比较window subsystem linux（WSL）与docker linux与linux与VMware Linux：

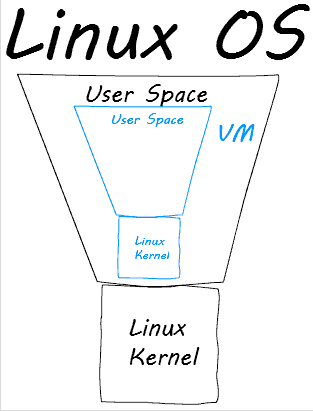
https://www.phoronix.com/scan.php?page=article&item=wsl-february-2018

结论是总体来讲性能与docker差不多，但是I/O上处理能力差了一些。任何虚拟程序本身都是真机中的一个进程。

硬件随着虚拟化技术而不断改变。新的处理器通过纳入高级指令来让来宾操作系统到 hypervisor 的过渡更加高效。此外，硬件也随着输入/输出（I/O）虚拟化而不断改变

当前技术：Mesos用来管理计算机硬件资源（CPU、Mem、Network等），管理的资源分配给K8s用docker进行容器编排创建一批无状态服务。Mesos自身可以构建有状态容器。

容器化：



Docke、Linux Container（LXC）都是都不是完全虚拟化，并且都是使用了linux下的命名空间。Linux系统比较接近的就是内核，内核能够有一些一样的接口，于是LXC只是对内核不进行虚拟，除此之外所有资源都通过命名空间实现隔离与虚拟： <https://www.cnblogs.com/xidongyu/p/5767020.html> ，<https://blog.csdn.net/tongtest/article/details/53440437>。这时候资源的管理通过cgroup（control group），是内核提供的可以限制、记录、隔离进程组所使用的物理资源的机制。命名空间是为了进行隔离或者说是为了能够实现资源共享，cgroup是为了能够进行资源的限制(ulimit程序虽然也支持限制一些资源，据说有人试过使用-m限制一个进程的物理内存，但是限制失败了，最后改用cgroup)。

Imctfy是Google开源版本的容器栈，它提供了用来代替LXC的Linux应用容器

虽然容器通过与真机直通一些东西（这里指的是内核，不是半虚拟化的指令直通）能够得到资源性能使用的提升，但是一旦操作坑爹，就要整个爆炸。

我想过文件系统不使用namespace而使用路径欺骗（一个容器的文件系统对应一个真机文件夹），namespace就是做这样的一件事情。通过上面第二个URL的图片就明白了。

| **类型** | **功能说明** |
| --- | --- |
| MNT Namespace | 提供磁盘挂载点和文件系统的隔离能力 |
| IPC Namespace | 提供进程间通信的隔离能力 |
| Net Namespace | 提供网络隔离能力 |
| UTS Namespace | 提供主机名隔离能力 |
| PID Namespace | 提供进程隔离能力 |
| User Namespace | 提供用户隔离能力 |

cgroup，其本身的作用只是将进程绑定在一个或者多个子系统上，并进行任务跟踪（限制、统计和分离）。但其它系统（比如cpusets，cpuacct），可以利用cgroup的这个功能实现一些新的属性，比如统计或者控制一个cgroup中进程可以访问的资源。

subsystem，子系统，一个通过cgroup提供的工具和接口来管理进程集合的模块。一个子系统就是一个典型的“资源控制器”，用来调度资源或者控制资源使用的上限。其实每种资源就是一个子系统。子系统可以是以进程为单位的任何东西，比如虚拟化子系统、内存子系统。

hierarchy，层级树，多个cgroup的集合，这些集合构成的树叫hierarchy。可以认为这是一个资源树，附着在这上面的进程可以使用的资源上限必须受树上节点（cgroup）的控制。hierarchy上的层次关系通过cgroupfs虚拟文件系统显示。系统允许多个hierarchy同时存在，每个hierachy包含系统中的部分或者全部进程集合。

cgroupfs是用户管理操纵cgroup的主要接口：通过在cgroupfs文件系统中创建目录，实现cgroup的创建；通过向目录下的属性文件写入内容，设置cgroup对资源的控制；向task属性文件写入进程ID，可以将进程绑定到某个cgroup，以此达到控制进程资源使用的目的；也可以列出cgroup包含的进程pid。这些操作影响的是sysfs关联的hierarchy，对其它hierarchy没有影响。

