https://www.docker.com/

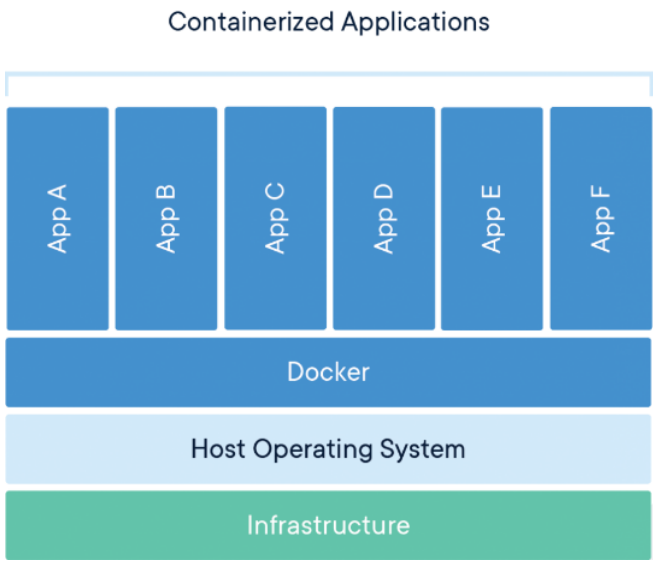
菜鸟教程https://www.runoob.com/docker/docker-command-manual.html

什么是dockerhttp://dockone.io/article/6051?tdsourcetag=s_pcqq_aiomsg

目前为止使用centos8 安装不了，官方推荐使用centos7的系统

docker是一个容器管理软件。轻量级的虚拟化技术（vmware），可以构建、迁移、运行任何程序在任何地方

Container --》容器--》容器就是将软件打包成标准化单元（其实本质上是一个进程，进程里跑一个应用）。

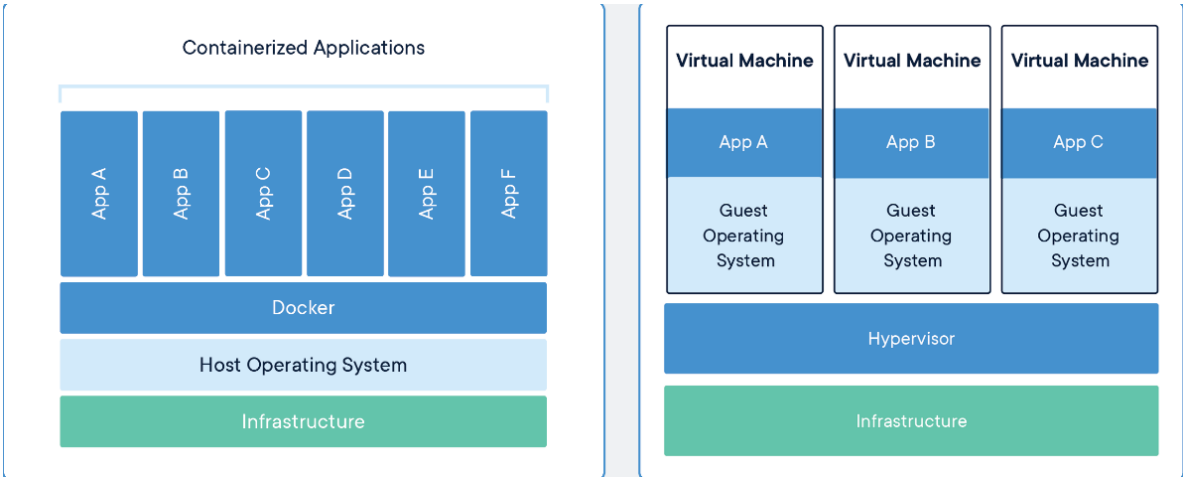


容器是一个标准的软件单元，它打包代码及其所有依赖项，以便应用程序从一个计算环境快速可靠地运行到另一个计算环境。无论基于Linux还是基于Windows的应用程序，无论基础设施如何，容器化软件都将始终运行相同的程序。容器将软件从其环境中隔离出来，并确保它统一工作，而不管例如开发和试运行之间的差异。容器共享机器的操作系统内核，因此每个应用程序不需要操作系统，提高了服务器效率

比较容器和虚拟机

例如，Docker守护进程可以直接与主操作系统进行通信，为各个Docker容器分配资源；它还可以将容器与主操作系统隔离，并将各个容器互相隔离。虚拟机启动需要数分钟，而Docker容器可以在数毫秒内启动。由于没有臃肿的从操作系统，Docker可以节省大量的磁盘空间以及其他系统资源。

两者有不同的使用场景。虚拟机更擅长于彻底隔离整个运行环境。例如，云服务提供商通常采用虚拟机技术隔离不同的用户。而Docker通常用于隔离不同的应用。容器不是模拟一个完整的操作系统，而是对进程进行隔离



容器

容器是应用程序层的抽象，将代码和依赖项打包在一起。多个容器可以在同一台机器上运行，并与其他容器共享操作系统内核，每个容器作为用户空间（cpu分为两部分，用户空间、内核空间）中的独立进程运行。容器占用的空间比虚拟机少，可以处理更多的应用程序。

虚拟机

虚拟机是物理硬件的抽象，将一台服务器转变为多台服务器。虚拟机管理程序允许多个虚拟机在一台机器上运行。每个虚拟机都包括一个操作系统、应用程序、必要的二进制文件和库的完整拷贝，占用很多空间。虚拟机启动也可能很慢。

docker和vmware都是虚拟化技术，docker好的在哪里?

