个容器

分配一个文件系统，并在只读的镜像层外面挂载一层可读写层

从宿主主机配置的网桥接口中桥接一个虚拟接口到容器中去

从地址池配置一个 ip 地址给容器

执行用户指定的应用程序

执行完毕后容器被终止

18：启动已终止容器：docker start 命令，直接将一个已经终止的容器启动运行。

容器中仅运行了指定的 bash 应用。这种特点使得 Docker 对资源的利用率极高，是货真价实的轻量级虚拟化。

19：守护态运行：需要让 Docker 容器在后台以守护态（Daemonized）形式运行。此时，可以通过添加 -d 参数来实现：docker run -d ubuntu:14.04 /bin/sh

20终止容器: docker stop 来终止一个运行中的容器。Docker容器中指定的应用终结时，容器也自动终止。 例如对于上一章节中只启动了一个终端的容器，用户通过 exit 命令或 Ctrl+d 来退出终端时，所创建的容器立刻终止。终止状态的容器可以用 docker ps -a 命令看到。

docker restart 命令会将一个运行态的容器终止，然后再重新启动它。

21:进入容器:在使用 -d 参数时，容器启动后会进入后台。 某些时候需要进入容器进行操作，有很多种方法，包括使用docker attach 命令或 nsenter 工具等。

attach 命令有时候并不方便。当多个窗口同时 attach 到同一个容器的时候，所有窗口都会同步显示。当某个窗口因命令阻塞时,其他窗口也无法执行操作了。

nsenter 命令: nsenter 可以访问另一个进程的名字空间。nsenter 要正常工作需要有 root 权限。

22:导出和导入容器: 如果要导出本地某个容器，可以使用 docker export 命令。

$ sudo docker ps -a

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

7691a814370e ubuntu:14.04 "/bin/bash" 36 hours ago Exited (0) 21 hours ago test

$ sudo docker export 7691a814370e > ubuntu.tar这样将导出容器快照到本地文件。

导入容器快照: 可以使用 docker import 从容器快照文件中再导入为镜像，

$ cat ubuntu.tar | sudo docker import - test/ubuntu:v1.0

$ sudo docker images

REPOSITORY TAG IMAGE ID CREATED VIRTUAL SIZE

test/ubuntu v1.0 9d37a6082e97 About a minute ago 171.3 MB

此外，也可以通过指定 URL 或者某个目录来导入

$sudo docker import http://example.com/exampleimage.tgz example/imagerepo

注：用户既可以使用 docker load 来导入镜像存储文件到本地镜像库，也可以使用 docker import 来导入一个容器快照到本地镜像库。这两者的区别在于容器快照文件将丢弃所有的历史记录和元数据信息（即仅保存容器当时的快照状态），而镜像存储文件将保存完整记录，体积也要大。此外，从容器快照文件导入时可以重新指定标签等元数据信息。

23: 可以使用 docker rm 来删除一个处于终止状态的容器。 例如

$sudo docker rm trusting\_newton

trusting\_newton

如果要删除一个运行中的容器，可以添加 -f 参数。Docker 会发送 SIGKILL 信号给容器。

24: 实际上注册服务器是管理仓库的具体服务器，每个服务器上可以有多个仓库，而每个仓库下面有多个镜像

25: 挂载主机目录作为数据卷：创建一个数据卷: 在用 docker run 命令的时候，使用 -v 标记来创建一个数据卷并挂载到容器里。在一次 run 中多次使用可以挂载多个数据卷。

例：docker run -it --ame centos –v /usr/docker:/usr/docker hub.c.163.com/cookienull/centos7\_pure\_lnmp /bin/bash

挂载本地文件作为数据卷：docker run -it -v ~/.bash\_history:/.bash\_history ubuntu /bin/bash可以记录在容器输入过的命令了

数据卷容器：在其他容器中使用 --volumes-from 来挂载 dbdata 容器中的数据卷。可以使用多个 --volumes-from 参数来从多个容器挂载多个数据卷。 也可以从其他已经挂载了数据卷的容器来挂载数据卷。

26：利用数据卷容器来备份、恢复、迁移数据卷：

首先使用 --volumes-from 标记来创建一个加载 dbdata 容器卷的容器，并从本地主机挂载当前到容器的/backup 目录，容器启动后，使用了 tar 命令来将 dbdata 卷备份为本地的 /backup/backup.tar。

