[Edit](http://maxiang.info/" \l "/?provider=evernote&guid=1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a&notebook=%E4%BA%91%E8%AE%A1%E7%AE%97)

# docker

云计算

[docker](https://www.docker.com/)

* [docker](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#docker)
* [docker简介](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#docker简介)
* [由paas 到container](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#由paas-到container)
* [docker产生的目的](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#docker产生的目的)
* [2. 比较](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#2-比较)
* [2.1 docker vs 传统虚拟化技术](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#21-docker-vs-传统虚拟化技术)
* [2.2 docker vs lxc](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#22-docker-vs-lxc)
* [container vs VMs](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#container-vs-vms)
* [docker应用场景](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#docker应用场景)
* [docker 核心技术预览](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#docker-核心技术预览)
* [隔离性：linux namespace (ns)](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#隔离性linux-namespace-ns)
* [docker 命令](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#docker-命令)
* [docker 安装](https://app.yinxiang.com/note/1a7c4c06-fe9a-4f87-a6c7-fd46d2fdc63a?usn=4968#docker-安装)

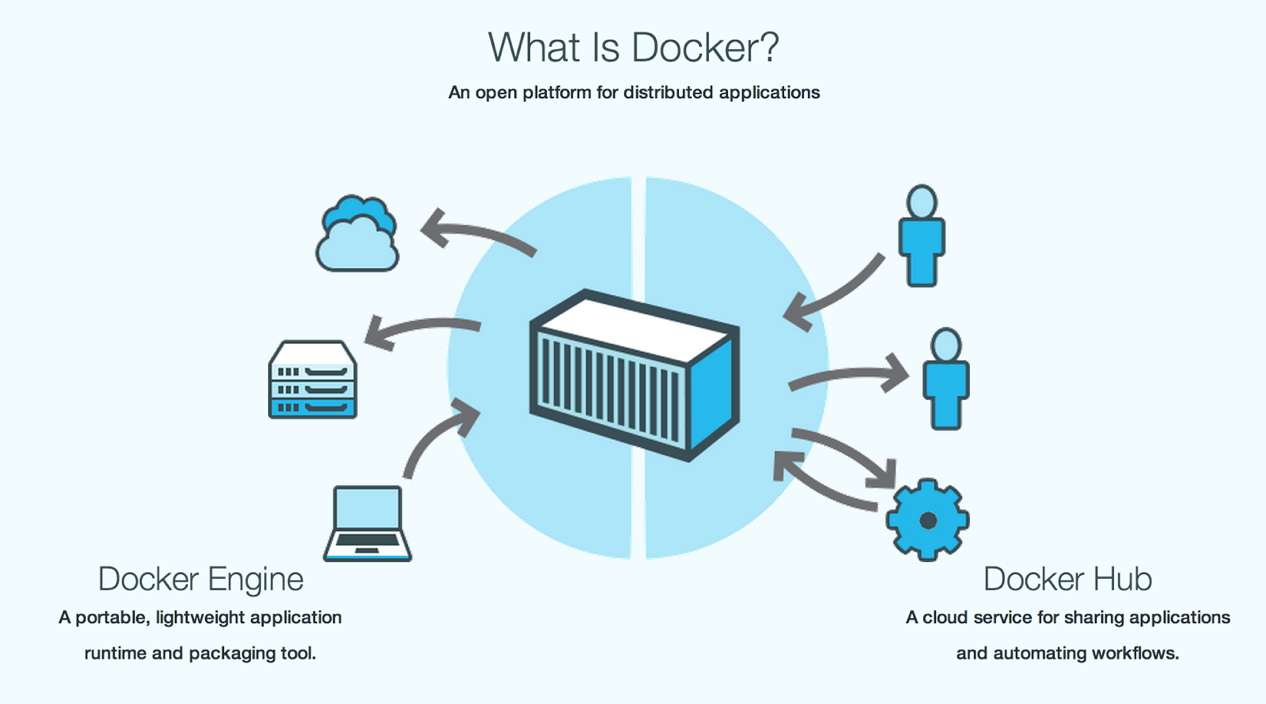
## docker简介

Docker is an open-source engine that automates the deployment of any application as a lightweight, portable, self-sufficient container that will run virtually anywhere.