可以对cpu、内存、磁盘io等资源进行限制。
部署非常方便，比虚拟机更加节省资源，速度更加快（启动、关闭、新建、删除），体积小

docker里的容器是如何隔离的，它的底层原理是什么?

一个容器对应操作系统里的一个进程，进程和进程之间是隔离的，是linux 内核管控的。
一个容器对应一个name space，里面的内容和别的name space里的内容可以一样也可以不一样。（同一个进程，在不同的命名空间进程号不同，命名空间保证了容器之间互不影响）

使用Namespaces实现了系统环境的隔离，Namespaces允许一个进程以及它的子进程从共享的宿主机内核资源（网络栈、进程列表、挂载点等）里获得一个仅自己可见的隔离区域（通过提供命名空间，可以让进程与进程之间，用户与用户之间彼此看不到对方）；

使用CGroups限制这个环境的资源使用情况，比如一台16核32GB的机器上只让容器使用2核4GB。使用CGroups还可以为资源设置权重，计算使用量，操控任务（进程或线程）启停等；

时间都是使用的是操作系统里的时间

Docker 的主要用途，目前有三大类

提供一次性的环境。比如，本地[测试他人的软件、持续集成的时候提供单元测试和构建的环境。

提供弹性的云服务。因为 Docker 容器可以随开随关，很适合动态扩容和缩容。

组建微服务架构。通过多个容器，一台机器可以跑多个服务，因此在本机就可以模拟出微服务架构

docker优势：快速启动删除数量巨大的容器，解决重复安装服务器和相关服务
缺点：需要定制镜像

过程

命令--官方https://docs.docker.com/engine/reference/commandline

docker镜像仓库https://hub.docker.com/

官方安装https://docs.docker.com/engine/install/centos/

安装
[root@localhost ~]# yum install docker -y

启动
[root@localhost ~]# service docker start
Redirecting to /bin/systemctl start docker.service

如果docker启动不了，建议关闭selinux。

设置docker开机启动
[root@docker ~]# systemctl enable docker

查看docker的版本
[root@docker ~]# docker version
Client:
 Version: 1.13.1
 API version: 1.26
 Package version: docker-1.13.1-96.gitb2f74b2.el7.centos.x86_64
 Go version: go1.10.3
 Git commit: b2f74b2/1.13.1
 Built: Wed May 1 14:55:20 2019
 OS/Arch: linux/amd64

Server:
 Version: 1.13.1
 API version: 1.26 (minimum version 1.12)
 Package version: docker-1.13.1-96.gitb2f74b2.el7.centos.x86_64
 Go version: go1.10.3
 Git commit: b2f74b2/1.13.1
 Built: Wed May 1 14:55:20 2019
 OS/Arch: linux/amd64
 Experimental: false
[root@docker ~]#

查看docker镜像
[root@localhost ~]# docker images
REPOSITORY   TAG   IMAGE ID   CREATED   SIZE
[root@localhost ~]#

搜索所有评价为3星级以上的带Nginx关键字的镜像
[root@Mariadb ~]# docker search nginx
[root@Mariadb ~]# docker search   --filter stars=3 nginx

[root@docker ~]# docker pull docker.io/nginx 下载nginx的镜像（docker官方网站）

如果不指定版本号，会下载最新的版本（latest）
[root@docker pub]# docker pull docker.io/mysql:5.7

使用国内的镜像安装nginx
https://hub.tenxcloud.com/ 时速云
到时速云下载镜像

报错

Error response from daemon: Get https://index.docker.io/v1/search?q=nginx&n=25: net/http: TLS handshake timeout

解决：镜像加速的方法：改用中国的 docker 镜像仓库
[root@mysql ~]# vim /etc/docker/daemon.json
[root@mysql ~]# cat /etc/docker/daemon.json
{
"registry-mirrors":["https://registry.docker-cn.com"]

}

导出一个镜像名叫nginx.tar
[root@docker ~]# docker save docker.io/nginx -o nginx.tar #save命令用于持久化镜像（不是容器）
[root@docker ~]# docker export <CONTAINER ID> -o /home/export.tar #导出正在运行的容器成一个镜像
-o file_path 相同与 > file_path

