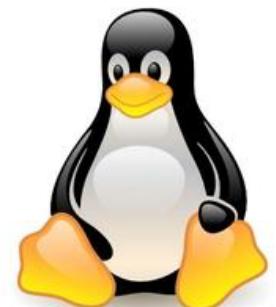


# Linux

*wangxiaodong@ouc.edu.cn*



# Linux History

- 1991年4月，芬兰赫尔辛基大学计算机系研究生Linus Torvalds开始为一个以后被称为“Linux”的内核而工作。（后附Linus的邮件）
- 1991年8月25日，Linus在网上发贴，寻找志同道合的合作伙伴。
- 1991年10月5日，Linus Torvalds在新闻组comp.os.minix发表了Linux V0.01，约有一万行代码。
- 1992年，全世界大约有1000个左右的人使用Linux，并有不少人提供初期的代码上载和评论。
- 1993年，大约由100多个程序员参与内核代码修改，内核核心由5人组成，V0.99 约有十万行代码。
- 1993年12月，Linux全球用户数约在10万左右。
- 1994年3月，Linux1.0问世，约有17万行代码。它完全按自由免费的协议发布，源码必须完全公开，之后很快Linux正式采用GPL协议。
- 1995年，Linux全球用户数大大超过50万，Linux已可在Intel、Digital和Sun SPARC处理器上运行，Linux Journal杂志已发行了10万册。内核发展到1.2，约有25万行代码。

# Linux History

## ■ 第一篇Linux帖子

From: torvalds@klaava.Helsinki.FI

Newsgroups: comp.os.minix

Subject: What would you like to see most in minix?

Date: 25 Aug 91 20:57:08 GMT

Hello everybody out there using minix. I'm doing a (free) operating system  
(just a hobby, won't be big and professional like gnu) for 386(486) AT clones.  
This has been brewing since april, and is starting to get ready. I'd like any  
feedback on things people like/dislike in minix, as my OS resembles it  
somewhat (same physical layout of the file-system (due to practical reasons)  
among other things).

# Linux History

- 1996年6月，Linux内核2.0发布，可支持多个处理器，约由40万行代码。Linux全球用户数约在350万左右。
- 1997年夏，制作电影《泰坦尼克号》所用的160台Alpha图形工作站中，有105台采用了Linux操作系统。
- ... ...
- <http://www.kernel.org>
- 最新的内核稳定版本为4.4.3。

The screenshot shows the homepage of The Linux Kernel Archives. At the top, there's a navigation bar with links for About, Contact us, FAQ, Releases, Signatures, and Site news. To the right of the navigation is a small Tux the Penguin icon. Below the navigation, there's a section for "Protocol" and "Location" with links for HTTP, GIT, and RSYNC. A prominent yellow button on the right side displays the "Latest Stable Kernel: 4.3.3" with a download icon. The main content area lists various kernel versions with their release dates and download links.

Protocol	Location
HTTP	<a href="https://www.kernel.org/pub/">https://www.kernel.org/pub/</a>
GIT	<a href="https://git.kernel.org/">https://git.kernel.org/</a>
RSYNC	<a href="rsync://rsync.kernel.org/pub/">rsync://rsync.kernel.org/pub/</a>

Latest Stable Kernel:  
4.3.3

mainline:	4.4-rc5	2015-12-14	[tar.xz]	[pgp]	[patch]	[view diff]	[browse]		
stable:	4.3.3	2015-12-15	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
stable:	4.2.8 [EOL]	2015-12-15	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	4.1.15	2015-12-15	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	3.18.25	2015-12-15	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	3.14.58	2015-12-09	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	3.12.51	2015-11-25	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]
longterm:	3.10.94	2015-12-09	[tar.xz]	[pgp]	[patch]	[inc. patch]	[view diff]	[browse]	[changelog]

# Linux History



Ken Thompson  
C语言之父和Unix之父



Dennis Ritchie  
C语言之父和Unix之父



Richard Stallman  
著名黑客，GNU创始人  
开发了Emacs、gcc、  
bash shell



Linus Torvalds  
Linux之父

# This man is Linus Torvalds.

Aalto Talk with Linus Torvalds

aaltouniversityace

+ 订阅

29 个视频 ▾



2012年6月14日，由阿尔托大学奥塔涅米创业中心(ACE)主办的Aalto Talk节目邀请到了Linux系统创始人Linus Torvalds，与观众进行互动交流。Linus期间非常出人意料地对NVIDIA爆出了粗口。

# Linux的特点

- Linux is free
  - 开放性
  - 可靠的系统安全
  - 良好的可移植性
  - 多用户性
  - 多任务
  - 良好的用户界面
  - 设备独立性
  - 强大的网络功能
- 
- Linux的版本一般指内核版本；
  - Linux通过不同的命名机制来区分内核类别，采用三个由“.”分割的数字来表示内核版本号。第一个数字叫主板本号，第二个次版本号，第三个叫修订版本号。**次版本号如果是偶数，那么内核是稳定的，如果是奇数，则表示内核是处在开发中。**

# Linux and GNU

- GNU, GNU is not UNIX. <http://www.gnu.org/>
- GNU/Hurd
- Common GNU software

Bash: The GNU shell

GCC: The GNU C Compiler

GDB: The GNU Debugger

Coreutils: a set of basic UNIX-style utilities, such as **ls**, **cat** and **chmod**

Findutils: to search and find files

Fontutils: to convert fonts from one format to another or make new fonts

The Gimp: GNU Image Manipulation Program

Gnome: the GNU desktop environment

Emacs: a very powerful editor

Ghostscript and Ghostview: interpreter and graphical frontend for PostScript files.

GNU Photo: software for interaction with digital cameras

Octave: a programming language, primarily intended to perform numerical computations and image processing.

GNU SQL: relational database system

Radius: a remote authentication and accounting server

.....



# Linux Distribution



*Timeline*

<http://futurist.se/gldt/>

[https://en.wikipedia.org/wiki/Linux\\_distribution](https://en.wikipedia.org/wiki/Linux_distribution)

<https://linuxtoy.org>

# Ubuntu

Ubuntu由Mark Shuttleworth ( 马克·舍特尔沃斯 ) 创立 , Ubuntu以 Debian GNU/Linux不稳定分支为开发基础。



# 嵌入式Linux

- **uClinux** 它是针对控制领域而设计的Linux系统，专门用来配合没有MMU的微处理器工作。
- **RTLinux** 它是一个实时的Linux版本。由于Linux2.6版本之前是不可抢占式，不能真正实现实时应用。 RT-Linux用巧妙的方式解决了此问题，**它并没有重写Linux的内核，而是实现了一个高效的可抢占式实时调度核心**并把Linux作为此核心的一个优先级最低的进程运行，用户可以编写自己的实时进程，和标准的Linux共同运行。

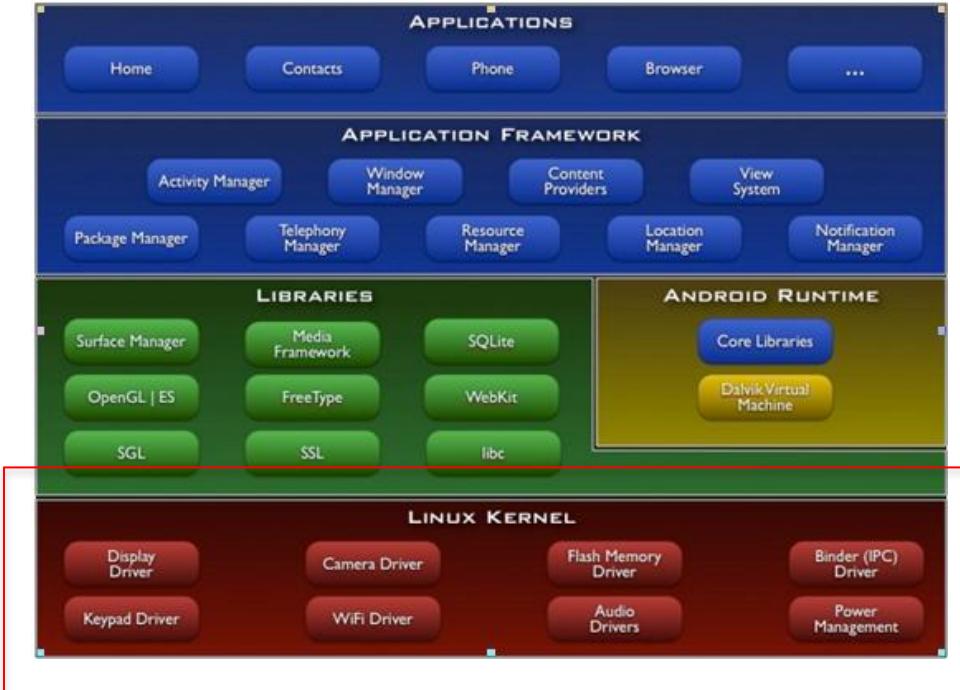
uClinux. <http://www.uclinux.org/>

RTLinux. <https://en.wikipedia.org/wiki/RTLinux>

抢占式和非抢占式内核. [http://blog.csdn.net/jack\\_wong2010/article/details/8573454](http://blog.csdn.net/jack_wong2010/article/details/8573454)

# Android

## ■ 架构

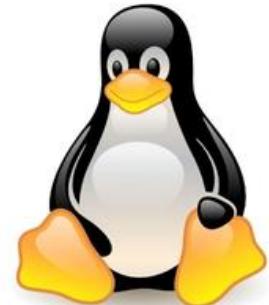


Google vs Oracle : Java版权之争

Linux Kernel

# So, what is Linux?

- Linux is an implementation of UNIX.
- The Linux operating system is written in the C programming language.
- There's a Linux for everyone.
- Linux uses GNU tools, a set of freely available standard tools for handling the operating system.