1）mount -t tmpfs cgroup\_root /sys/fs/cgroup      挂载cgroup根文件系统，类型为tmpfs

2）mkdir /sys/fs/cgroup/cpuset     在cgroupfs根目录下创建子cgroup，名为cpuset

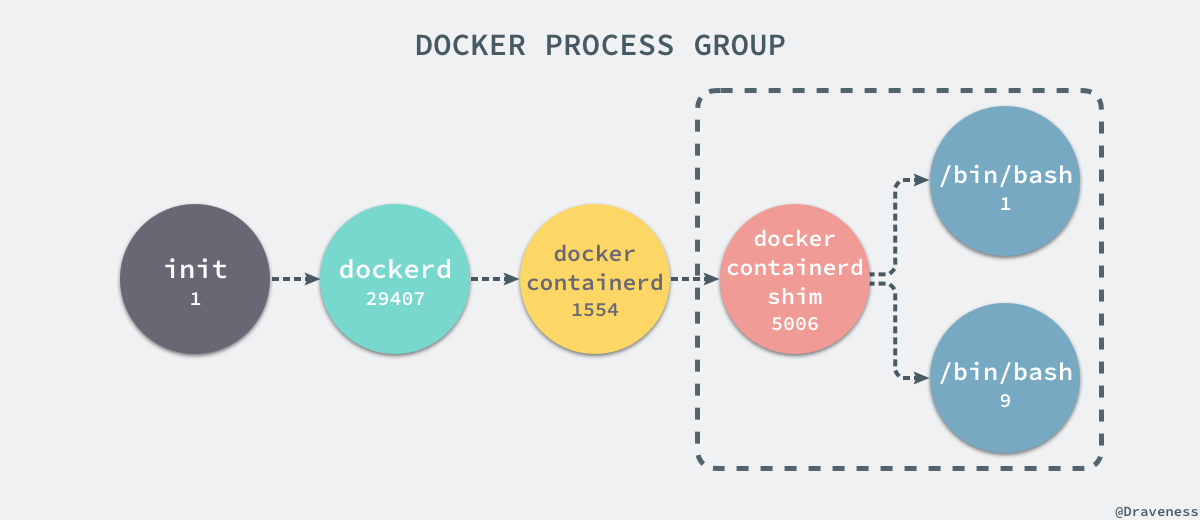
3）mount -t cgroup -o cpuset cpuset /sys/fs/cgroup/cpuset     将名为cpuset的cgroup关联到cpuset子系统

4）在cpuset目录下创建目录，生成一个子cgroup，属性文件中写入相应内容，设置属性。

    cd Charlie     echo 2-3 > cpuset.cpus     echo 1 > cpuset.mems

5）启动需要限制的进程，查找其对应的进程ID，将其写入对应的task文件中

    echo $$> tasks //$$是当前shell的pid



四种网络模式：

* Host模式，即Vmware中桥接模式，与宿主在同一个网络但无独立IP，直接使用宿主机的IP与port。
* Container模式，与其他容器共享Network Namespace，与指定容器共享IP、port范围。
* None模式，关闭网络，容器被放在自己未配置的网络栈中。
* Bridge模式，相当于Vmware的Nat模式，容器使用独立的Network Namespace，并连接到docker0虚拟网卡（默认模式）。通过docker0网桥以及Iptables nat表配置与宿主机通信；bridge模式是Docker默认的网络设置，此模式会为每一个容器分配Network Namespace、设置IP等，并将一个主机上的Docker容器连接到一个虚拟网桥上。