在其他机器上导入镜像
[root@localhost ~]# docker load < nginx.tar
-i file_path 相同与 < file_path

docker中镜像的相关文档 主页面 -》products -》docker hub -》go to docker hub

创建一个容器

docker create **
docker start **
docker ps
docker run 等价于 docker create 后再执行docker start（如果镜像没有，会先pull再启动）

docker pull index.tenxcloud.com/docker_library/nginx
docker create -it index.tenxcloud.com/docker_library/nginx
docker start <CONTAINER ID>
docker ps -a 查看创建的容器(正在运行的容器)

```
1  docker run -d -p 80:80 --name my_nginx
   index.tenxcloud.com/docker_library/nginx
2  -d 作为后台的进程运行,没有接-d,将不能访问网站
3  -p 端口映射  80:80  访问本机的80端口，映射到容器里的80端口  底层是iptables
4  --name 是容器的名字
```

查看
[root@Mariadb ~]# docker ps -a 查看所有的容器
[root@Mariadb ~]# docker ps 查看运行的容器
[root@Mariadb ~]# docker port my_nginx 端口查看

查看容器的日志
[root@mysql web]# docker logs -f my_nginx3
192.168.0.51 - - [19/Apr/2018:03:07:26 +0000] "GET / HTTP/1.1" 200 12 "-" "curl/7.29.0" "-"

进入nginx1容器的内部，查看内容
[root@localhost web]# docker exec -it nginx1 /bin/bash
root@e916a2712a94:/etc/nginx# cd /var/log
root@e916a2712a94:/var/log# ls

alternatives.log apt bootstrap.log btmp dmesg dpkg.log faillog fsck lastlog nginx wtmp

终止
[root@Mariadb ~]# docker stop my_nginx 使用容器名字停止容器
[root@Mariadb ~]# docker stop 62156f4bd64f 使用容器id停止容器
[root@Mariadb ~]# docker restart my_nginx 重启容器
[root@Mariadb ~]# docker start my_nginx 将停止运行的容器启动

修改了配置文件，如何让配置文件生效?
[root@docker mysql]# docker restart sc_mysql_21

进入容器内部
[root@Mariadb ~]# docker exec -it my_nginx /bin/bash

```
-i   交互模式启动容器 interaction
-d   将容器放到后台执行 deamon
-t   开启一个终端显示   terminate
/bin/bash   执行/bin/bash程序   ，就可以和容器交互了
```

删除容器
[root@Mariadb ~]# docker rm <CONTAINER ID> 正在运行的不能删除

docker rm $(docker ps -aq) # 删除全部容器

删除镜像
docker rmi runoob/ubuntu:v4

docker rmi `docker image -q` # 删除所有的镜像

强制删除容器 db01、db02
docker rm -f db01 db02

得到某个容器的底层信息包括此容器的ip，网关

docker inspect 容器

得到容器dns服务器地址---》容器的dns服务器和宿主机是一样的

进入容器后 cat /etc/resolv.conf

用于容器与主机之间的数据拷贝

docker cp

```
将主机/www/runoob目录拷贝到容器96f7f14e99ab的/www目录下。
docker cp /www/runoob 96f7f14e99ab:/www/

将容器96f7f14e99ab的/www目录拷贝到主机的/tmp目录中。
docker cp  96f7f14e99ab:/www /tmp/
```

启动mysql
[root@docker pub]# docker run -d -p 3306:3306 --name sc_mysql_57 -e
MYSQL_ROOT_PASSWORD='sc123456' mysql:5.7
-e 作用是往容器里传递参数
MYSQL_ROOT_PASSWORD='sc123456' 给mysql设置密码

如何连接到容器的mysql里?
[root@docker pub]# mysql -h 127.0.0.1 -P 3306 -uroot -psc123456
（在此机器安装mariadb的软件包，获得一个可以连接到mysql里的客户端命令）

```
如果我需要去修改配置文件，例如开启二进制日志，修改server-id、慢日志等功能，如何实现？
    1.自己去制作一个符合你所有要求和功能的docker镜像文件，然后去启动
    2.数据卷功能：  将宿主机里的文件共享到容器里，让容器里的mysql使用我们宿主机里的配置文件，启动mysql
```

容器里的数据是保存在哪里的?

```
[root@docker docker]# cd /var/lib/docker/
[root@docker docker]# ls
builder  buildkit  containers  image  network  overlay2  plugins  runtimes  swarm  tmp  trust  volumes
[root@docker docker]# cd containers/
[root@docker containers]# ls
067b088cea5791b4b55400e3784f1c40358c196e0d7f06e5806340ff18978685  84506057e97871e2cf90b3c9b102b578adb6357c7e45a9aff8ac3ee621e4f780
59df04a5a697bf56d767139eac120ad20a6c7ebf2cb0f6bc65318a271e6f1cca  87cb19970910b31e69a73ae4a28bebcbbc4f4679aaa1dd98c4b98005cdf55a16
5c722d9ef069eb41896d5624b6c809da2d170d99b0974e8bc9016ddd76a92ff4  b58c5b2458e53af510ff119a51324c2d538035be3e910e1c32b31a8c3acf0b97
7964cfc4f9b51cae0e64ad5a06870268a80eadc13efa5abbaacfae64e5bab5fb  d575da09a5b2b005516799661e861ed461e25daa1365fb0873670aeba69e9caa
[root@docker containers]#
```

```
容器里的数据是保存在哪里的？
    /var/lib/docker/containers/
docker容器是一个进程，进程是工作内存里的，如果进程被杀死都导致数据丢失
如果关闭docker服务会导致所有的容器停止服务。

如果把容器停止，容器里的数据还有吗？--》有
    重新启动容器还是有数据

config.v2.json 记录具体容器信息的文件
```