# Installation

- 直接格掉Windows，安装一个Linux发行版，如Ubuntu（强烈推荐）
- 双系统，Windows和Linux并存（不推荐）
- 虚拟机方式安装
  - Windows系统中通过虚拟机安装Linux（较推荐）
  - Linux系统中通过虚拟机安装Windows（Kid me？）



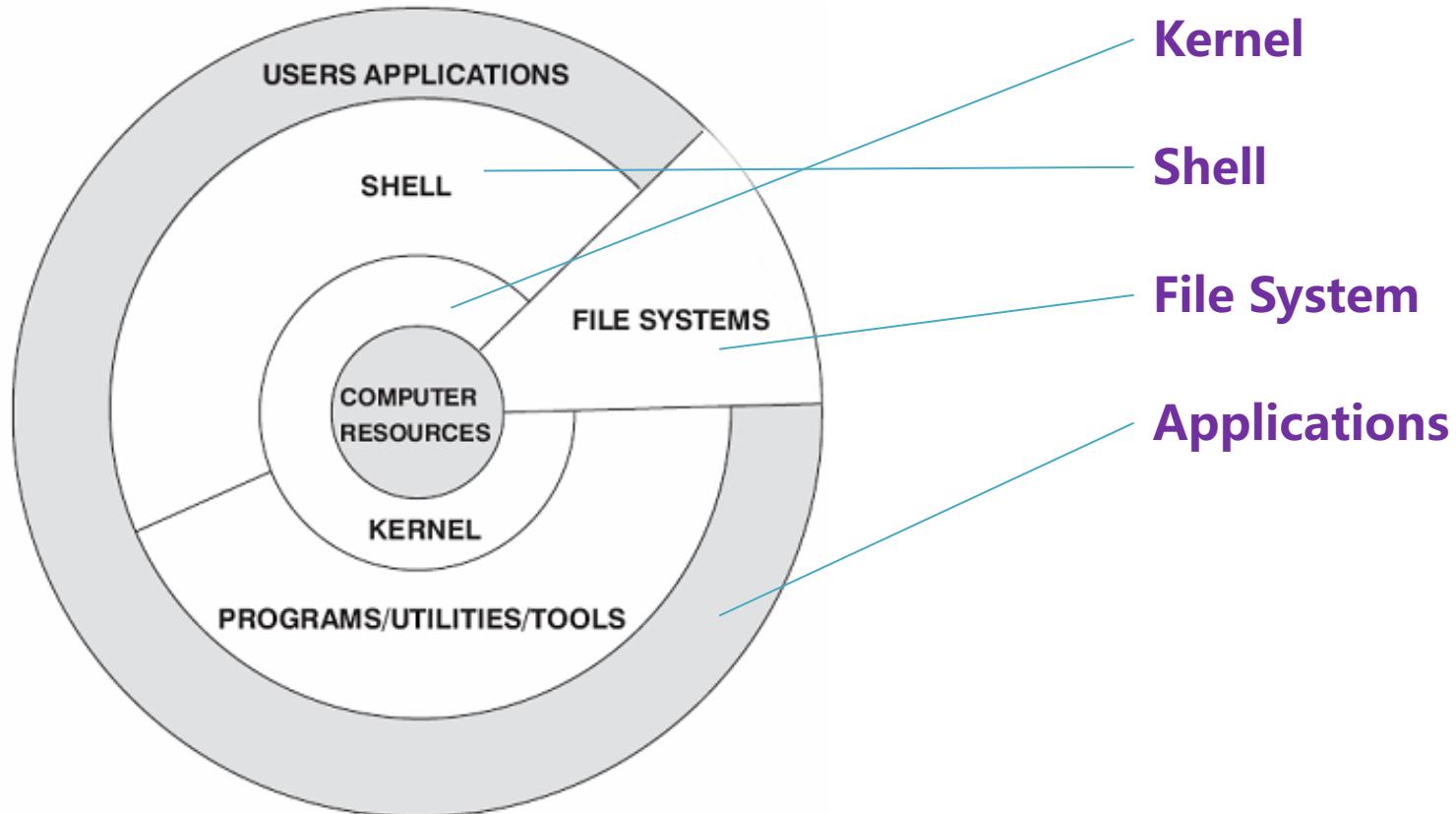
阴险的笑

建议用一个连续的时间（例如你的寒假，或者没人打扰的完整的一周）做以下这件事情：

从 0 构建一个 Linux     **Linux from Scratch.**

<http://www.linuxfromscratch.org/>

# Architecture



# Architecture

## Kernel

- 内存管理
- 进程管理
- 文件系统（虚拟文件系统 VFS）
- 设备驱动
- 网络接口

建议阅读Linux内核源代码，可以从Linux-0.11入手，推荐Linux内核完全注释  
(赵炯著)

# Architecture

## Shell

- 系统的用户界面
- 提供用户与内核进行交互操作的一种接口
- 接收用户输入的命令并把它送入内核去执行
- Shell具备可编程特性
- 主要的Shell版本
  - Bourne [bo:n] Shell , 贝尔实验室开发
  - **BASH , GNU的Bourne Again Shell , 基本默认**
  - Korn Shell
  - C Shell , SUN公司Shell的BSD版本

# Architecture

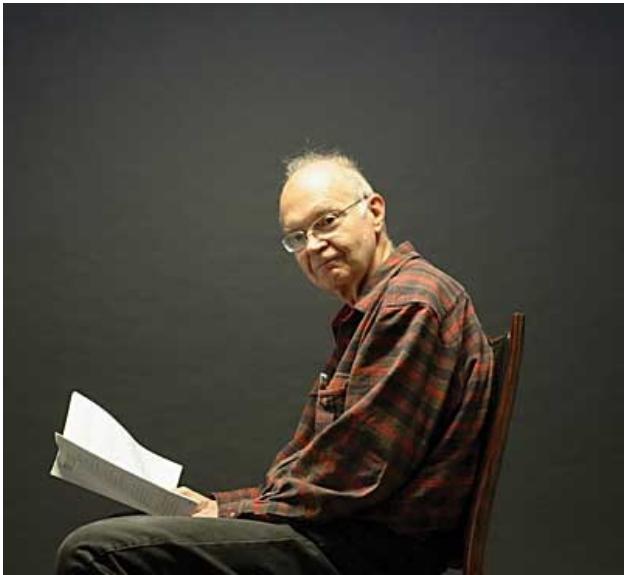
## Shell

```
Ubuntu 14.04.3 LTS ubuntu tty1  
ubuntu login: xiaodong  
Password:  
Last login: Mon Aug 31 01:39:14 PDT 2015 on tty1  
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-63-generic x86_64)  
  
 * Documentation: https://help.ubuntu.com/  
  
xiaodong@ubuntu:~$ _
```

- 如果每天工作在这样的计算机前我们肯定会比现在更加健康
- 比现在更像一个计算机系的学生
- 比现在更爱看书
- 比现在更爱户外活动

# Architecture

高德纳 ( Donald Ervin Knuth ) 的桌面 ( 基于 FVWM 的窗口系统 )