容器的实时迁移：

实时迁移：将运行中的容器迁移到其他的机器上并且不能中断服务

迁移原因：<https://blog.csdn.net/karamos/article/details/80124834>

我想到的办法：（1）先克隆进程与相关数据到目标机器上，在释放本机器上的内容。（2）使用其他的方式维持服务，比如缓存（如果仅仅使用缓存，容易出错，但是很快），比如分布式。 一般实时迁移都是web方面的事情，可以通过改变公司自己的DNS服务器来实现用户请求目标的改变。（3）如果已经实现负载均衡（或者请求引流），这时候可以一个个来移。

虚拟化将应用整合到一台性能剩余的机器上，从而节约一定的成本

windows中来宾账户是限制权限，Linux是无效修改。

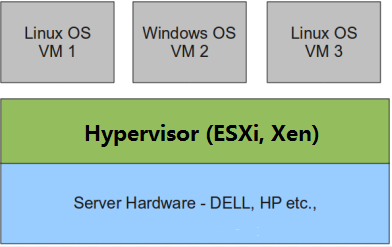
现在容器的文件系统都是用户态文件系统FUSE，因为容器与内核相比，这只是一个外部程序。传统文件系统完全实现在内核态，最终倒是内核维护上的麻烦，现在变成了一个内核的模块（fuse模块）+与内核通信的接口（libfuse）+用户态文件系统（fuse提供了挂载工具fusermount）。 fuse本质上（数据处理时）是处于现有文件系统之上的（具体实现是和现有文件系统处于同一个层次的），fuse不参与底层磁盘数据的存取，只负责处理对读取和写入的数据在逻辑上的操作而已。

一个 24 核的物理机只能虚出 4 台 4 核的 VM

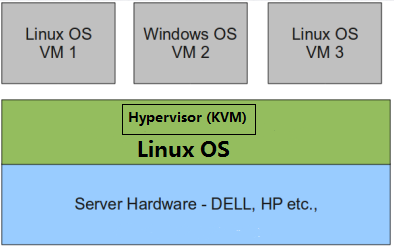
阿里巴巴 曾经不借助Hypervisor，通过cgroup以及定制的clib等实现类似于docker的技术。

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。

1型虚拟化：Hypervisor 直接安装在物理机上，多个虚拟机在 Hypervisor 上运行。Hypervisor 实现方式一般是一个特殊定制的 Linux 系统。Xen 和 VMWare 的 ESXi 都属于这个类型。



2型虚拟化：物理机上首先安装常规的操作系统，比如 Redhat、Ubuntu 和 Windows。Hypervisor 作为 OS 上的一个程序模块运行，并对管理虚拟机进行管理。KVM、VirtualBox 和 VMWare Workstation、Hyper-V 都属于这个类型。



理论上讲：

1型虚拟化一般对硬件虚拟化功能进行了特别优化，性能上比2型要高；

2型虚拟化因为基于普通的操作系统，会比较灵活，比如支持虚拟机嵌套。嵌套意味着可以在KVM虚拟机中再运行KVM。

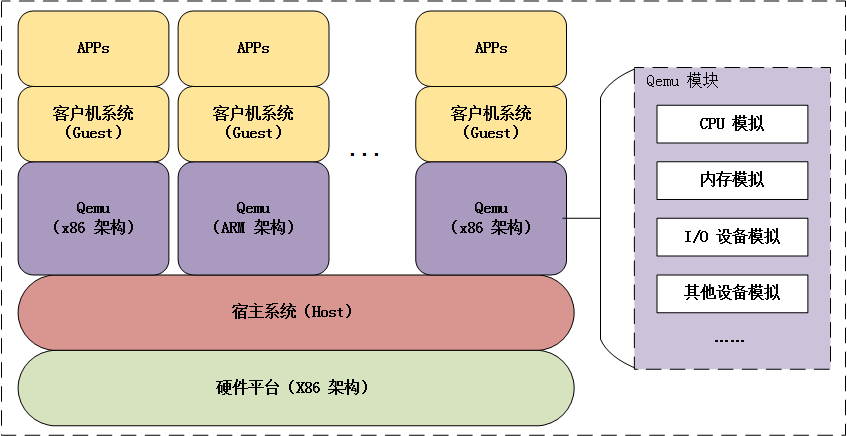
Hyper-V底层的Hypervisor运行在最高的特权级别下，[微软](https://baike.baidu.com/item/%E5%BE%AE%E8%BD%AF)将其称为ring -1（而Intel则将其称为root mode），而[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA)的OS内核和驱动运行在ring 0，[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F)运行在ring 3下，这种架构就不需要采用复杂的BT（二进制[特权指令](https://baike.baidu.com/item/%E7%89%B9%E6%9D%83%E6%8C%87%E4%BB%A4)翻译）技术，可以进一步提高安全性。hyperV上跑linux的图形性能没vmware workstation好，但是windows几乎是物理机水平。