[root@docker pub]# docker run -d -p 6397:6397 --name sc_redis docker.io/redis

容器的名字使用过，就会被记住，而且是唯一，不用的话，建议删除

数据卷：真实linux机器和容器进行数据共享的平台。

volume
使用数据卷，挂载本地的目录到容器里，访问网页内容是自己定义的。

数据卷的挂载是在启动时完成

方便了数据备份

-v /web:/usr/share/nginx/html 将当前linux系统里的/web目录挂载到容器里的/usr/share/nginx/html目录下

```
[root@mysql web]# cat index.html
hello,world
[root@mysql web]# docker run -d -p 82:80 --name my_nginx3 -v
/web:/usr/share/nginx/html index.tenxcloud.com/docker_library/nginx

44bda7eae69c810609196fe0bb7b3c67b0acd0a54e0082120e527e98db477a6f
[root@mysql web]# curl  192.168.0.51:82 外面直接访问
hello,world
```

把宿主机的配置文件挂载到容器中

第一次启动容器，把宿主机的目录挂载到容器的 /mnt 目录下，把配置文件 cp 到 /mnt 目录下，这样宿主机就有了配置文件，第二次启动容器并挂载。

镜像

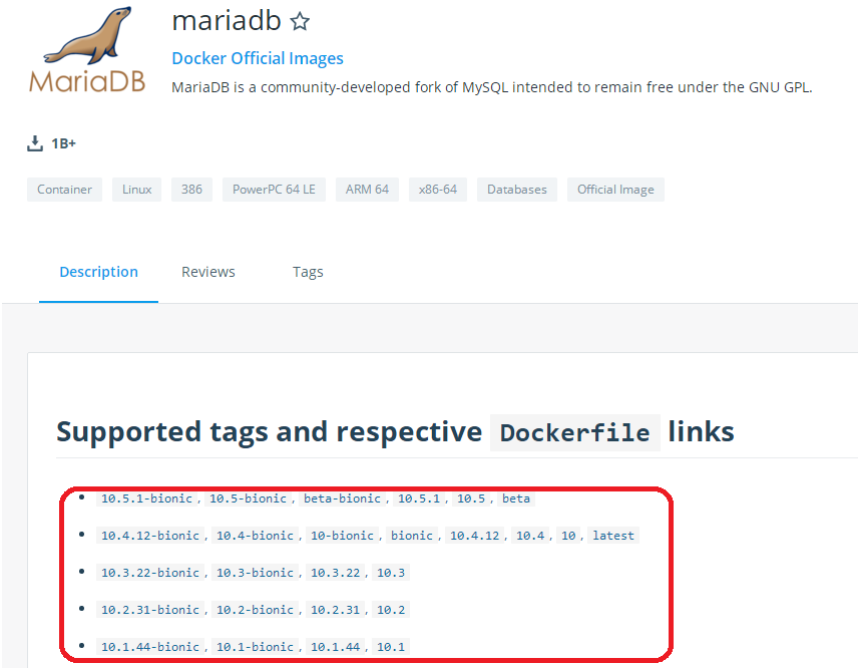
国内镜像https://hub.daocloud.io/repos/2b7310fb-1a50-48f2-9586-44622a2d1771

使用官方的镜像是否可以满足公司的需要?

答案：不可以（也可以）

为什么要制作镜像，docker hub上不是有很多镜像吗?
答案：
1.不能满足我们的需要
2.不够安全，有安全隐患

dockerfile是制作镜像的配置文件，到docker官网提供的镜像文件处的获取



使用官方的教程自己使用Dockerfile创建一个docker镜像
官方网址：
https://docs.docker.com/get-started/part2/#run-the-app

事例Dockerfile （可以去dockerhub看别人是怎么写的）

```
FROM centos



#ENV 设置环境变量
ENV PATH /usr/sbin/:$PATH

#ADD  将文件(文件夹的话，会将文件夹下的文件)拷贝进容器，文件放在当前目录下，拷过去会自动解压
ADD nginx.conf /backup

#RUN 执行以下命令
RUN yum install epel-release -y
RUN yum install -y net-tools  vim
RUN yum install -y nginx

#这会创建3层，使用 && 连接，会创建一层

#WORKDIR 相当于cd
WORKDIR /

#EXPOSE 映射端口
EXPOSE 80