The screenshot shows a Linux desktop environment with a terminal window open. The terminal displays Mathematica code related to asymptotic analysis and complex variable theory. The code includes definitions for functions like `AddToFunc`, `InitFunction`, and `RestartFunction`, and uses `Series` and `NIntegrate` to compute limits and integrals. The background shows a window titled "John Cristy, E.I. du Pont De Nemours and Company Incorporated" with a date of "1 May 1994". A digital clock in the top right corner shows "08:23 SAT 20 MAY".

```
+ "I" Function Basic-Setup
+ "I" Desk 0 2
+ "I" Exec exec xosview -geometry 400x200+11+92
+ "I" Wait xosview
+ "I" Desk 0 1
+ "I" Exec exec rxvt -geometry 82x50-3-0 -fg \#ffff4c0 -bg \#381900 -sr
+ "I" Wait rxvt
+ "I" Exec exec emacs -geometry 80x58+0+0 -fg \#ffe97a -bg \#00002b
+ "I" Wait emacs
+ "I" Desk 0 0
+ "I" Exec exec rxvt -geometry 82x50-3-0 -fg white -bg black -sr
+ "I" Wait rxvt
+ "I" Exec exec emacs -geometry 80x58+0+0 -fg \#ffe97a -bg \#002b00
+ "I" Wait emacs
## "I" Exec exec xeyes -geometry 55x50+823+33 -fg DarkGreen -center dimgray -ou
tline \#000060
## "I" Wait xeyes
+ "I" Exec exec unclutter
+ "I" Echo ready...
AddToFunc RestartFunction "I" Echo restarting...
+ "I" Function Basic-Setup

By putting the line "xset root 2 > /tmp/xmw.out" in my .xinitrc,
# I'm getting an error log in /tmp/xmw.out. The following command
xsetroot -root 2> /tmp/xmw.out (Fundamental) -- 992
8981.99999, 99948, 43434$-$64182,00000, 00000$-$5\cr
\noalign{\vskip 3pt}
&\multispan2\hrulefill&\multispan2\hrulefill\&\cr
}
\nendinsert

The asymptotic behavior of $L-k$ can be determined by using simple
````asymptotic methods```
principles of complex variable theory. The denominator of Eq(25) is zero only
when E<-1= $\infty$ , namely when
E<(x-1)\cos y=\sqrt{quad\boxed{box}}(\text{and}\sqrt{quad e^x(x-1)\sin y}=y,\text{Eqno}(27))
if we write x=x+iy.
Figure 3 shows the superimposed graphs of these two equations, and we note
that there intersect at the points
(Def x=\sqrt{quad\boxed{box}}(x+iy),Def z=2w z\?99?)
$z\approx_0 z-1,z-1,z-2,z-2,\dots$ where z_0=0 is,
z_1<(3.08984,-30165,-13044)+(-7.46148,-92886,-54255)+i, Eqno(28) is
and the imaginary part Im(z_{k+1}) is roughly equal to
Im(z_k) for large k. Since
|\lim_{k\rightarrow\infty} z_k|Bigl(<1-z\over e^x(x-1)-z\Biggr)=1,\sqrt{quad}
\boxed{box}(For k>0, and since the limit is -2 for k=0, the function
z\approx_0 z-1,z-1,z-2,z-2,\dots
R_m(z)=z+(2z\over z-z_0)+(z\over z-z_1)
+(z\over z-z_2)+(z\over z-z_3)+\dots+(z\over z-z_m)
has no singularities in the complex plane for
-5,1,3.tex :388 (TeX)----572
```

[https://en.wikipedia.org/wiki/Donald\\_Knuth](https://en.wikipedia.org/wiki/Donald_Knuth)  
<http://www-cs-faculty.stanford.edu/~uno/>

# Architecture

## Shell Commands

Command	Meaning
<b>apropos</b>	Search information about a command or subject.
<b>cat</b>	Show content of one or more files.
<b>cd</b>	Change into another directory.
<b>exit</b>	Leave a shell session.
<b>file</b>	Get information about the content of a file.
<b>info</b>	Read Info pages about a command.
<b>logout</b>	Leave a shell session.
<b>ls</b>	List directory content.
<b>man</b>	Read manual pages of a command.
<b>passwd</b>	Change your password.
<b>pwd</b>	Display the current working directory.

# Architecture

## Shell

- 输入输出重定向：
  - > : 重定向输出
  - < : 重定向输入
- 管道：将某一个程序的输出直接送入到另一个程序，作为输入，符号为 “|”  
命令1|命令2 |命令3...
- 后台执行命令：执行shell命令时，如果在命令后加一个 “&” ，则回车后立刻返回到命令提示符状态下，而命令在后台执行。  
命令&

# Architecture

## Shell

- 暂停和恢复进程
  - command& 让进程在后台运行
  - jobs 查看后台运行的进程
  - fg n 让后台运行的进程n到前台来
  - bg n 让进程n到后台去
- ctrl+z 可以将一个正在前台执行的命令放到后台，并且暂停

# Architecture

## Shell Scripts

```
#!/bin/bash  
echo "Hello World!"
```

- "#!" 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种Shell。
- echo命令用于向窗口输出文本。

## 执行脚本

```
chmod +x ./test.sh  
./test.sh
```

# Architecture

## File System

- 文件系统是文件存放在磁盘等存储设备上的组织方法
- Linux能够支持目前流行的N多文件系统，如Btrfs、JFS、ReiserFS、ext、ext2、ext3、ext4、.....

## 文件类型

- 普通文件：C语言元代码、SHELL脚本、二进制的可执行文件等，分为纯文本和二进制
  - 目录文件：目录，存储文件的唯一地方
  - 链接文件：指向同一个文件或目录的文件
- 设备文件：与系统外设相关的，通常在/dev下面，分为块设备和字符设备
- 管道(FIFO)文件：提供进程建通信的一种方式
- 套接字(socket)文件：该文件类型与网络通信有关

# Architecture

## File System

标准化的目录结构

```
tree -L 1
```

```
xiaodong@ubuntu:/$ tree -L 1
.
├── bin
├── boot
├── dev
├── etc
├── home
│   ├── initrd.img -> boot/initrd.img-3.13.0-63-generic
│   └── initrd.img.old -> boot/initrd.img-3.13.0-32-generic
├── lib
├── lib64
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── srv
├── sys
└── tmp
├── vmlinuz -> boot/vmlinuz-3.13.0-63-generic
└── vmlinuz.old -> boot/vmlinuz-3.13.0-32-generic

20 directories, 4 files
xiaodong@ubuntu:/$
```

# Architecture

## File System

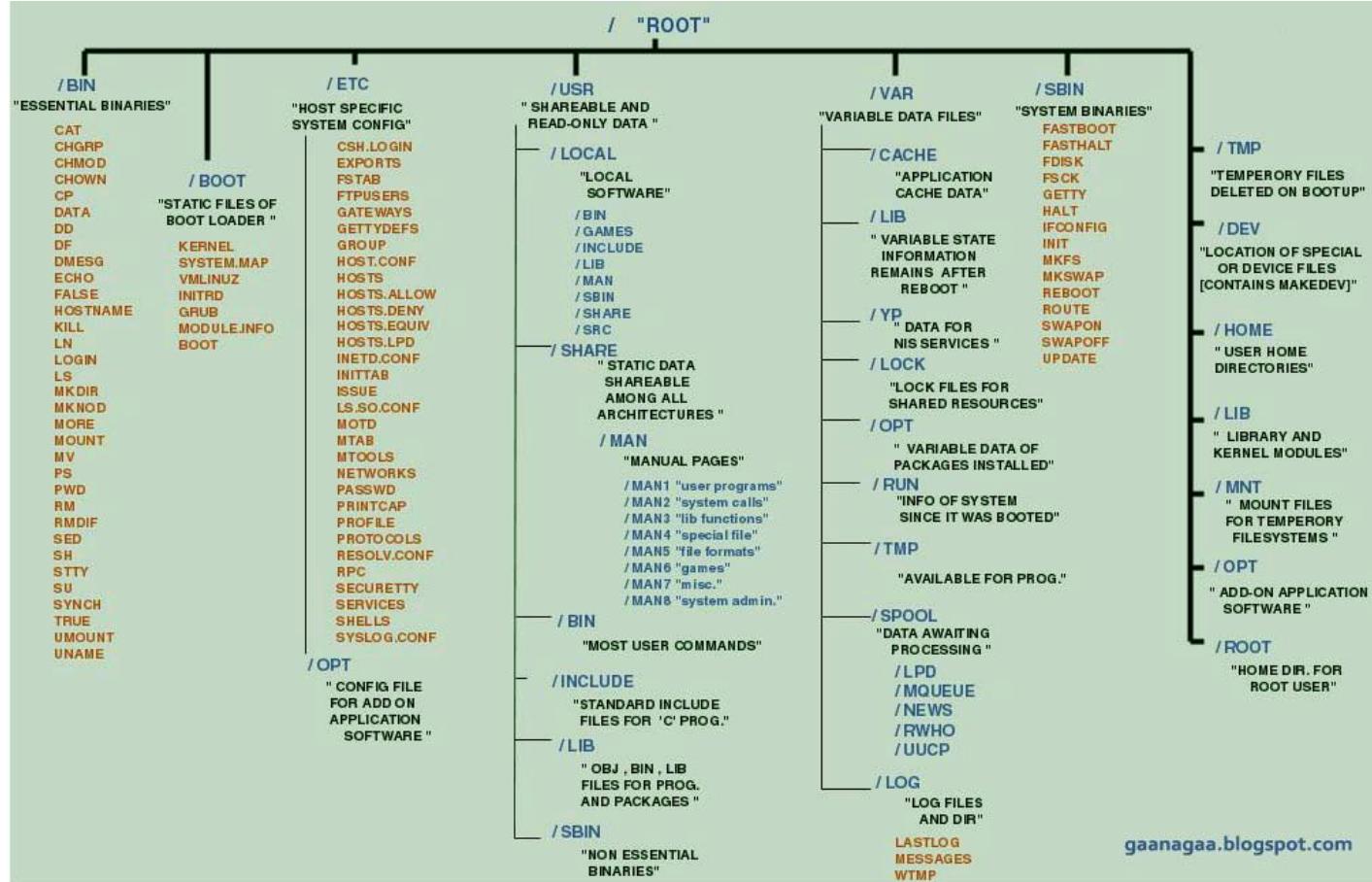
```
tree -L 1 -p /dev
```

显示文件类型及权限

目录树的显示深度

# Architecture

## File System



# Architecture

## File System

### 磁盘分区

```
sudo fdisk -l
```

```
xiaodong@ubuntu:~$ sudo fdisk -l

Disk /dev/sda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders, total 41943040 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000dae13