[Docker](https://www.docker.com/)是 PaaS 提供商[dotCloud](https://www.dotcloud.com/)开源的一个基于 LXC 的高级容器引擎， [源代码](https://github.com/docker/docker)托管在 Github 上, 基于go语言并遵从Apache2.0协议开源。Docker近期非常火热，无论是从 GitHub 上的代码活跃度，还是Redhat宣布在[RHEL7中正式支持Docker](http://server.cnw.com.cn/server-os/htm2014/20140616_303249.shtml)，都给业界一个信号，这是一项创新型的技术解决方案。就连 Google 公司的 Compute Engine 也支持 docker 在其之上运行，国内“BAT”先锋企业百度Baidu App Engine(BAE)平台也是以[Docker作为其PaaS云基础](http://blog.docker.com/2013/12/baidu-using-docker-for-its-paas/)。

### 由paas 到container

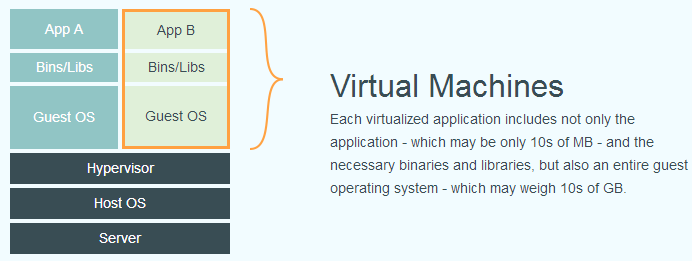
2013年2月，前Gluster的CEO Ben Golub和dotCloud的CEO Solomon Hykes坐在一起聊天时，Solomon谈到想把dotCloud内部使用的Container容器技术单独拿出来开源，然后围绕这个技术开一家新公司提供技术支持。28岁的Solomon在使用python开发dotCloud的PaaS云时发现，使用 LXC(Linux Container) 技术可以打破产品发布过程中应用开发工程师和系统工程师两者之间无法轻松协作发布产品的难题。这个Container容器技术可以把开发者从日常部署应用的繁杂工作中解脱出来，让开发者能专心写好程序；从系统工程师的角度来看也是一样，他们迫切需要从各种混乱的部署文档中解脱出来，让系统工程师专注在应用的水平扩展、稳定发布的解决方案上。他们越深入交谈，越觉得这是一次云技术的变革，紧接着在2013年3月Docker 0.1发布，拉开了基于云计算平台发布产品方式的变革序幕。

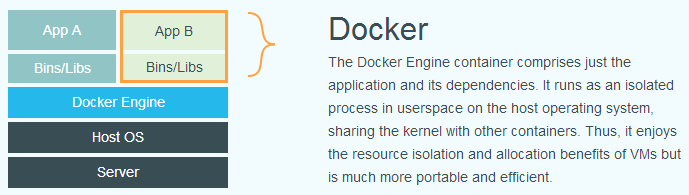


### docker产生的目的

Docker产生的目的就是为了解决以下问题：

1. 环境管理复杂：从各种OS到各种中间件再到各种App，一款产品能够成功发布，作为开发者需要关心的东西太多，且难于管理，这个问题在软件行业中普遍存在并需要直接面对。Docker可以简化部署多种应用实例工作，比如Web应用、后台应用、数据库应用、大数据应用比如Hadoop集群、消息队列等等都可以打包成一个Image部署。
2. 云计算时代的到来：AWS的成功，引导开发者将应用转移到云上, 解决了硬件管理的问题，然而软件配置和管理相关的问题依然存在 (AWS cloudformation是这个方向的业界标准, 样例模板可参考这里)。Docker的出现正好能帮助软件开发者开阔思路，尝试新的软件管理方法来解决这个问题。
3. 虚拟化手段的变化：云时代采用标配硬件来降低成本，采用虚拟化手段来满足用户按需分配的资源需求以及保证可用性和隔离性。然而无论是KVM还是Xen，在 Docker 看来都在浪费资源，因为用户需要的是高效运行环境而非OS，GuestOS既浪费资源又难于管理，更加轻量级的LXC更加灵活和快速。