hyper-v建立出来的vhd可以直接mount成虚拟硬盘，也可以直接引导系统变成多启动

LXC 是对大部分操作系统内容都进行了虚拟，共用内核，所以相比较还是比VMware轻量。LXC本身操作起来不是很友好，为了向下兼容默认的一些机制都没有开启，而且只有一套操作工具没有daemon进程，即没有办法提供REST api。于是有了升级后的LXD。他们的文件系统是LXCFS。

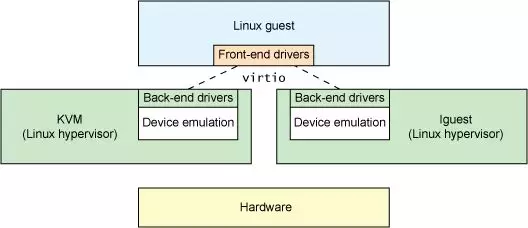
docker只能在linux系统下运行，Windows下运行必要要支持Hyper-v( 注：windows专业版本才有这个功能)。安装docker的时候会自动创建一个虚拟的linux系统，后续的其他操作实际上是间接使用这个虚拟系统进行的。

QEMU： 纯软件实现的虚拟化模拟器

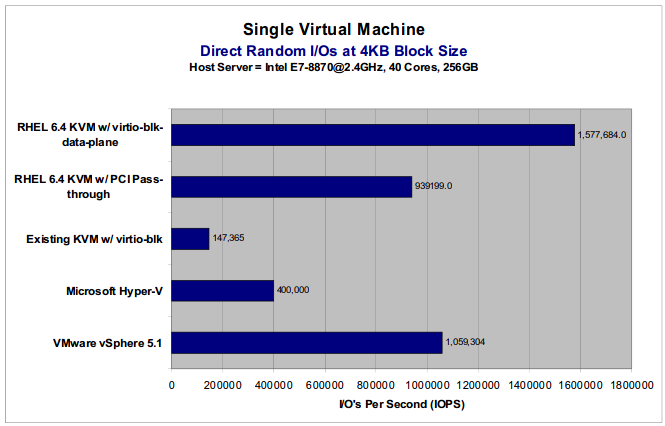


QEMU因为是纯软件实现，所以如果仅仅使用它的话性能并不是很好，一般会配合KVM使用，因为 KVM 是硬件辅助的虚拟化技术，主要负责 比较繁琐的 CPU 和内存虚拟化，而 Qemu 则负责 I/O 虚拟化，两者合作各自发挥自身的优势，相得益彰。闭源的KQemu 在性能上提升不少，百度百科中说接近真机。

virtio 是解决虚拟机前后端通信的方案。由于不同 guest 前端设备其工作逻辑大同小异（如块设备、网络设备、PCI设备、balloon驱动等），单独为每个设备定义一套接口实属没有必要，而且还要考虑扩平台的兼容性问题，另外，不同后端 Hypervisor 的实现方式也大同小异（如KVM、Xen等），这个时候，就需要一套通用框架和标准接口（协议）来完成两者之间的交互过程，virtio 就是这样一套标准，它极大地解决了这些不通用的问题。