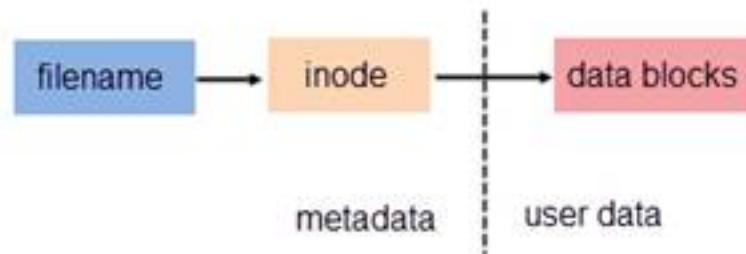
      Device Boot   Start     End   Blocks Id System
/dev/sda1  *       2048 39845887 19921920  83 Linux
/dev/sda2          39847934 41940991 1046529    5 Extended
/dev/sda5          39847936 41940991 1046528  82 Linux swap / Solaris
xiaodong@ubuntu:~$ _
```

# Architecture

## File System

### 硬链接和软链接

文件都有文件名与数据，这在 Linux 上被分成两个部分：**用户数据 (user data)** 与**元数据 (metadata)**。用户数据，即文件数据块 (data block)，数据块是记录文件真实内容的地方；而元数据则是文件的附加属性，如文件大小、创建时间、所有者等信息。在 Linux 中，元数据中的 inode 号 (inode 是文件元数据的一部分但其并不包含文件名，inode 号即索引节点号) 才是文件的唯一标识而非文件名。文件名仅是为了方便人们的记忆和使用，系统或程序通过 inode 号寻找正确的文件数据块。



# Architecture

## File System

### 硬链接和软链接

在 Linux 系统中查看 inode 号可使用命令 stat 或 ls -i

使用命令 mv 移动并重命名文件，其结果不影响文件的用户数据及 inode 号

```
xiaodong@ubuntu:~$ stat docker.json
  File: 'docker.json'
  Size: 109          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 137781      Links: 1
Access: (0664/-rw-rw-r--)
Modify: 2015-09-16 08:55:52.815148894 -0700
Change: 2015-09-16 08:55:52.815148894 -0700
 Birth: -
xiaodong@ubuntu:~$ mv docker.json docker.json.1
xiaodong@ubuntu:~$ stat docker.json.1
  File: 'docker.json.1'
  Size: 109          Blocks: 8          IO Block: 4096   regular file
Device: 801h/2049d      Inode: 137781      Links: 1
Access: (0664/-rw-rw-r--)
Modify: 2015-09-16 08:55:52.815148894 -0700
Change: 2015-12-17 07:26:26.266540741 -0800
 Birth: -
xiaodong@ubuntu:~$
```

# Architecture

## File System

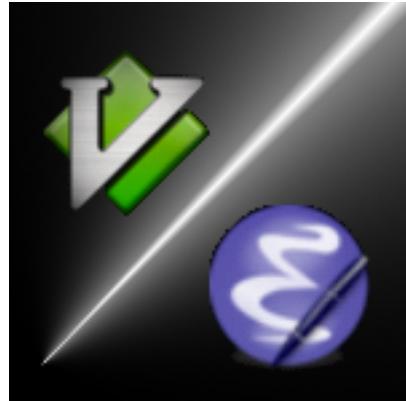
### 硬链接和软链接

为解决文件的共享，Linux引入2种链接：硬链接和软链接。

- 若一个inode号对应多个文件名，则称这些文件为硬链接，使用ln创建；
- 软链接与硬链接不同，若文件用户数据块中存放的内容是另一文件的路径名的指向，则该文件就是软连接，使用ln -s创建；
- 软链接就是一个普通文件，只是数据块内容有点特殊，软链接有着自己的inode号以及用户数据块。

# Architecture Applications

标准的Linux系统（Linux发行版）一般都有一套称为应用程序集，它包括文本编辑器、编程语言、X Window、办公套件、Internet工具和数据库等。



- Vim
- Emacs

知乎：*Vim 和 Emacs 到底哪个更牛逼一点？* <http://www.zhihu.com/question/20846396>

# Linux启动脚本风格

- System V
- BSD
- Slackware使用BSD风格的init脚本，而很多别的发行版使用System V风格的init脚本。
- SysV和BSD脚本都是能让人读懂的，即它们都是Shell脚本，而不是已编译的程序，其主要的区别在于脚本的设计和组织方式。

# Ubuntu启动流程分析

## Upstart方式

- 采用该种方式的发行版有Ubuntu ( 6.10 and later ) , Fedora ( 9.10 and later ) , Debian ( optional ) 。
- Upstart基于事件机制，系统的所有服务、任务都是由事件驱动的。

## System V

启动流程依赖`/etc/inittab`，init进程启动后第一时间找inittab，根据inittab中的配置初始化系统，设置系统runlevel及进入各runlevel对应要执行的命令。

假设当前inittab中设置的默认runlevel是5，则init会运行`/etc/init.d/rc 5`命令，该命令会依据系统服务的依赖关系遍历执行`/etc/rc5.d`中的脚本/程序。进入`/etc/rc5.d`目录可以发现里面的文件都是到`/etc/init.d/`下对应的脚本/程序的软链接。以S开头的为启动，以K开头的为停止。并且S/K后面的两位数数字代表了服务的启动顺序（由服务依赖关系决定）。

# Ubuntu启动流程分析

**Upstart方式**，事件驱动的，系统服务的启动、停止等等均是由事件决定的，反过来，系统服务的启动、停止也可以作为事件源触发其他服务。并且事件并不一定得由系统内部产生，用户可以手工的键入start/stop [Service]产生事件来启动/终止服务。

```
man upstart-events
```

系统服务基于**/etc/init**中的配置确定自身应该何时启动和终止。

```
xiaodong@ubuntu:/etc/init$ cat cron.conf
# cron - regular background program processing daemon
#
# cron is a standard UNIX program that runs user-specified programs at
# periodic scheduled times

description      "regular background program processing daemon"

start on runlevel [2345]
stop on runlevel [!2345]

expect fork
respawn

exec cron
```

# Ubuntu启动流程分析

- Ubuntu并不是完全使用的是Upstart方式的初始化，由于6.10之前的版本采用的SysV init，以及某些服务的需要，Ubuntu采用的是兼容模式。
- 系统中既有SysV风格启动的服务，也有Upstart启动的服务。

```
/etc/init  
/etc/init.d  
/etc/rc${runlevel}.d
```

- 进入/etc/init目录，会发现几个跟rc有关的配置文件：

```
rc.conf  
rc-sysinit.conf  
rcS.conf
```

- rc-sysinit在startup事件发生时被启动
- rc在系统runlevel变化时被启动
- rcS在系统runlevel为S时启动

# Ubuntu启动流程分析

- 采用Upstart方式启动的服务则在/etc/init/目录中有属于自己的一份配置文件，终端下键入：