1. LXC的便携性：LXC在 Linux 2.6 的 Kernel 里就已经存在了，但是其设计之初并非为云计算考虑的，缺少标准化的描述手段和容器的可便携性，决定其构建出的环境难于分发和标准化管理(相对于KVM之类image和snapshot的概念)。Docker就在这个问题上做出了实质性的创新方法。

Docker的主要特性如下：

* 文件系统隔离：每个进程容器运行在完全独立的根文件系统里。
* 资源隔离：可以使用cgroup为每个进程容器分配不同的系统资源，例如CPU和内存。
* 网络隔离：每个进程容器运行在自己的网络命名空间里，拥有自己的虚拟接口和IP地址。
* 写时复制：采用写时复制方式创建根文件系统，这让部署变得极其快捷，并且节省内存和硬盘空间。
* 日志记录：Docker将会收集和记录每个进程容器的标准流（stdout/stderr/stdin），用于实时检索或批量检索。
* 变更管理：容器文件系统的变更可以提交到新的映像中，并可重复使用以创建更多的容器。无需使用模板或手动配置。
* 交互式Shell：Docker可以分配一个虚拟终端并关联到任何容器的标准输入上，例如运行一个一次性交互shell。

## 2. 比较

### 2.1 docker vs 传统虚拟化技术

作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式（xen、kvm、vmware）相比具有众多的优势。

首先，Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多。 其次，Docker 对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器。容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行 10 个不同的应用就要起 10 个虚拟机，而Docker 只需要启动 10 个隔离的应用即可。

具体说来，Docker 在如下几个方面具有较大的优势。

1. 更快速的交付和部署   
   对开发和运维（devop）人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。   
   开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。 Docker 可以快速创建容器，快速迭代应用程序，并让整个过程全程可见，使团队中的其他成员更容易理解应用程序是如何创建和工作的。 Docker 容器很轻很快！容器的启动时间是秒级的，大量地节约开发、测试、部署的时间。
2. 更高效的虚拟化   
   Docker 容器的运行不需要额外的 hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效率。
3. 更轻松的迁移和扩展   
   Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。 这种兼容性可以让用户把一个应用程序从一个平台直接迁移到另外一个。
4. 更简单的管理   
   使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

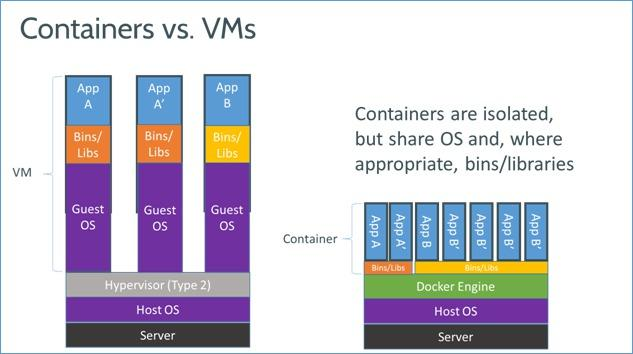
|  |  |  |
| --- | --- | --- |
| **特性** | **容器** | **虚拟机** |
| 启动 | 秒级 | 分钟级 |
| 硬盘使用 | 一般为 MB | 一般为 GB |
| 性能 | 接近原生 | 弱于 |
| 系统支持量 | 单机支持上千个容器 | 一般几十个 |

### 2.2 docker vs lxc

Docker以Linux容器LXC为基础，实现轻量级的操作系统虚拟化解决方案。在LXC的基础上Docker进行了进一步的封装，让用户不需要去关心容器的管理，使得操作更为简便，具体改进有：