从下面的图中看出KVM使用virtio的IO性能比直通还要好（应该因为缓冲区的实现）



virtio按照前后端的思想，在KVM-QEMU中，客户机为前端，QEMU为后端，中间是virtio的两层： 通过批处理思想，减少QEMU的工作量，提升IO性能



KVM，Xen, VMware vSphere，Huawei FusionSphere 都是半虚拟化。半虚拟化通过底层硬件辅助的方式，将部分没必要虚拟化的指令通过硬件来完成。KVM要求CPU必须支持硬件虚拟化，Intel即VT-x，AMD即AMD-V。kvm和VMware vSphere esxi在直通方面灵活性很高优势很大。

KVM（Kernel-based Virtual Machine）有一个内核模块叫 kvm.ko，只用于管理虚拟 CPU 和内存。IO 外设的任务交给 Linux 内核和 Qemu。

RemoteFX可以将[GPU](https://baike.baidu.com/item/GPU)虚拟化提供给每一个[虚拟桌面](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%A1%8C%E9%9D%A2)，也就是说每个[虚拟机](https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E6%9C%BA/104440)都可以获得独立的虚拟GPU。hyperV通常是不会把N/A卡直接用于remotefx的，这是N/A两家的问题，N家只有grid支持gpu虚拟化。虚拟化——资源隔离，可以并行。

VMM 虚拟机监控器，当虚拟机中发生“陷阱”的时候，就是它捕获。

虚拟机可能会对各方面性能都降低，因为是对所有硬件进行虚拟。

半虚拟化：在全虚拟化的基础上，把客户操作系统进行了修改，增加了一个专门的API，这个API可以将客户操作系统发出的指令进行最优化，即不需要Hypervisor耗费一定的资源进行翻译操作，因此Hypervisor的工作负担变得非常的小

**目标：**

Docker 关注于短期的、无状态的、最小化的容器，这些容器通常并不会升级或者重新配置，而是作为一个整体被替换掉。这就使得 Docker 及类似项目更像是一种软件发布机制，而不是一个机器管理工具。docker是软件的快速部署、复杂软件跨平台的方式，一般的小软件应该通过特殊的打包方式比如flatpak、AppImage等。

**安装相关：**

docker-io 是以前早期的版本，版本号是 1.\*，最新版是 1.13，而 docker-ce 是新的版本，分为社区版 docker-ce 和企业版 docker-ee，版本号是 17.\* ，最新版是 17.12。

深度基于 sid 版本开发，通过 $(lsb\_release -cs) 获取的版本信息为 unstable，而docker官方源并没提供 sid 这种unstable版本的docker，所以使用官方教程是安装不成功的。

具体：https://bbs.deepin.org/forum.php?mod=viewthread&tid=139514&highlight=%E5%AE%89%E8%A3%85docker

我使用：wget -qO- https://get.docker.com/ | sh

cat /etc/debian\_version 可以知道deepin是基于哪一个debian版本。

容器只运行一个进程（HTTP 守护进程、应用程序服务器、数据库）时效果最佳，但如果运行一个以上进程，在管理和检索日志以及单独更新进程时就会遇到很多麻烦。不过实际上一个docker容器扮演的是一个单独的服务器角色，比如一个节点上会安装spark与zookeeper。

配置文件中镜像存储位置设置：（也可以通过自动挂载来改变镜像存储位置）（默认位置/var/lib/docker）

在 Ubuntu/Debian 系统下:

/etc/default/docker 中 DOCKER\_OPTS="-dns 8.8.8.8 -dns 8.8.4.4 -g /mnt"

在 Fedora/Centos 系统下:

/etc/sysconfig/docker中 other\_args="-g /mnt"

If you would like to use Docker as a non-root user, you should now consider

adding your user to the "docker" group with something like: sudo usermod -aG docker vega