```
initctl list
```

```
xiaodong@ubuntu:/etc/init$ initctl list
gnome-keyring-gpg stop/waiting
indicator-application start/running, process 2642
unicast-local-avahi stop/waiting
update-notifier-crash stop/waiting
update-notifier-hp-firmware stop/waiting
xsession-init stop/waiting
dbus start/running, process 2444
update-notifier-cds stop/waiting
gnome-keyring-ssh stop/waiting
gnome-session (Unity) start/running, process 2553
ssh-agent stop/waiting
unity-voice-service stop/waiting
upstart-dbus-session-bridge start/running, process 2477
unity7 stop/waiting
```

# Virtualization & Docker

*Wang Xiaodong*

# Virtualization

In computing, virtualization refers to the act of creating a virtual (rather than actual) version of something, including virtual computer hardware platforms, operating systems, storage devices, and computer network resources.

Virtualization began in the 1960s, as a method of logically dividing the system resources provided by mainframe computers between different applications. Since then, the meaning of the term has broadened.

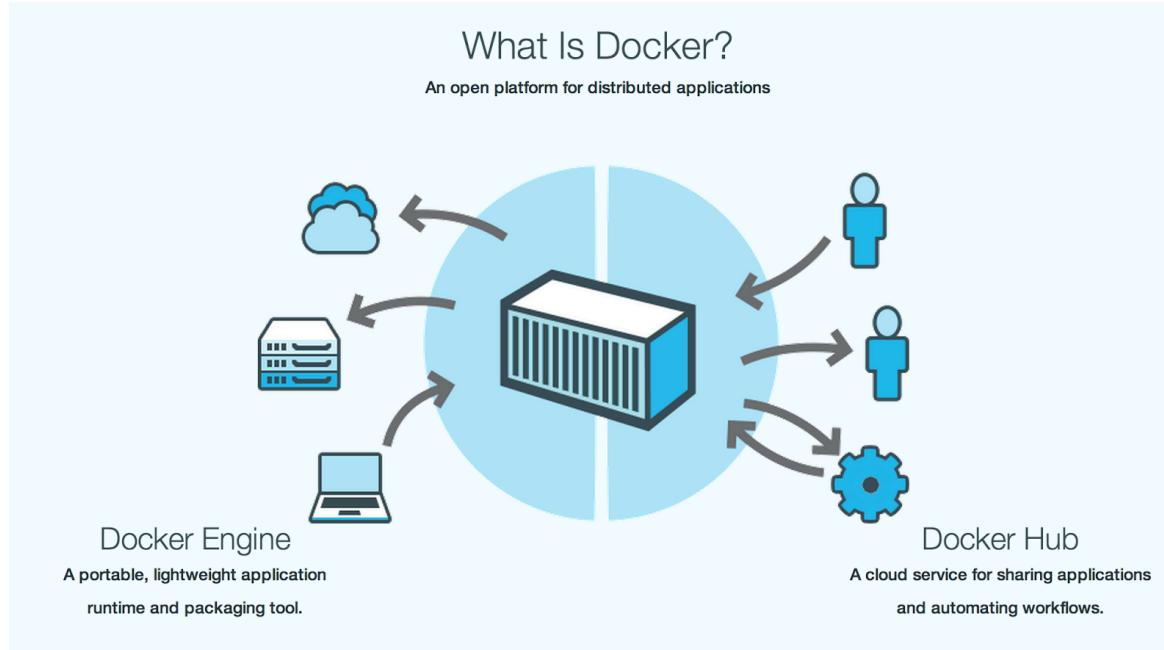
## Related Products

- Vmware
- Virtualbox

# Docker

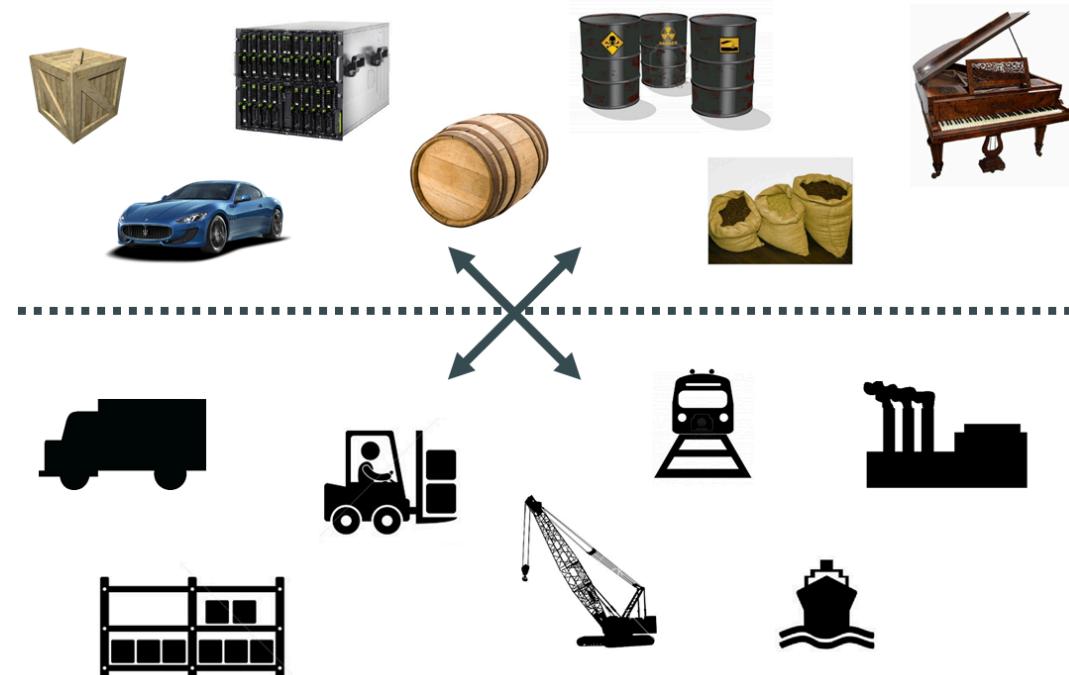


Docker is an open platform for building, shipping and running distributed applications. It gives programmers, development teams and operations engineers the common toolbox they need to take advantage of the distributed and networked nature of modern applications.

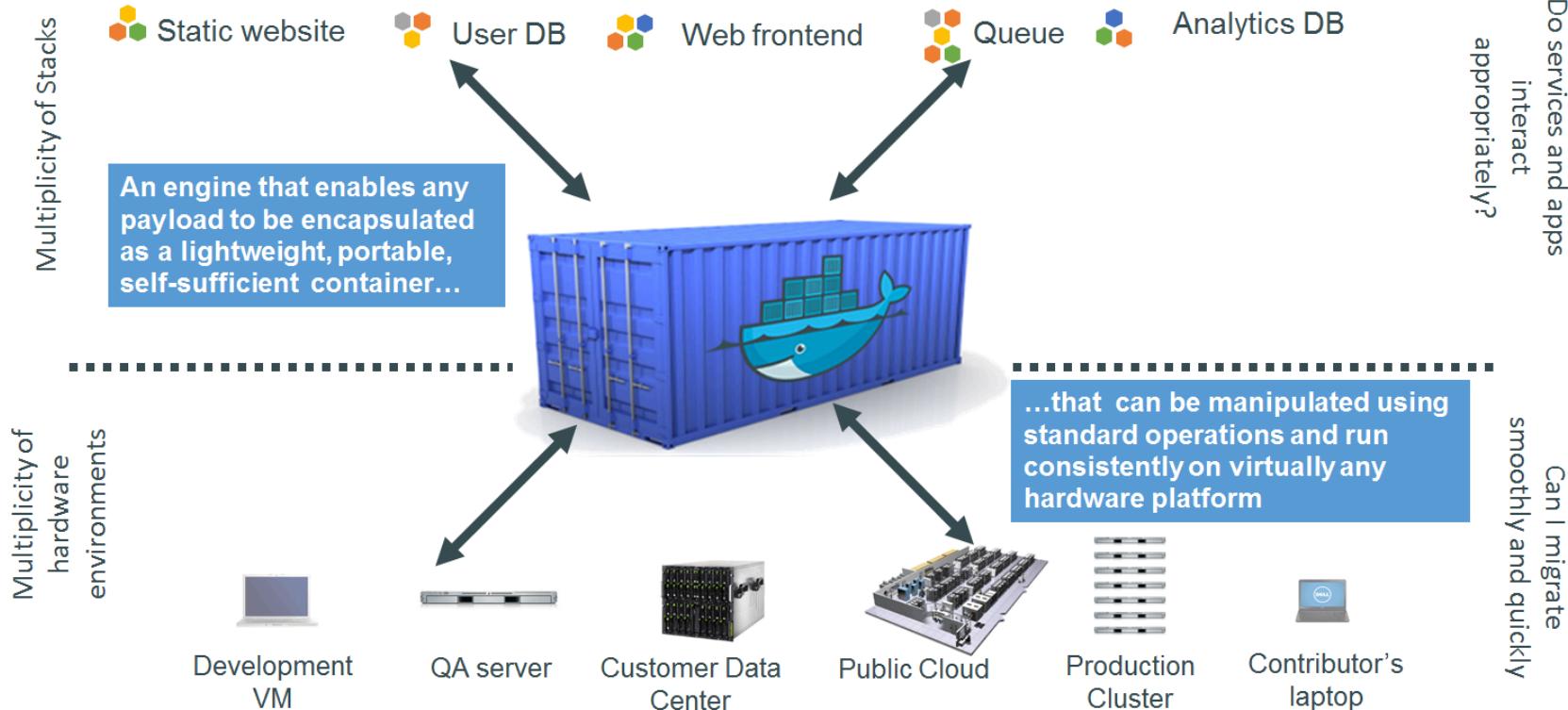


# Docker

简单的说Docker是一个构建在LXC之上的，基于进程容器（Process Container）的轻量级VM解决方案。

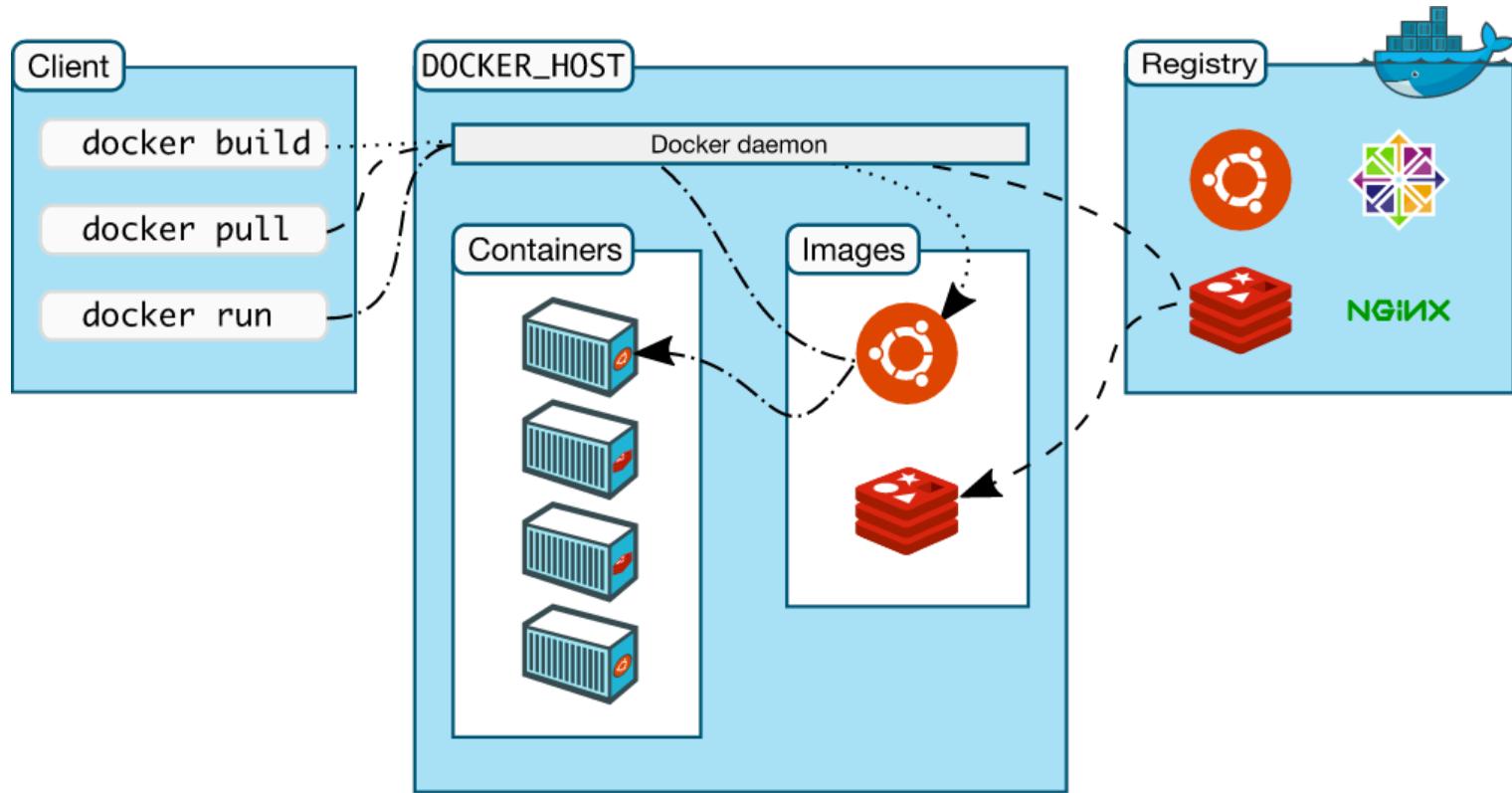


# Docker



Docker的初衷是将各种应用程序和他们所依赖的运行环境打包成标准的container/image，进而发布到不同的平台上运行。

# Docker

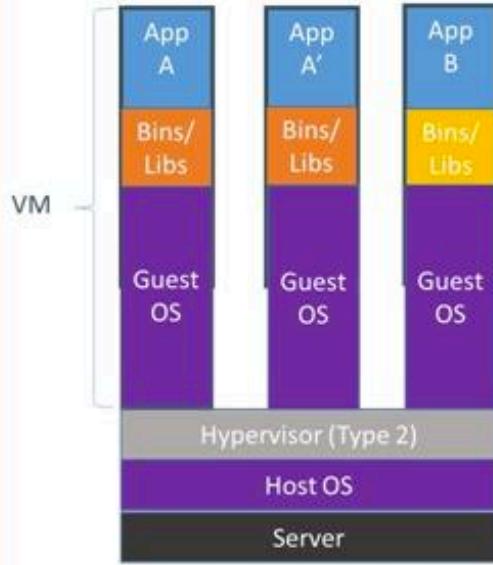


# Why Docker?

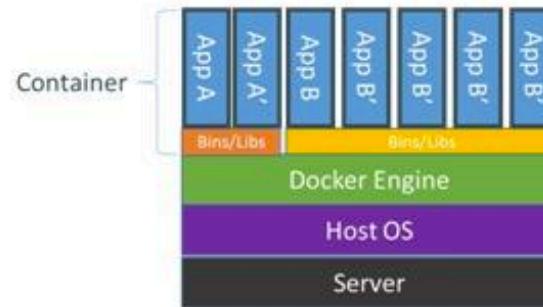
- 应用环境管理复杂
- 云计算时代的到来
- 虚拟化手段的变化
- LXC的便携性

# Docker

## Containers vs. VMs



Containers are isolated,  
but share OS and, where  
appropriate, bins/libraries



Docker Container和普通的虚拟机Image相比，最大的区别是它并不包含操作系统内核。

# Docker vs. LXC

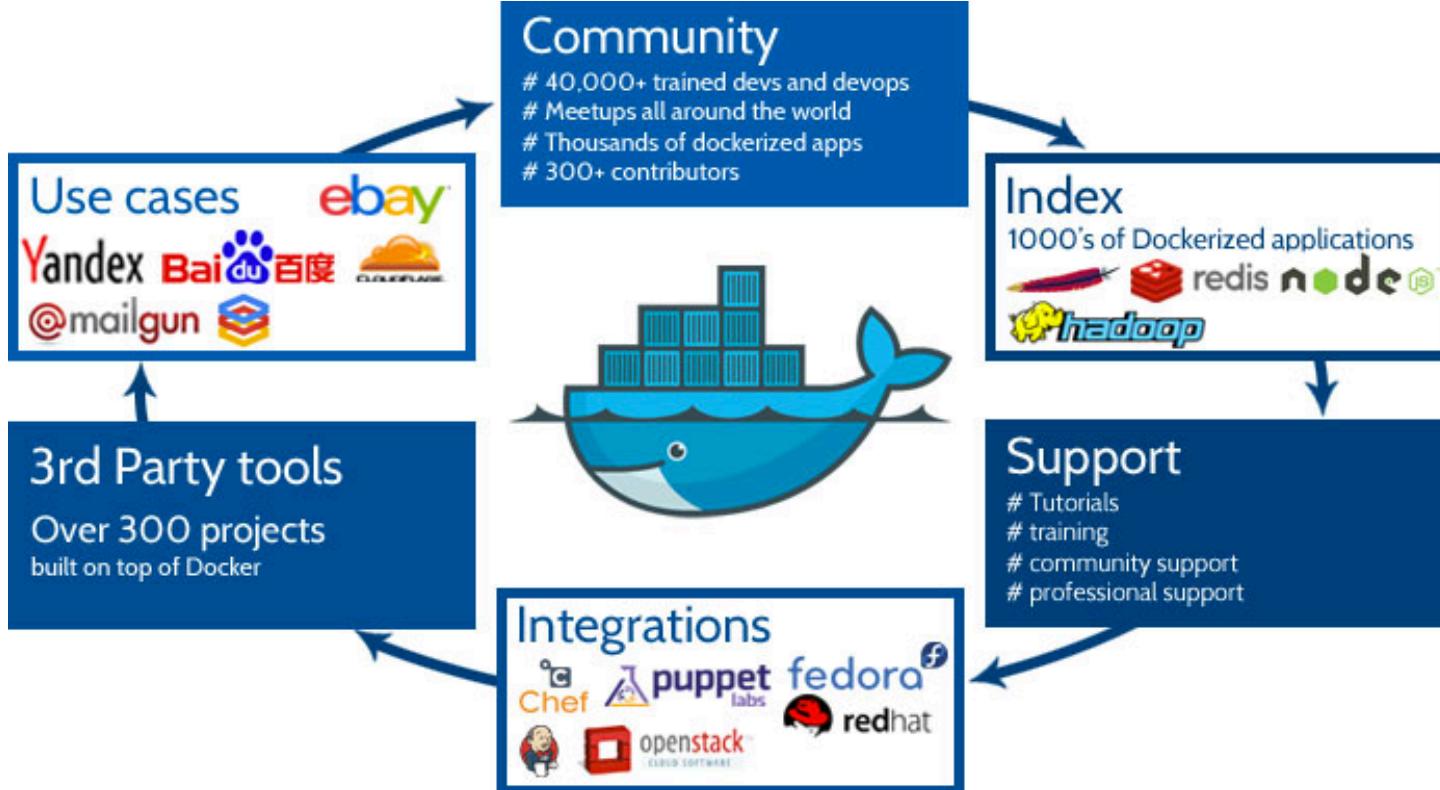
基本上可以认为目前的Docker是LXC的一个高级封装，提供了各种辅助工具和标准接口方便使用LXC，你可以依靠LXC和各种脚本实现与Docker类似的功能，就像你不使用APT/yum等工具也可以安装软件包一样，使用他们的关键原因是方便易用！

实际使用中，我们一般不用关心底层LXC的细节，同时也不排将来Docker实现基于非LXC方案的可能性（已经在这样做了）。

在LXC的基础上，Docker额外提供的特性包括：

- 标准统一的打包部署运行方案
- 历史版本控制
- Image的重用和共享发布等

# Growing Docker



例如，百度的BAE平台的PaaS服务由Docker支持。