1. Portable deployment across machines   
   Docker提供了一种可移植的配置标准化机制，允许你一致性地在不同的机器上运行同一个Container；而LXC本身可能因为不同机器的不同配置而无法方便地移植运行；
2. Application-centric   
   Docker以App为中心，为应用的部署做了很多优化，而LXC的帮助脚本主要是聚焦于如何机器启动地更快和耗更少的内存；
3. Automatic build   
   Docker为App提供了一种自动化构建机制（Dockerfile），包括打包，基础设施依赖管理和安装等等；
4. Versioning   
   Docker提供了一种类似git的Container版本化的机制，允许你对你创建过的容器进行版本管理，依靠这种机制，你还可以下载别人创建的Container，甚至像git那样进行合并；
5. Component reuse   
   Docker Container是可重用的，依赖于版本化机制，你很容易重用别人的Container（叫Image），作为基础版本进行扩展；
6. Sharing   
   Docker Container是可共享的，有点类似github一样，Docker有自己的INDEX，你可以创建自己的Docker用户并上传和下载Docker Image；
7. Tool ecosystem   
   Docker提供了很多的工具链，形成了一个生态系统；这些工具的目标是自动化、个性化和集成化，包括对PAAS平台的支持等。

### container vs VMs



## docker应用场景

Docker作为一个开源的应用容器引擎，让开发者可以打包他们的应用以及依赖包到一个可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。Docker可以自动化打包和部署任何应用、创建一个轻量级私有PaaS云、搭建开发测试环境、部署可扩展的Web应用等。这决定了它在企业中的应用场景是有限的，Docker将自己定位为“分发应用的开放平台”，其网站上也明确地提到了Docker的典型应用场景：

Automating the packaging and deployment of applications

Creation of lightweight, private PAAS environments

Automated testing and continuous integration/deployment

Deploying and scaling web apps, databases and backend services

对应用进行自动打包和部署，创建轻量、私有的PAAS环境，自动化测试和持续整合与部署，部署和扩展Web应用、数据库和后端服务。

平台即服务一般与大数据量系统同在，反观当前我司各IT系统，可以在以下情形下使用docker替代方案：

结合vagrant或supervisor，搭建统一的开发、测试环境

多个开发人员共同进行一个项目，就必须保持开发环境完全一致，部署到测试环境、正式环境后，最好都是同一套环境，通过容器来保存状态，分发给开发人员或部署，可以让“代码在我机子上运行没有问题”这种说辞将成为历史。

对memcached、mysql甚至tomcat，打包成一个个容器，避免重复配置

比如将一个稳定版本的、已配置完善的mysql，固化在一个镜像中，假如有新的环境要用到mysql数据库，便不需要重新安装、配置，而只需要启动一个容器瞬间完成。tomcat应用场景更多，可以将不同版本的jvm和tomcat打包分发，应用于多tomcat集群，或在测试服务器上隔离多个不同运行环境要求的测试应用（例如旧系统采用的是jdk6，新系统在jdk7上开发，但共用同一套测试环境）

## docker 核心技术预览

docker基本概念

* 镜像(image)：镜像就是一个只读模板
* 容器(container)：利用容器来运行应用，容器是从镜像创建的运行实例，它可以启动，开始停止，删除，每个容器都是相互隔离的，保证安全的平台(容器在启动的时候创建一层可写层作为最上层)
* 仓库(repository)存放镜像文件的场所，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）。   
  公开仓库的[docker hub](https://hub.docker.com/), [docker pool](http://www.dockerpool.com/)

docker 核心是一个操作系统级虚拟化方法，理解起来可能并不像VM那样直观，我们从虚拟化方法的四个方面： **隔离性， 可配额/可度量，便携性，安全性**来详细介绍docker的技术细节