**安装原因：**

污染了本机的开发环境。比如编译安装内核的时候需要安装编译过程中的依赖，但是这些依赖只是过程中需要。

软件一般安装在/下，但是如果使用docker可以将软件及需要的依赖包装存放到其他地方。减少了我一键备份的体积。比如数据库、服务器软件。kali的镜像。

公司使用Docker，是因为Dockerfile或者docker pull就是环境的一键安装包，而且几乎没有性能损耗。

**原理相关：**

主要基于Go语言编写

如果是在 docker 里面的话 ，关闭了操作系统层面的地址随机化保护 ， 物理机也会关闭 ， 感觉还是比较危险的

运行docker run命令时，Docker会启动一个进程，并为这个进程分配其独占的文件系统、网络资源和以此进程为根进程的进程组（进程树）。处于几个单独的Linux资源 命名空间中。

shell中打出docker，使用tab按键或者执行docker命令的时候都会触发docker daemon的启动，deepin右上角存在消息提示。

镜像缓存机制: docker build过程中每执行一步RUN命令，就生成一个镜像，另外镜像之间有血缘关系，是否需要重新编译两个小时取决于你的修改影响的第一个镜像有多早。简单理解当你修改dockerfile时,当你修改了第N行的RUN命令, N+1之后的命令都要重跑.docker image --tree #查看镜像的依赖关系,名字为none的便是每一步执行了RUN的缓存。对于ADD和COPY命令而言，还需要检查文件的校验和（不考虑文件的最后访问时间与修改时间）。

只读层与读写层： image的所有层都是默认只读的,当你启动一个image当做container运行的时候,docker会在image的只读层上加一层薄薄的可写层.你在container里面做的所有操作都是反映在可写层，使用commit命令会再次产生一层叠在上面。docker和没有persistence分区的live系统很像。docker使用的storage driver有https://blog.csdn.net/vchy\_zhao/article/details/70238690。

docker并不是足够安全的。 Docker命令没有内置的审计和日志功能，但是sudo有。

如果网卡切换了wifi，那么需要关闭容器重新systemctl restart docker才能让容器有网

国内加速：

https://www.daocloud.io/mirror#accelerator-doc

有可能下到一半就0Kb了，并且重新下载与proxychains4都不能解决这个问题

缩小镜像：

当然对于公司，功能型与易用型才是重点。对于我的个人笔记本要考虑自己的能力以及磁盘的存储空间。

开发 Dockerfile 时分别 RUN 每条命令，目的是分层减少重新 build 的时间，开发好了合并 RUN 为一条，减少分层。

Alpine Linux 是一个社区开发的面向安全应用的轻量级 Linux 发行版。基于这种Linux的Docker会小上很多。使用alpine,我构建的symfony＋nginx＋php-fpm的开发环境，只需要60MB。这个在ubuntu的image上，你会不知不觉update到大概600MB左右。alpine 最蛋疼的地方是太精简了 连 tzdata 都没，想要用 TZ env 设时区。alpine使用的是musl libc而其他发行版是glibc。推荐alpine，golang 可以用 scratch。而一般的发行版中debian比较轻量。

在 Dockerfile 中， 每一条指令都会创建一个镜像层因为commit，继而会增加整体镜像的大小,注意是镜像不是容器，容器stop之后不会丢失。如果通过&&将指令一条条连接起来，减少镜像层的创建从而减小体积。如果是创建文件100M的指令，比如RUN dd if=/dev/zero of=/tmp/foo/bar bs=1048576 count=100，即便后来将文件删除了（不在同一层中操作），docker的大小就还是增大了100M。推荐对于一些需要通过 COPY 命令的方式拷贝到镜像里面的文件，可以使用 wget xx -O xxx(或者不使用-O参数,那就要配合cd或者mv来确保文件的位置)(curl -fSL xx -o xx) 的方式，用完之后在同一层中删除

删除不必要的内容：编译后的依赖，/var/cache（apt clean && apt clean all），/var/tmp，/tmp

Alpine中先apk add --virtual .build-deps将本次安装的所有包封装成一个名为.build-deps的虚拟包。这样做的好处是可以通过apk del .build-deps一键清除这些包