如果要恢复数据到一个容器，首先创建一个带有数据卷的容器 dbdata2。

然后创建另一个容器，挂载 dbdata2 的容器，并使用 untar 解压备份文件到挂载的容器卷中。

27：容器的网络功能：容器中可以运行一些网络应用，要让外部也可以访问这些应用，可以通过 -P 或 -p 参数来指定端口映射。当使用 -P 标记时，Docker 会随机映射一个 49000~49900 的端口到内部容器开放的网络端口。

docker run -d -P training/webapp python app.py

docker ps –l查看端口映射。

-p（小写的）则可以指定要映射的端口，并且，在一个指定端口上只可以绑定一个容器。支持的格式有ip:hostPort:containerPort | ip::containerPort（绑定 localhost 的任意端口到容器的 5000 端口，本地主机会自动分配一个端口） | hostPort:containerPort（此时默认会绑定本地所有接口上的所有地址。） docker run -d -p 5000:5000 training/webapp python app.py

还可以使用 udp 标记来指定 udp 端口

使用 docker port 来查看当前映射的端口配置，也可以查看到绑定的地址

28：容器有自己的内部网络和 ip 地址（使用 docker inspect 可以获取所有的变量，Docker 还可以有一个可变的网络配置。）

-p 标记可以多次使用来绑定多个端口

29：使用 --link 参数可以让容器之间安全的进行交互。该系统会在源和接收容器之间创建一个隧道，接收容器可以看到源容器指定的信息。

--link 参数的格式为 --link name:alias ，其中 name 是要链接的容器的名称， alias 是这个连接的别名。web 容器链接到 db 容器，web 容器将被允许访问 db 容器的信息。

30：Docker 通过 2 种方式为容器公开连接信息：

（1）环境变量

（2）更新 /etc/hosts 文件

env 命令来查看 web 容器的环境变量

用户可以链接多个父容器到子容器，比如可以链接多个 web 到 db 容器上

31：高级网络配置：当 Docker 启动时，会自动在主机上创建一个 docker0 虚拟网桥，实际上是 Linux 的一个 bridge，可以理解为一个软件交换机。它会在挂载到它的网口之间进行转发。Docker 随机分配一个本地未占用的私有网段（在 RFC1918 中定义）中的一个地址给 docker0 接口。比如典型的 172.17.42.1 ，掩码为 255.255.0.0 。此后启动的容器内的网口也会自动分配一个同一网段（ 172.17.0.0/16 ）的地址。

当创建一个 Docker 容器的时候，同时会创建了一对 veth pair 接口（当数据包发送到一个接口时，另外一个接口也可以收到相同的数据包）。这对接口一端在容器内，即 eth0 ；另一端在本地并被挂载到docker0 网桥，名称以 veth 开头（例如 vethAQI2QT ）。通过这种方式，主机可以跟容器通信，容器之间也可以相互通信。Docker 就创建了在主机和所有容器之间一个虚拟共享网络。

32：容器要想访问外部网络，需要本地系统的转发支持。在Linux 系统中，检查转发是否打开。$sysctl net.ipv4.ip\_forward

net.ipv4.ip\_forward = 1

如果为 0，说明没有开启转发，则需要手动打开。$sysctl -w net.ipv4.ip\_forward=1

如果在启动 Docker 服务的时候设定 --ip-forward=true , Docker 就会自动设定系统的 ip\_forward 参数为 1。

33：默认情况下，不同容器之间是允许网络互通的。如果为了安全考虑，可以在 /etc/default/docker文件中配置 DOCKER\_OPTS=--icc=false 来禁止它。

34：默认情况下，容器可以主动访问到外部网络的连接，但是外部网络无法访问到容器

容器所有到外部网络的连接，源地址都会被NAT成本地系统的IP地址。这是使用 iptables 的源地址伪装操作实现的。

35:容器允许外部访问，可以在 docker run 时候通过 -p 或 -P 参数来启用。

不管用那种办法，其实也是在本地的 iptable 的 nat 表中添加相应的规则。

**2.3 docker镜像管理从Docker Hub 中操作**

**2.3.1 search镜像**

[root@docker ~]# docker search centos:6

**2.3.2 push镜像**

[root@docker ~]# docker pull centos:6

**2.3.3 查看镜像**

[root@docker ~]# docker images

**2.3.4 导入/导出镜像**

[root@docker ~]# docker save centos:6 >/opt/centos.tar.gz ==》导出镜像

[root@docker ~]# docker load </opt/centos.tar.gz ==》导入镜像

**2.3.5 删除镜像**

[root@docker ~]# docker rmi $ IMAGE ID/$TAG

**2.3.6 利用镜像创建容器**

[root@docker ~]# docker run -it centos:6 /bin/bash

#### attach命令进入

[root@docker ~]# docker attach $CONTAINER ID

注意：该命令是系统自带的，并且启用attach命令有时候并不是特别的方便，当多个窗口同时attach到一个容器的时候，多有的窗口都会同步显示，当某个窗口因某个命令堵塞时，其他窗口也就无法再执行命令了，在退出容器时，该容器会自动停止