# Technologies in Docker

Docker是一个操作系统级的、容器化的虚拟化方法。

隔离性	Linux Namespace (NS)
pid namespace	不同用户的进程就是通过pid namespace隔离开的，且不同namespace中可以有相同pid，所有的LXC进程在docker中的父进程为docker进程，每个LXC进程具有不同的namespace。由于允许嵌套，因此可以很方便的实现Docker in Docker。
net namespace	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，每个container的网络能够隔离，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。

# Technologies in Docker

隔离性	Linux Namespace (NS)
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。Container的进程间交互实际上还是Host上具有相同pid namespace的进程间交互，因此在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。
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。

# Technologies in Docker

隔离性	Linux Namespace (NS)
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。
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。

# Technologies in Docker

## 可配额/可度量 Control Groups (cgroups)

cgroups 实现了对资源的配额和度量。cgroups 提供类似文件的接口，在/cgroup目录下新建一个文件夹即可新建一个group，在此文件夹中新建task文件，并将pid写入该文件，即可实现对该进程的资源控制。groups可以限制blkio、cpu、cpuacct、cpuset、devices、freezer、memory、net\_cls、ns九项子系统的资源。

blkio 这个子系统设置限制每个块设备的输入输出控制。例如磁盘、光驱以及usb等。

cpu 这个子系统使用调度程序为cgroup任务提供cpu的访问。

cpuacct 产生cgroup任务的cpu资源报告。

cpuset 如果是多核心的cpu，这个子系统会为cgroup任务分配单独的cpu和内存。

devices 允许或拒绝cgroup任务对设备的访问。

freezer 暂停和恢复cgroup任务。

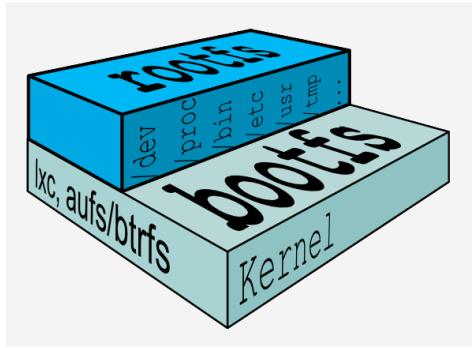
memory 设置每个cgroup的内存限制以及产生内存资源报告。

net\_cls 标记每个网络包以供cgroup方便使用。

ns 名称空间子系统。

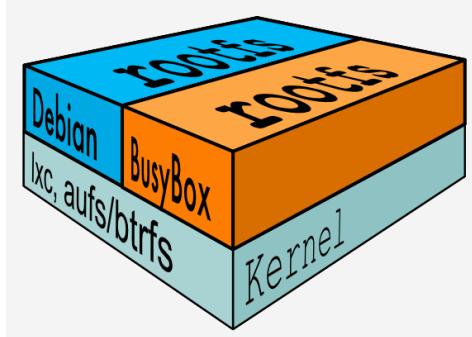
Refer to: <https://www.kernel.org/doc/Documentation/cgroups/>

# Technologies in Docker



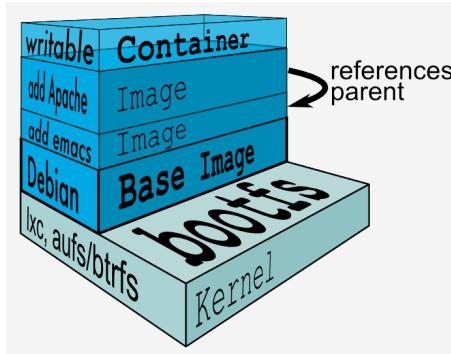
典型Linux启动运行需要两个FS: bootfs + rootfs

bootfs (boot file system) 主要包含bootloader和kernel，bootloader主要是引导加载kernel，当boot成功后kernel被加载到内存中后bootfs就被umount，rootfs包含的就是典型Linux系统中的/dev, /proc, /bin, /etc等标准目录和文件。

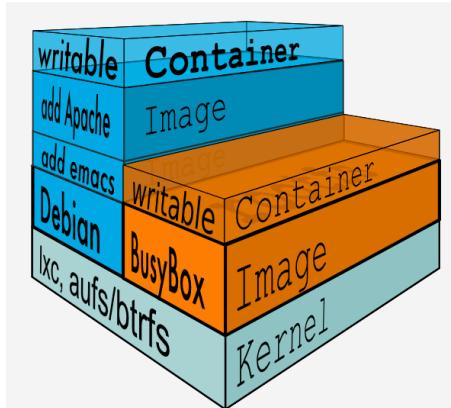


对于不同的linux发行版，bootfs基本是一致的，但rootfs会有差别，因此不同的Linux发行版可以公用bootfs。

# Technologies in Docker



典型的Linux在启动后，首先将rootfs设置为readonly，进行一系列检查，然后将其切换为readwrite供用户使用。在Docker中，初始化时也是将rootfs以readonly方式加载并检查，然而接下来利用union mount的方式将一个readwrite文件系统挂载在readonly的rootfs之上，并且允许再次将下层的FS设定为readonly并且向上叠加，这样一组readonly和一个writeable的结构构成一个Container的运行时态，每一个FS被称作一个FS层。



上层的Image依赖下层的Image，因此Docker中把下层的Image称作父Image，没有父Image的Image称作base Image。因此要从一个Image启动一个Container，Docker会先加载这个Image和依赖的父Images以及Base Image，用户的进程运行在writeable的Layer中。所有Parent Image中的数据信息以及ID、网络和LXC管理的资源限制等具体Container的配置，构成一个Docker概念上的Container。

# Install Docker on Ubuntu 14.04 (LTS)