```

```
24
25
26 #容器里接受终止进程的信号为SIGTERM
27 STOPSIGNAL SIGTERM
28
29 #CMD 运行以下命令
30 CMD ["nginx", "-g", "daemon off;"]  #在前台启动nginx程序 ，-g  daemon  off 将
   off值赋给daemon这个变量，告诉nginx不要在后台启动，在前台启动
31 daemon是守护进程--》默认在后台运行
32
33 nginx -g选项的作用是 设置一个全局的变量 ，给它赋值
34
```

docker build --tag pyynginx .

```
1  [root@manager17 docker]# cat Dockerfile
2  FROM docker.io/sglim2/centos7
3
4
5  WORKDIR /
6  COPY nginx-1.19.0.tar.gz  /
7
8  RUN yum install zlib zlib-devel openssl openssl-devel pcre pcre-devel gcc
   gcc-c++ autoconf automake make  -y \
9      && tar xf nginx-1.19.0.tar.gz \
10     && cd nginx-1.19.0 \
11     &&  ./configure  --prefix=/usr/local/nginx8   --with-threads  --with-
   file-aio  --with-http_ssl_module  --with-http_stub_status_module  --with-
   stream  \
12     &&make -j 2 ; make install  \
13     &&mkdir /app \
14     && sed -i '44 c root /app ;' /usr/local/nginx8/conf/nginx.conf
15 #将网页的位置变为 /app，因为宿主机挂载容器的/usr/share/nginx/html会报错
16
17
18 ENV PATH /usr/local/nginx8/sbin:$PATH   #设置环境变量
19 EXPOSE 80
20 STOPSIGNAL  SIGTERM
21
22 ENTRYPOINT ["nginx"]
23 CMD ["-g","daemon off;"]
24 #CMD ["nginx","-g","daemon off;"]
```

容器数据卷挂载nfs服务器

搭建nfs服务器,发布共享文件。

docker服务器上安装nfs-utils，方便挂载到nfs服务器山上

`yum install -y nfs-utils rpcbind`

docker服务器上创建volume，并挂载nfs

`docker volume create --driver local --opt type=nfs --opt o=addr=nfs服务器ip,rw --opt device=:/web  数据卷名`

启动容器使用数据卷

```
docker run -d -p 8899:80  --name  lijian  -v 数据卷名:/app  sc_nginx5
```

如果是swarm集群，那么所有节点需要创建一样的名字的数据卷挂载nfs服务器的相同共享目录

swarm集群，在manager节点上创建服务
 docker service create --name nfs-sc-nginx --publish 9988:80 --mount type=volume,source=数据卷名,destination=/app --replicas 4 sc_nginx5

关于数据卷的创建查看删除https://docs.docker.com/storage/volumes/

---

docker的网络

一种网络类型对应一个name space 实现隔离

172.17.0.0/16 私有网段

宿主机是所有容器的网关

把所有容器看作是虚拟机，容器间互相访问使用ip

```
得到某个容器的底层信息，包括此容器的ip，网关
[root@docker mydocker]# docker inspect 容器

eg:
[root@docker mydocker]# docker inspect optimistic_heyrovsky    得到ip地址和网关
                    "Gateway": "172.17.0.1",
                    "IPAddress": "172.17.0.11",

容器中查看，得到dns服务器地址---》容器的dns服务器和宿主机是一样的
root@d1a138a940bb:/app# cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 114.114.114.114
root@d1a138a940bb:/app#
```

[root@docker mydocker]# iptables -L -vn 可以查看所有的容器的ip

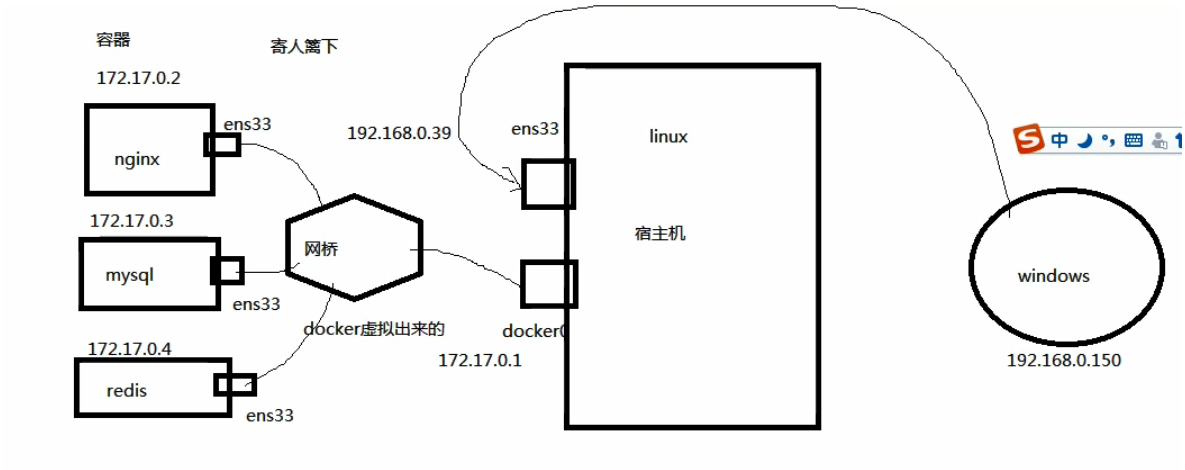
多容器之间的链接：
 --link 选项：是让2个容器之间建立联系，其实就是在容器里的/etc/hosts文件添加一个主机名对应某个ip地址，可以使用主机名直接访问，这一点在容器中查看环境变量可以看到