#### nsenter命令进入

[root@docker ~]# yum install -y util-linux ==》安装nsenter

[root@docker ~]# docker inspect --format "{{.State.Pid}}" test ==》找到容器进程ID

[root@docker ~]# nsenter -t 19245 -u -i -m –p ==》进入容器

-t, --target <pid> target process to get namespaces from

指定容器的进程ID

-m, --mount[=<file>] enter mount namespace

进入到mount namespace空间中

-u, --uts[=<file>] enter UTS namespace (hostname etc)

进入到UTS namespace空间中

-i, --ipc[=<file>] enter System V IPC namespace

进入到System V IPC namespace空间中

-n, --net[=<file>] enter network namespace

进入到network namespace空间中

-p, --pid[=<file>] enter pid namespace

进入到pid namespace空间

#### 2.4.4.3 编写脚本快速进入容器

[root@docker scripts]# vim docker\_in.sh

#/bin/bash

PID=$(docker inspect -f "{{.State.Pid}}" $1)

nsenter -t $PID -m -u -i -n -p

 以特权方式启动容器

指定--privileged参数

如：# docker run -it --privileged=true -v /test:/soft centos /bin/bash

**docker挂载本地目录实现文件共享：**

镜像改名

docker tag 镜像ID xxxx:xxxx

例子：  
docker tag 899bd984 lyb/php:1.0

删除特殊镜像

使用repository：tag的组合来删除特殊的镜像

端口映射：docker run  -p hostPort:containerPort  redis

docker run -it --privileged=true -v /usr/docker/test/:/usr/test -p 26381:26381 centos/java8/wrapper:latest /bin/bash

查看端口开放：lsof -i:26381什么都没有则没有开放

查看docker容器中的ip：docker inspect --format='{{.NetworkSettings.IPAddress}}' 56a390301a17

docker启动容器时会出现标题所示的报错：

Error response from daemon: Cannot start container web: iptables failed: iptables -t nat -A DOCKER -p tcp -d 0/0 –dport 32797 -j DNAT –to-destination 172.17.0.30:5000 ! -i docker0: iptables: No chain/target/match by that name.

解决办法：重建docker0网络恢复

依次执行下面的命令：

pkill docker

iptables -t nat -F

ifconfig docker0 down

brctl delbr docker0

docker -d

进入docker操作

docker exec -it 30c339acbe79 /bin/bash

service docker restart

默认情况下容器使用的资源是不受限制的

使用kubernetes来实现一个机器上的docker负载均衡

Docker相关的本地资源存放在/var/lib/docker/目录下，其中container目录存放容器信息，graph目录存放镜像信息，aufs目录下存放具体的镜像底层文件。

7：临时退出一个正在交互的容器的终端，而不终止它：按Ctrl+p，后按Ctrl+q，如果按Ctrl+c会使容器内的应用进程终止，进而会使容器终止。

8：很多应用容器都是默认后台运行的，怎么查看它们的输出和日志信息：docker logs，后面跟容器的名称或者ID信息；

9：docker的配置文件位置：Ubuntu系统下Docker的配置文件是/etc/default/docker，CentOS系统配置文件存放在/etc/sysconfig/docker

解决docker中java程序乱码问题：

vi /etc/profile

export LC\_ALL=en\_US.UTF-8

source /etc/profile

vi /etc/locale.conf

LANG="en\_US.UTF-8"

source /etc/locale.conf

每次docker重启需要重新执行一下source /etc/profile和source /etc/locale.conf

据说在 Linux Docker中无法使用 systemd(systemctl) 相关命令的原因是 1号进程不是 init ，而是其他例如 /bin/bash ，所以导致缺少相关文件无法运行。（System has not been booted with systemd as init system (PID 1). Can't operat）

解决方案：/sbin/init

例如：Ubuntu 18.04 ，

docker run -tid --name test --privileged=true ubuntu:18.04 /sbin/init

docker exec -it test /bin/bash

PS:--privilaged=true一定要加上的。

# [关于Docker目录挂载的总结](https://www.cnblogs.com/ivictor/p/4834864.html)

<https://www.cnblogs.com/ivictor/p/4834864.html>

centos7/8防火墙操作

开启端口  
[root@centos7 ~]# firewall-cmd --zone=public --add-port=80/tcp --permanent

查询端口号80 是否开启：

[root@centos7 ~]# firewall-cmd --query-port=80/tcp

重启防火墙：

[root@centos7 ~]# firewall-cmd --reload

查询有哪些端口是开启的:

[root@centos7 ~]# firewall-cmd --list-port

systemctl stop firewalld.service #停止firewall

systemctl disable firewalld.service #禁止firewall开机启动