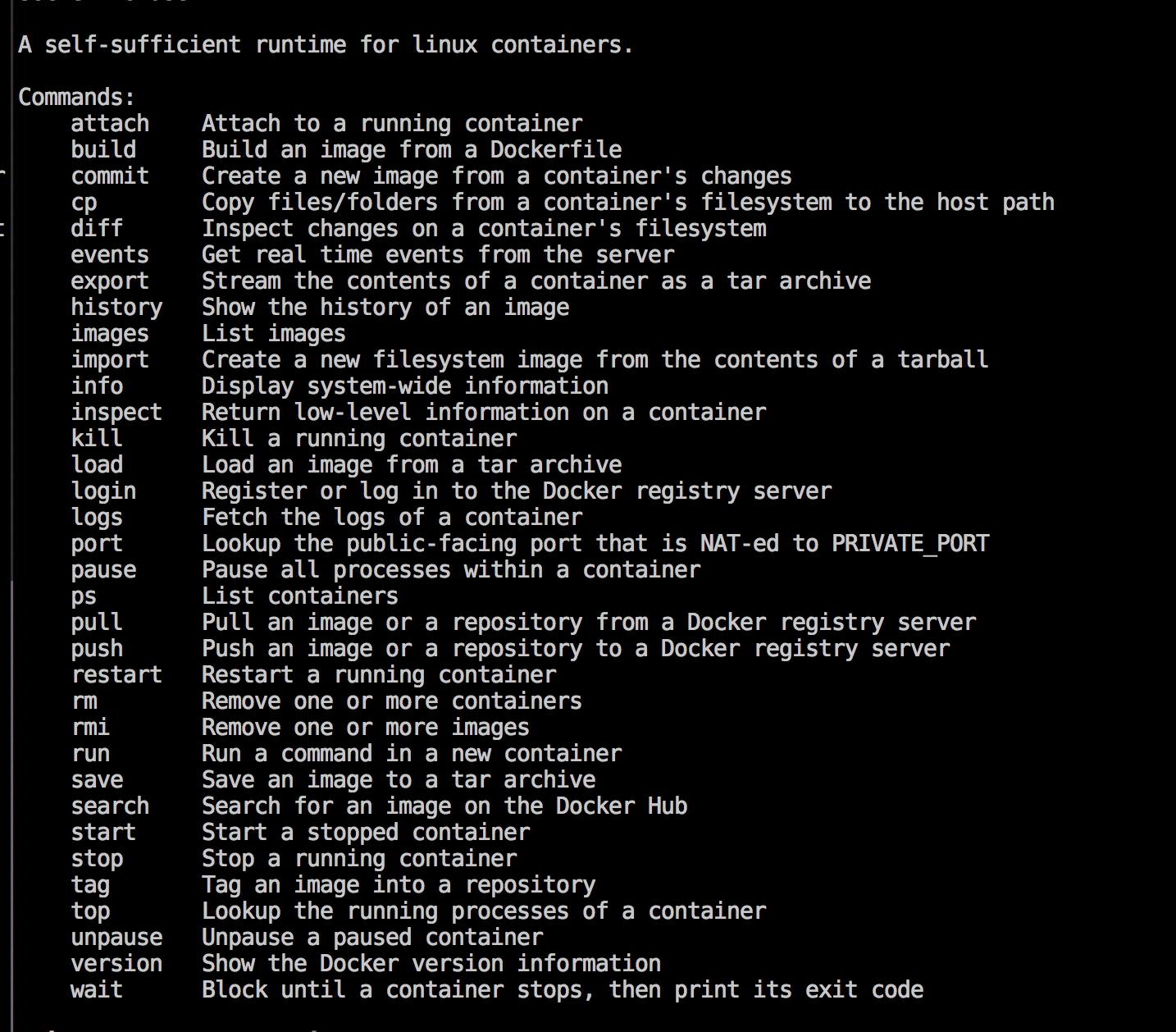
### 隔离性：linux namespace (ns)

每个用户实例之间相互隔离, 互不影响。 一般的硬件虚拟化方法给出的方法是VM，而LXC给出的方法是container，更细一点讲就是kernel namespace。其中pid、net、ipc、mnt、uts、user等namespace将container的进程、网络、消息、文件系统、UTS(“UNIX Time-sharing System”)和用户空间隔离开。