启动Redis持久化功能

```
docker run -d -p 4001:80 --name web_flask --link sc_redis:redis friendlyhello

flask web容器链接到redis容器（sc_redis）,链接别名必须叫redis，因为app.py里的连接redis的函数里
指定的主机名是redis，flask容器会在容器的/etc/hosts文件里记录对应的sc_redis的ip地址
172.17.0.10 redis add142e7eb43 sc_redis
```

bridge（默认的模式）



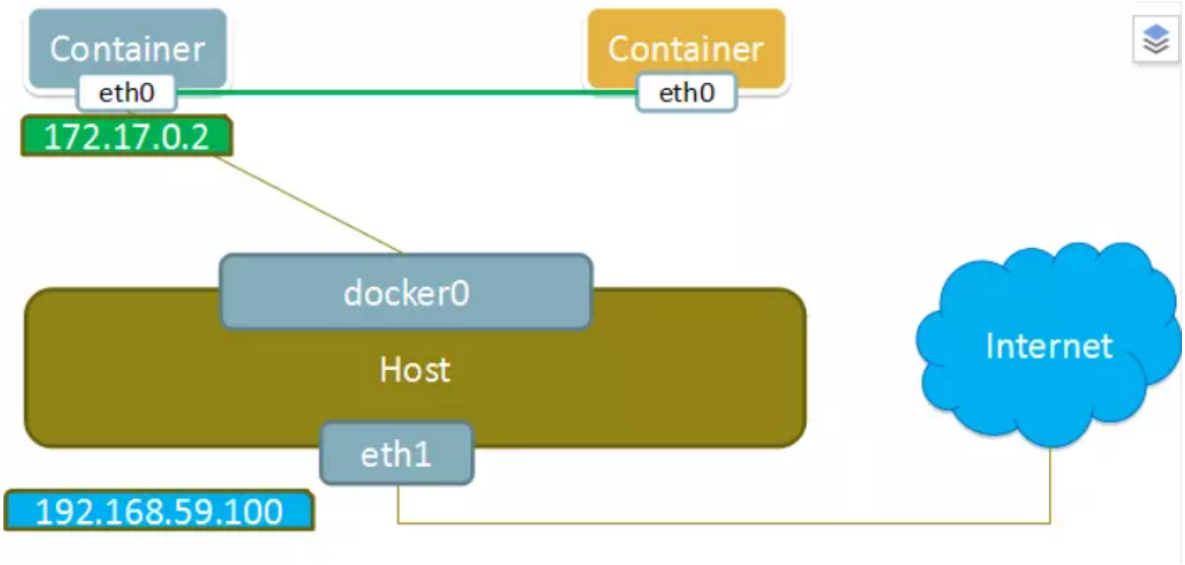
网桥上虚拟了很多接口（veth），在真实机上可通过ip add看到

Docker网桥是宿主机虚拟出来的，并不是真实存在的网络设备，外部网络是无法寻址到的，这也意味着外部网络无法通过直接Container-IP访问到容器。如果容器希望外部访问能够访问到，可以通过映射容器端口到宿主主机（端口映射），docker实际是在iptables做了DNAT规则，实现端口转发功能。

host

和宿主机共用一个Network Namespace，使用宿主机的IP和端口,端口号要区别开。但是，容器的其他方面，如文件系统、进程列表等还是和宿主机隔离的。使用host模式的容器可以直接使用宿主机的IP地址与外界通信，不需要进行NAT，host最大的优势就是网络性能比较好，但是docker host上已经使用的端口就不能再用了，网络的隔离性不好。

container



容器和容器共享一个 Network Namespace(很多容器共享一个ip地址)，而不是和宿主机共享。新创建的容器不会创建自己的网卡，配置自己的 IP，而是和一个指定的容器共享 IP、端口范围等。同样，两个容器除了网络方面，其他的如文件系统、进程列表等还是隔离的。两个容器的进程可以通过 lo 网卡设备通信。

none

容器只有lo回环网络，没有其他网卡。none模式可以在容器创建时通过--network=none来指定。这种类型的网络没有办法联网，封闭的网络能很好的保证容器的安全性。