压缩镜像：使用docker自带的命令 export 和 import来压缩会导致会丢失镜像原有的一些信息，比如：导出端口，环境变量，默认指令。推荐使用https://github.com/jwilder/docker-squash ，但是很可能最终的效果不是很大。

mysql 5.6 298MB

Mariadb xxxxxx 198MB

redis 4.0-alpine 27.5MB

php 5.6.31-fpm-alpine 53.9MB

nginx 1.12-alpine 15.5MB

我已经将使用的Dockerfile上传到自己的github/docker

使用别人的Dockerfile： https://docs.docker.com/samples/library https://github.com/docker-library

nginx:https://github.com/nginxinc/docker-nginx/blob/master/stable/alpine

tomcat:Apache Tomcat 8.5.x 旨在替换 8.0.x，并包含从 9.0.x 分支中提取的新特性。

php:需要增加pdo\_mysql，mysqli

hadoop:https://github.com/kiwenlau/hadoop-cluster-docker

不追求最小化可以直接使用：阿里云镜像、网易蜂巢等

构建一个php7-alpine： https://segmentfault.com/a/1190000011876870

即便是MAINTAINER指令也会有一层

创建镜像：

tar命令配合一个xxbootxxx可以以当前环境为主创建镜像，但是不如Dockerfile

连接容器：

同一个宿主机上的多个docker容器之间如果想进行通信，可以通过使用容器的ip地址来通信，也可以通过宿主机的ip加上容器暴露出的端口号来通信，前者会导致ip地址的硬编码，不方便迁移，并且容器重启后ip地址会改变，除非使用静态ip，后者的通信方式比较单一，只能依靠监听在暴露出的端口的进程来进行有限的通信。通过docker的link机制可以通过一个name来和另一个容器通信，link机制方便了容器去发现其它的容器并且可以安全的传递一些连接信息给其它的容器。所以容器间连接推荐使用的是--link参数，而不是通过IP，也不是通过-p映射端口

例子：已经有一个nginx容器，并配置期配置文件php为php-fpm:9000。有一个php-fpm容器，并删除配置文件中监听端口前的127.0.0.1。

sudo docker run -itd --name php-fpm -d <php-fpm-images-name> --name与-d选项顺序不能反

sudo docker run --name nginx -p 80:80 -v /path/nginx/www:/var/www -d --link php-fpm:php-fpm <nginx-images-name>

例子：https://www.jianshu.com/p/fcd0e542a6b2

Cloudinsight 是可以用于docker的一种系统监测工具

会创建一个虚拟网卡，为docker container提供网络

命令:

IMAGEID <=> REPOSITORY:TAG 不过多个REPOSITORY:TAG可能对应一个TAG

latest tag其实在使用中不是最新的意思，而是默认值(defalut)的意思。也就是说，如果在tag为可选的命令中，我们没有写上tag，如 docker pull entel\_zmc\_images:zmc\_base vs docker pull entel\_zmc\_images ，前者有确定的tag，而后者没有，这时系统会自动添加一个:latest标签，然后去匹配。

sudo docker info 查看完整的docker信息

sudo docker build -t nginx ./ 按照当期目录下的Dockerfile创建一个nginx:latest镜像，dockerfile创建方式其实就是创建一个容器以后不停commit来改变镜像

创建的container hostname是随机的，这一点可以通过hostname命令或者查看/etc/hosts文件，如果有需求固定成为自己指定的hostname，例如搭建hadoop环境的时候，应该使用--hostname=xxx。

你可以运行下面的命令来启动docker：

start首先运行下面的命令确保 Docker 服务运行正常：

sudo docker ps -a

sudo docker stats -a 持续性查看ps(没有-a就会只显示正在运行的container)

docker logs xxxx 查看xxx的标准输出

下载以后使用一下命令找出本地想要的docker镜像ID

docker images

docker images ls -a看到一堆<none>:<none>中间镜像，他们不会造成硬盘空间占用的问题（因为这是镜像的父层，必须存在的），这些中间镜像不能被删除