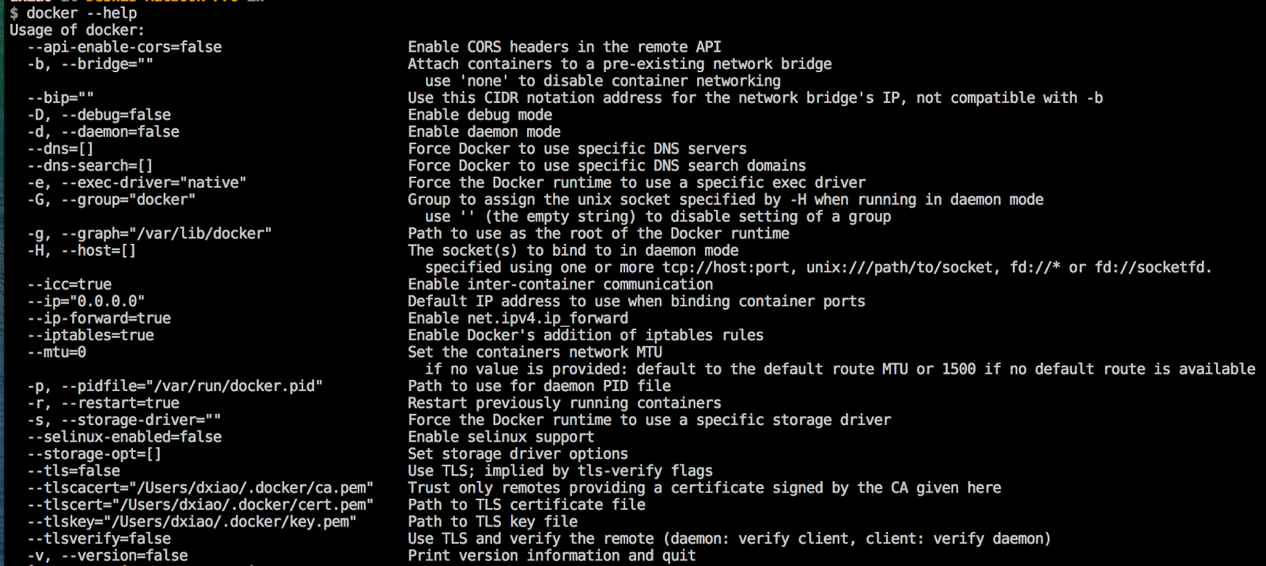
1. pid namespace   
   不同用户的进程就是通过pid numespace隔离开的，且不同的namespace中可以有相同pid，所有的lxc进程在docker中的父进程为docker的进程，每个lxc进程具有不同的namespace。同时由于允许嵌套，因此可以很方便的实现 Docker in Docker。
2. ipc namespace   
   container中进程交互还是采用linux常见的进程间交互方法(interprocess communication - IPC), 包括常见的信号量、消息队列和共享内存。然而同 VM 不同的是，container 的进程间交互实际上还是host上具有相同pid namespace中的进程间交互，因此需要在IPC资源申请时加入namespace信息 - 每个IPC资源有一个唯一的 32 位 ID。
3. net namespace   
   有了 pid namespace, 每个namespace中的pid能够相互隔离，但是网络端口还是共享host的端口。网络隔离是通过net namespace实现的， 每个net namespace有独立的 network devices, IP addresses, IP routing tables, /proc/net 目录。这样每个container的网络就能隔离开来。docker默认采用veth的方式将container中的虚拟网卡同host上的一个docker bridge: docker0连接在一起。
4. mnt namespace   
   类似chroot，将一个进程放到一个特定的目录执行。mnt namespace允许不同namespace的进程看到的文件结构不同，这样每个 namespace 中的进程所看到的文件目录就被隔离开了。同chroot不同，每个namespace中的container在/proc/mounts的信息只包含所在namespace的mount point。
5. uts namespace   
   UTS(“UNIX Time-sharing System”) namespace允许每个container拥有独立的hostname和domain name, 使其在网络上可以被视作一个独立的节点而非Host上的一个进程。
6. user namespace   
   每个container可以有不同的 user 和 group id, 也就是说可以在container内部用container内部的用户执行程序而非Host上的用户。