以上4种模式，只是考虑宿主机和容器之间的通信

overlay

https://docs.docker.com/network/overlay/

容器监控Prometheus

步骤https://docs.docker.com/config/daemon/prometheus/

官方https://prometheus.io/

Prometheus由三个部分组成：Prometheus server、Alertmanager和exporters。exporters以独立进程或容器的方式运行在目标机器上，生成各种指标数据，通过API的方式发送给Prometheus server。Prometheus server负责服务发现，也可以从exporters直接拉取指标数据，然后将数据存储在Prometheus的数据库中，用于可视化或告警服务。Alertmanager用于设置告警规则，分析Prometheus数据库中的数据，当触发某个规则时，向接收者发送警报。在这里可以找到大量的exporters，它们都得到了Prometheus的官方支持和社区的维护。
Prometheus以作为Docker容器运行。

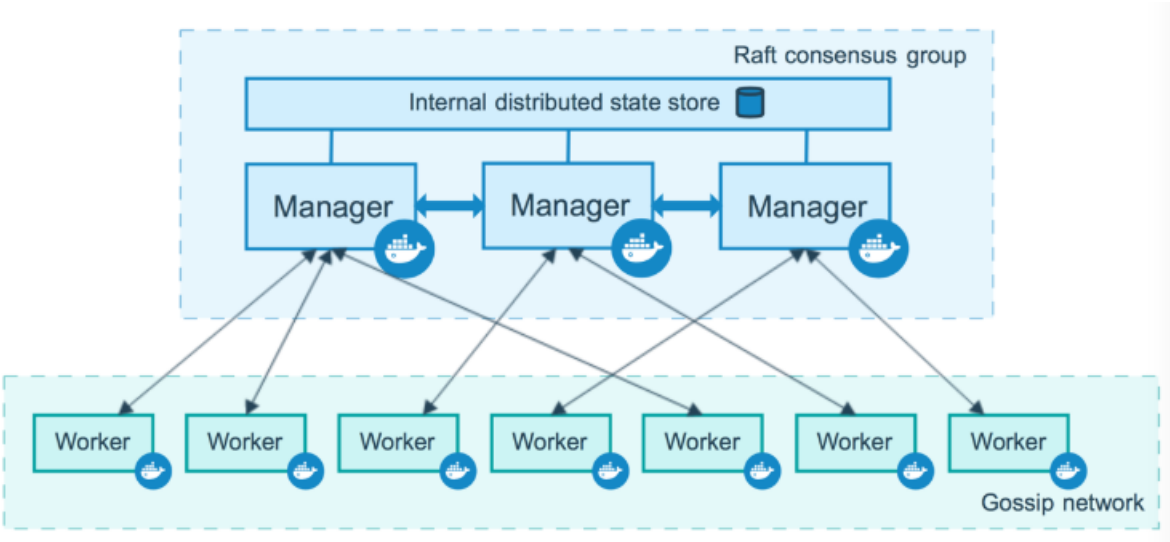
---

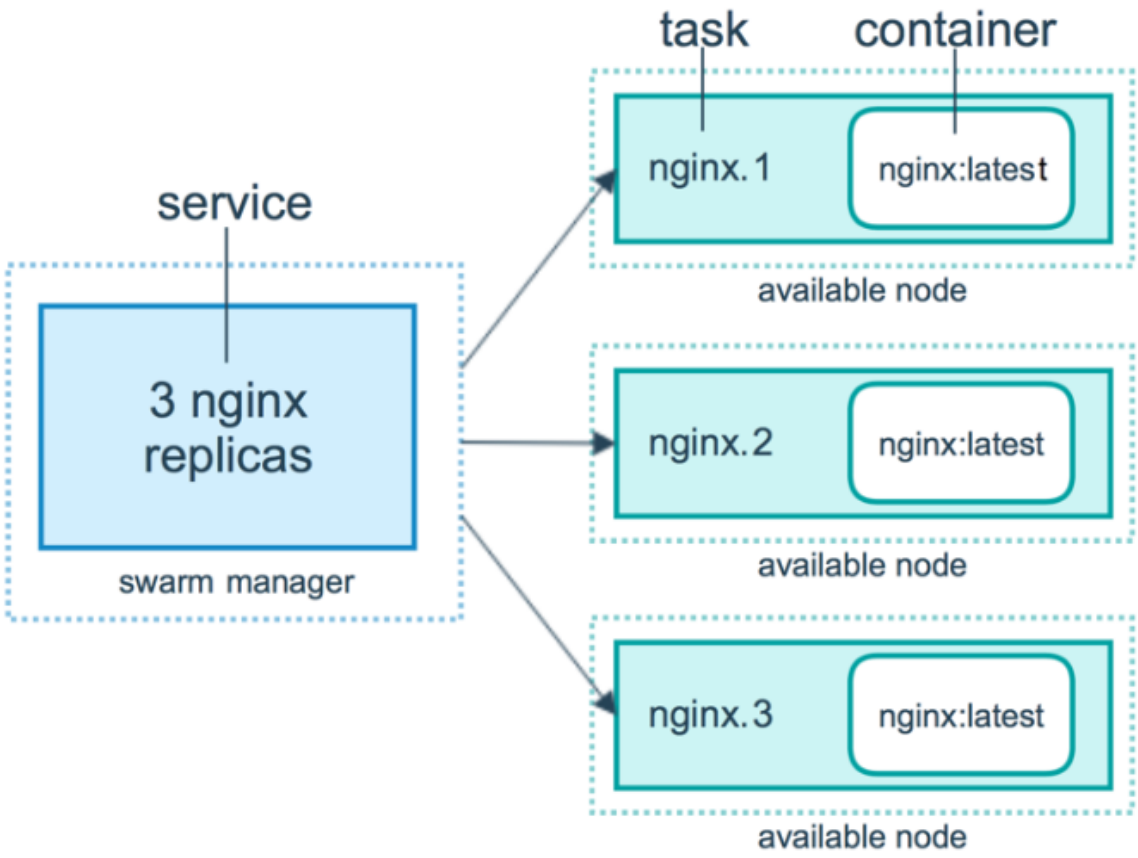
swarm

docker的集群管理软件

https://docs.docker.com/engine/swarm/swarm-tutorial/create-swarm/

https://www.cnblogs.com/zhujingzhi/p/9792432.html





Swarm的调度策略

```
Swarm在调度(scheduler)节点（leader节点）运行容器的时候，会根据指定的策略来计算最适合运行容器的节点，目前支持的策略有：spread, binpack, random.
1）Random
顾名思义，就是随机选择一个Node来运行容器，一般用作调试用，spread和binpack策略会根据各个节点的可用的CPU, RAM以及正在运
行的容器的数量来计算应该运行容器的节点。

2）Spread
在同等条件下，Spread策略会选择运行容器最少的那台节点来运行新的容器，binpack策略会选择运行容器最集中的那台机器来运行新的节点。
使用Spread策略会使得容器会均衡的分布在集群中的各个节点上运行，一旦一个节点挂掉了只会损失少部分的容器。

3）Binpack
Binpack策略最大化的避免容器碎片化，就是说binpack策略尽可能的把还未使用的节点留给需要更大空间的容器运行，尽可能的把容器运行在
一个节点上面。
```

搭建swarm集群

manager节点，下一条在各个worker节点输入

```
[root@manager41 ~]#  docker swarm init --advertise-addr 192.168.0.41
Swarm initialized: current node (jtdc9flz98cpkklx62i3v5xih) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join \
    --token SWMTKN-1-2g7k9v7on5m6qrgcmch5qnrd981sxt7y516j5oj3nf9b325a1z-5emxi4kzwfutlf1zmncn534bl \
    192.168.0.41:2377
```

manager节点

```
[root@manager41 ~]# docker network create -d overlay nginx_net
kfwhc51w1v2qjlda2639r34md