```
sudo apt-get update
sudo apt-get install curl
curl -sSL https://get.docker.com/ | sh
[sudo] password for xiaodong:
.....
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 xiaodong
```

Remember that you will have to log out and back in for  
this to take effect!

```
sudo usermod -aG docker xiaodong
```

安装Docker

将用户加入docker组，以避免使用sudo，  
需要Logout才能生效

# Deploy Images to Docker

## ■ 使用docker load来导入容器镜像

```
xiaodong@ubuntu:~/Workspace_Docker/Images$ ls  
docker_ubuntu_14.04.tar  
xiaodong@ubuntu:~/Workspace_Docker/Images$ docker load < docker_ubuntu_14.04.tar  
xiaodong@ubuntu:~/Workspace_Docker/Images$ docker images  
REPOSITORY          TAG           IMAGE ID      CREATED        VIRTUAL SIZE  
ubuntu              14.04         b39b81afc8ca   7 months ago  188.3 MB
```

## ■ 使用docker pull从Docker Hub拉取镜像

```
xiaodong@ubuntu:~/Workspace_Docker/Images$ docker search mysql  
NAME                  DESCRIPTION           STARS  
OFFICIAL   AUTOMATED  
mysql                 MySQL is a widely used, open-source relati...  1006  
[OK]  
mysql/mysql-server     Optimized MySQL Server Docker images. Crea...  41  
[OK]  
xiaodong@ubuntu:~/Workspace_Docker/Images$ docker pull mysql  
Using default tag: latest  
latest: Pulling from library/mysql  
.....
```

# Run a Container from a Image

xiaodong@ubuntu:~/Workspace_Docker\$ docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL
mysql	latest	7eee2d462c8f	2 weeks ago	283.6 MB
ubuntu	14.04	b39b81afc8ca	7 months ago	188.3 MB

- 从ubuntu:14.04镜像运行一个容器并进入交互模式

```
xiaodong@ubuntu:~/Workspace_Docker$ docker run -i -t --name myubuntu ubuntu:14.04
/bin/bash
root@72d09d5c22c2:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys
tmp  usr  var
```

# Run a Container from a Image

- 从ubuntu:14.04镜像启动一个容器并在后台运行

```
xiaodong@ubuntu:~/Workspace_Docker$ docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 2; done"
```

```
5265d540c7b37c6ec1b525e6a2e4e3f4d1bd366cfda924a69b793d10d3f29288
```

```
xiaodong@ubuntu:~/Workspace_Docker$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	
STATUS	PORTS	NAMES		
5265d540c7b3	ubuntu:14.04	/bin/sh -c 'while tr'	About a minute ago	Up
About a minute		reverent_kirch		

可以使用`docker logs ${container-id}`来从容器外面查看容器输出；`docker attach ${container-id}`可以连接到正在运行的容器终端，使用`ctrl-c`退出，`container`则停止运行，按`ctrl-p ctrl-q`可以退出到宿主机，容器依然在运行。

# Run a Container from a Image

- 从ubuntu:14.04镜像启动一个容器并在后台运行

```
xiaodong@ubuntu:~/Workspace_Docker$ docker run -d ubuntu:14.04 /bin/sh -c "while true; do echo hello world; sleep 2; done"
```

```
5265d540c7b37c6ec1b525e6a2e4e3f4d1bd366cfda924a69b793d10d3f29288
```

```
xiaodong@ubuntu:~/Workspace_Docker$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	
STATUS	PORTS	NAMES		
5265d540c7b3	ubuntu:14.04	/bin/sh -c 'while tr'	About a minute ago	Up
About a minute		reverent_kirch		

可以使用`docker logs ${container-id}`来从容器外面查看容器输出；`docker attach ${container-id}`可以连接到正在运行的容器终端，使用`ctrl-c`退出，`container`则停止运行，按`ctrl-p ctrl-q`可以退出到宿主机，容器依然在运行。

# Enter into a Container

## ■ 使用ssh登陆进容器

需要在容器中启动sshd，存在开销和攻击面增大的问题。同时也违反了Docker所倡导的一个容器一个进程的原则。

## ■ 使用nsenter、nsinit等第三方工具

需要额外学习和安装第三方工具。

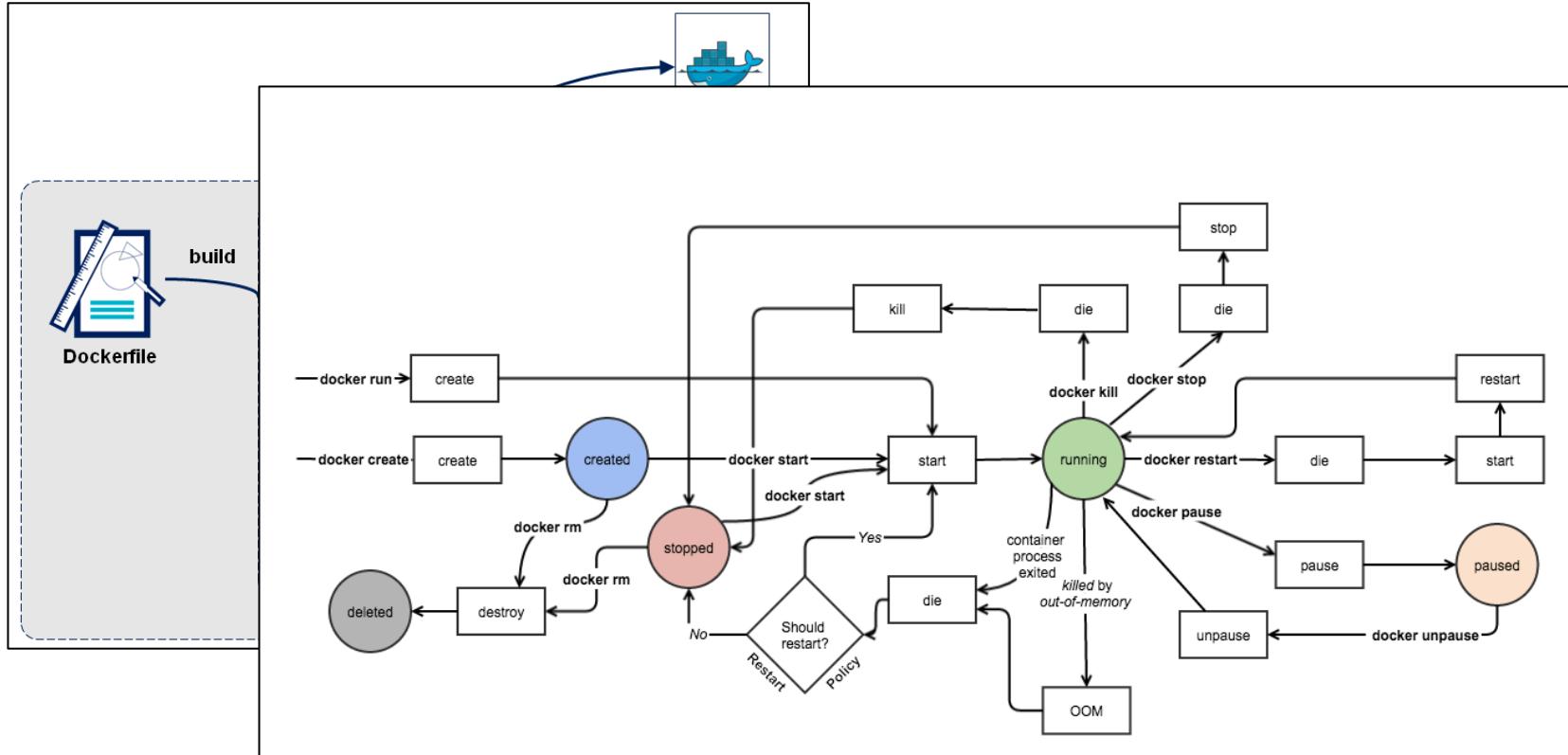
## ■ 使用docker本身提供的工具

- docker attach — attach到已经运行容器的stdin

- docker exec —