[深入浅出Docker（一）：Docker核心技术预览](http://www.infoq.com/cn/articles/docker-core-technology-preview)

## docker 命令



后台进程



|  |  |
| --- | --- |
| **参数** | **解释** |
| --api-enable-cors=false | 开放远程API调用的 CORS 头信息。这个接口开关对想进行二次开发的上层应用提供了支持。 |
| -b, --bridge="" | 挂载已经存在的网桥设备到 Docker 容器里。注意，使用 none 可以停用容器里的网络。 |
| --bip="" | 使用 CIDR 地址来设定网络桥的 IP。注意，此参数和 -b 不能一起使用。 |
| -D, --debug=false | 开启Debug模式。例如：docker -d -D |
| -d, --daemon=false | 开启Daemon模式。 |
| --dns=[] | 强制容器使用DNS服务器。例如： docker -d –dns 8.8.8.8 |
| --dns-search=[] | 强制容器使用指定的DNS搜索域名。例如： docker -d –dns-search [example.com](http://example.com/) |
| -e, --exec-driver="native" | 强制容器使用指定的运行时驱动。例如：docker -d -e lxc |
| -G, --group="docker" | 在后台运行模式下，赋予指定的Group到相应的unix socket上。注意，当此参数 –group 赋予空字符串时，将去除组信息。 |
| -g, --graph="/var/lib/docker" | 配置Docker运行时根目录 |
| -H, --host=[] | 在后台模式下指定socket绑定，可以绑定一个或多个 [tcp://host:port](tcp://host:port/), unix:///path/to/socket, fd://\* 或 [fd://socketfd](fd://socketfd/)。例如： |
|  | $ docker -H tcp://0.0.0.0:2375 ps 或者 |
|  | $ export DOCKER\_HOST="tcp://0.0.0.0:2375" |
|  | $ docker ps |
| --icc=true | 启用内联容器的通信。 |
| --ip="0.0.0.0" | 容器绑定IP时使用的默认IP地址 |
| --ip-forward=true | 启动容器的 net.ipv4.ip\_forward |
| --iptables=true | 启动Docker容器自定义的iptable规则 |
| --mtu=0 | 设置容器网络的MTU值，如果没有这个参数，选用默认 route MTU，如果没有默认route，就设置成常量值 1500。 |
| -p, --pidfile="/var/run/docker.pid" | 后台进程PID文件路径。 |
| -r, --restart=true | 重启之前运行中的容器 |
| -s, --storage-driver="" | 强制容器运行时使用指定的存储驱动，例如,指定使用devicemapper, 可以这样： |
|  | docker -d -s devicemapper |
| --selinux-enabled=false | 启用selinux支持 |
| --storage-opt=[] | 配置存储驱动的参数 |
| --tls=false | 启动TLS认证开关 |
| --tlscacert="/Users/dxiao/.docker/ca.pem" | 通过CA认证过的的certificate文件路径 |
| --tlscert="/Users/dxiao/.docker/cert.pem" | TLS的certificate文件路径 |
| --tlskey="/Users/dxiao/.docker/key.pem" | TLS的key文件路径 |
| --tlsverify=false | 使用TLS并做后台进程与客户端通讯的验证 |
| -v, --version=false | 显示版本信息 |

## docker 安装

centos 6下

$ sudo yum install http://[mirrors.yun-idc.com/epel](http://mirrors.yun-idc.com/epel)/6/i386/epel-release-6-8.noarch.rpm

$ sudo yum install docker-io

centos 7 下

$ yum **install** docker

获取镜像，可以使用docker pull命令来从仓库获取所需要的镜像

$ sudo service docker start

$ sudo docker pull ubuntu:12.04

#有时候官方仓库注册服务器下载慢，可以从其他仓库下载

$ sudo docker pull [dl.dockerpool.com](http://dl.dockerpool.com/):5000/ubuntu:12.04

# 现在就可以使用这个镜像啦，便开始创建一个容器，让其中运行bash应用。

$ sudo docker run -t -i ubuntu:12.04 /bin/bash

# 列出本地镜像

$ sudo docker images