sudo docker tag 466a0faa305d php-fpm:7.2

为指定image\_id镜像增加一个tag

docker run --help 可以看到对多种cpu、memory的限制选项，基于cgroup

docker run -i -t 镜像ID/bin/bash 然后就会创建一个新的容器并进入可以交互的tty

-d 后台运行docker容器并打印容器ID。如果不加-d参数，那么容器运行会和终端绑定，如果终端关闭，那么容器也会关闭，但是容器不会被删除。

-p 80:80 docker容器和外侧的端口映射

-v 指定一个path进行持久化存储<=>Dockerfile中的VOLUME

sudo docker run -i -t alpine:3.7 /bin/sh

sudo docker run --name alpine-spark -it alpine:3.7 /bin/sh #正确

sudo docker run -itd --name alpine-spark -d alpine:3.7 #如果没有-itd，能创建容器，但是进不去

sudo docker run --name nginx -p 80:80 -v $PWD/www:/var/www -d nginx

sudo docker inspect -f {{.Volumes}} test

--env run的时候提供环境变量

--volume /path/flume/node2.conf:/opt/flume-config/flume.conf 将外部目录映射到docker容器内部（冒号前面是外部）

sudo docker attach <container\_id> 进入容器

vega@vega-Laptop:/path/nginx$ sudo docker start f0d45af2262e

f0d45af2262e

vega@vega-Laptop:/path/nginx$ sudo docker attach f0d45af2262e

/ # exit

vega@vega-Laptop:/path/nginx$ sudo docker attach f0d45af2262e

You cannot attach to a stopped container, start it first

上面相当于直接sudo docker start f0d45af2262e -ai

sudo docker exec -it <container\_id or container\_name> /bin/sh 不要用attach进入，使用此命令来进入就不会因为exit而关闭容器运行线程

docker ps 在容器外面执行可以验证所进入的容器镜像

sudo docker container ls -a

docker stop <container\_id> 退出当前container再尝试删除.

docker rm <container\_id>删除容器

sudo docker rm $(sudo docker container ls -aq) 删除所有容器

docker rmi <image name>删除镜像。这是使用的是image name。删除镜像前需要删除掉相关container，如果直接-f会删除镜像与相关容器。

sudo docker rmi -f $(sudo docker images -aq) 删除所有镜像

删除latest tag:用rmi REPOSITORY:TAG的办法来指定删除即可，latest即默认TAG

vega@vega-Laptop:/path/nginx/git$ sudo docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

alpine 3.7 3fd9065eaf02 3 months ago 4.15MB

alpine latest 3fd9065eaf02 3 months ago 4.15MB

vega@vega-Laptop:/path/nginx/git$ sudo docker rmi alpine

Untagged: alpine:latest

vega@vega-Laptop:/path/nginx/git$ sudo docker images

REPOSITORY TAG IMAGE ID CREATED SIZE

alpine 3.7 3fd9065eaf02 3 months ago 4.15MB

docker cp <containerId>:/file/path/within/container /host/path/target 从Docker容器内拷贝文件到主机上

docker commit alpine-spark alpine-test:0.0 #如果没有修改，则等效于docker tag

docker stop $(docker ps -a -q) #关闭所有容器

docker export 7691a814370e > ubuntu.tar

docker import xxx.tar test/ubuntu:v1.0

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

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

export是将容器导出为一个tar包，checkpoint还处于试验状态，感觉两者都算是vmware里面的快照，但是checkpoint肯定是不能移植的，因为docker是一层一层，设置checkpoint可以很简单，直接去掉后面增加的几层就好。这么说export接近vmware中的快照，checkpoint倒是docker中的快照。

docker export 通过--help可以查看到作用：a container's filesystem as a tar archive

vega@vega-Laptop:~$ docker checkpoint

docker checkpoint is only supported on a Docker daemon with experimental features enabled

创建一个桥接网段： docker network create --driver=bridge hadoop

之后run的时候 ：（我将docker run的一堆命令写在shell脚本中）

docker run -itd \

--net=hadoop \

--ip=172.18.0.3 \

--name hadoop-slave1 \

--hostname hadoop-slave1 \

sebge2/hadoop-slave &> /dev/null

Dockerfile的编写：见php-fpm的dockerfile

Kubernetes（k8s）是自动化容器操作的开源平台，这些操作包括部署，调度和节点集群间扩展。分布式架构方案。

docker-compose ,一个使用容器的架构或者应用可能需要多个容器（比如hadoop的多个节点）组成，使用dockerfile配置不同的容器，docker-compose.yml最后组合在一起。