[root@manager41 ~]# docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
8ff436e4a7af        bridge              bridge              local
200acf207a78        host                host                local
ogfs2wt1wjmk        ingress             overlay             swarm
kfwhc51w1v2q        nginx_net           overlay             swarm
036f56f66dfc        none                null                local
[root@manager41 ~]# docker service create --replicas 1 --network nginx_net --name my_nginx -p 80:80 nginx
```

查询到哪个节点正在运行该服务　`docker service ps my_nginx`

动态扩容/缩容（控制集群容器总数）　`docker service scale my_nginx=4`

查看正在运行的服务列表 `docker service ls`

要查看有关服务的详细信息 `docker service inspect --pretty my_nginx`

要以json格式返回服务详细信息，请运行不带`--pretty`

查看群集的当前状态 `docker info`

查看有关节点的信息 `docker node ls`

有哪些方法可以实现宿主机和容器之间传递文件

docker cp
Dockerfile ADD ， COPY

compose

容器编排工具--》批量启动容器的工具。

Compose是用于定义和运行多容器Docker应用程序的工具。通过Compose，您可以使用YAML文件来配置应用程序的服务。然后，使用一个命令，就可以从配置中创建并启动所有服务

官方的文档https://docs.docker.com/compose/gettingstarted/

安装https://docs.docker.com/compose/install/

```
curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose
```

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

新 `volumes` 密钥将主机上的项目目录（当前目录） `/code` 安装到容器内部，使您可以即时修改代码，而不必重建映像。该 `environment` 键设置了 `FLASK_ENV` 环境变量，该变量指示 `flask run` 要在开发模式下运行并在更改时重新加载代码。此模式仅应在开发中使用。

---

service

`Services` 实际上是"containers in production". 一个service只能运行一个image，但是可以运行出同一个image的多个containers。

https://docs.docker.com/engine/reference/commandline/service/

要验证

task对应运行在一个service中的单个container（理论上Task也可以对应到非container的执行单元如进程上，但是目前只对应container）。每个task在service中被分配了一个唯一的数字ID，从1一直到service设置的replicas数。每个Task中有一个对应的container。