```
xiaodong@ubuntu:~/Workspace_Docker$ docker exec -it 526 /bin/bash
root@5265d540c7b3:/# ps aux
USER        PID %CPU %MEM      VSZ      RSS TTY      STAT START   TIME COMMAND
root          1  0.0  0.0    4444     656 ?      Ss 04:57   0:00 /bin/sh -c
while true; do echo hello world; sleep 2; done
root         21  0.0  0.1   18140    1948 ?      Ss 04:57   0:00 /bin/bash
root         83  0.0  0.0    4344     364 ?      S 04:59   0:00 sleep 2
root         84  0.0  0.1   15568    1120 ?      R+ 04:59   0:00 ps aux
```

# Docker Transition



# Docker命令行

## 常用Docker命令

---

- 容器生命周期管理 docker [run|start|stop|restart|kill|rm|pause|unpause]
- 容器操作运维 docker [ps|inspect|top|attach|events|logs|wait|export|port]
- 容器rootfs命令 docker [commit|cp|diff]
- 镜像仓库 docker [login|pull|push|search]
- 本地镜像管理 docker [images|rmi>tag|build|history|save|import]
- 其他命令 docker [info|version]

# Libcontainer

Libcontainer是Docker中用于容器管理的包，基于Go语言实现，通过管理namespaces、cgroups、capabilities以及文件系统来进行容器控制。可以使用Libcontainer创建容器，并对容器进行生命周期管理。

Docker是一款基于LXC的开源容器管理引擎。把LXC复杂的容器创建与使用方式简化为Docker自己的一套命令体系。随着Docker的不断发展，它开始有了更为远大的目标，那就是反向定义容器的实现标准，将底层实现都抽象化到Libcontainer的接口。这就意味着，底层容器的实现方式变成了一种可变的方案，无论是使用namespace、cgroups技术抑或是使用systemd等其他方案，只要实现了Libcontainer定义的一组接口，Docker都可以运行。这也为Docker实现全面的跨平台带来了可能。

Linux Container

**LXC**

# Install LXC on Ubuntu 14.04 (LTS)

```
sudo apt-get install lxc  
lxc-checkconfig
```

安装LXC用户空间工具

检查当前Linux内核支持LXC的情况

```
xiaodong@ubuntu:/var/lib$ lxc-checkconfig  
Kernel configuration not found at /proc/config.gz; searching...  
Kernel configuration found at /boot/config-3.13.0-63-generic  
--- Namespaces ---  
Namespaces: enabled  
Utsname namespace: enabled  
Ipc namespace: enabled  
Pid namespace: enabled  
User namespace: enabled  
Network namespace: enabled  
Multiple /dev/pts instances: enabled  
--- Control groups ---  
Cgroup: enabled  
Cgroup clone_children flag: enabled  
Cgroup device: enabled  
Cgroup sched: enabled  
Cgroup cpu account: enabled  
Cgroup memory controller: enabled  
Cgroup cpuset: enabled  
--- Misc ---  
Veth pair device: enabled  
Macvlan: enabled  
Vlan: enabled  
File capabilities: enabled  
Note : Before booting a new kernel, you can check its configuration  
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig
```

# Install LXC on Ubuntu 14.04 (LTS)

```
ifconfig
```

```
lxcbr0 Link encap:Ethernet HWaddr 6a:24:5e:5c:44:51  
inet addr:10.0.3.1 Bcast:10.0.3.255 Mask:255.255.255.0  
inet6 addr: fe80::6824:5eff:fe5c:4451/64 Scope:Link  
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1  
RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
TX packets:67 errors:0 dropped:0 overruns:0 carrier:0  
collisions:0 txqueuelen:0  
RX bytes:0 (0.0 B) TX bytes:9436 (9.4 KB)
```

LXC的默认网桥接口（lxcbr0）已自动创建（已在/etc/lxc/default.conf中配置）

```
cat /etc/lxc/default.conf
```

```
lxc.network.type = veth  
lxc.network.link = lxcbr0  
lxc.network.flags = up  
lxc.network.hwaddr = 00:16:3e:xx:xx:xx
```

# Install LXC on Ubuntu 14.04 (LTS)

```
ls /usr/share/lxc/templates
```

```
lxc-alpine lxc-busybox lxc-debian lxc-gentoo lxc-oracle lxc-ubuntu  
lxc-altninux lxc-centos lxc-download lxc-openmandriva lxc-plamo lxc-ubuntu-cloud  
lxc-archlinux lxc-cirros lxc-fedora lxc-opensuse lxc-sshd
```

创建某个特定目标环境的LXC容器需要相应的LXC模板，Ubuntu上的LXC用户空间工具随带一系列预先准备好的LXC模板在/usr/share/lxc/templates目录下

```
sudo lxc-create -n lxc -t ubuntu
```

```
Creating SSH2 RSA key; this may take some time ...  
Creating SSH2 DSA key; this may take some time ...  
Creating SSH2 ECDSA key; this may take some time ...  
Creating SSH2 ED25519 key; this may take some time ...  
update-rc.d: warning: default stop runlevel arguments (0 1 6) do not match ssh Default-Stop values (none)  
invoke-rc.d: policy-rc.d denied execution of start.
```

```
Current default time zone: 'America/Los_Angeles'  
Local time is now:  Fri Sep  4 08:13:28 PDT 2015.  
Universal Time is now: Fri Sep  4 15:13:28 UTC 2015.
```

```
##  
# The default user is 'ubuntu' with password 'ubuntu'!  
# Use the 'sudo' command to run tasks as root in the container.  
##
```

创建Ubuntu容器， 默认创建与本地主机同一版本号和同一架构的最小Ubuntu安装系统

lxc容器创建后， 默认的登录信息可以使用

# Install LXC on Ubuntu 14.04 (LTS)

```
ls /var/lib/lxc/<container-name>
```

```
root@ubuntu:/var/lib/# ls /var/lib/lxc/lxc/
config fstab rootfs
root@ubuntu:/var/lib# ls /var/lib/lxc/lxc/rootfs/
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
```

容器存储在`/var/lib/lxc/<container-name>`, 根文件系统则位于`/var/lib/lxc/<container-name>/rootfs`

```
sudo lxc-ls -f
```

列出主机上的LXC容器列表

```
xiaodong@ubuntu:/var/lib$ sudo lxc-ls -f
NAME STATE IPV4 IPV6 AUTOSTART
-----
lxc STOPPED - - NO
```

# Install LXC on Ubuntu 14.04 (LTS)

```
sudo lxc-start -n lxc
```

```
xiaodong@ubuntu:/var/lib$ sudo lxc-start -n lxc
<4>init: plymouth-upstart-bridge main process (5) terminated with status 1
```

```
...
* Starting Signal sysvinit that virtual filesystems are mounted ...done.
* Starting Signal sysvinit that virtual filesystems are mounted ...done.
* Starting Signal sysvinit that local filesystems are mounted ...done.
* Starting configure network device security ...done.
```

```
...
Ubuntu 14.04.3 LTS lxc console
```

```
lxc login: ubuntu
```

```
Password:
```

```
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-63-generic x86_64)
```

```
* Documentation: https://help.ubuntu.com/
```

The programs included with the Ubuntu system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/\*/\*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by  
applicable law.

```
ubuntu@lxc:~$
```

启动名为lxc的容器，赋予一个IP地址并  
自动连接到LXC的内部网桥lxcbr0

```
brctl show lxcbr0
```

```
xiaodong@ubuntu:~$ brctl show lxcbr0
bridge name  bridge id   STP enabled  interfaces
lxcbr0      8000.fe9dc1f95a06  no        vethU3B4J3
```

# Install LXC on Ubuntu 14.04 (LTS)

```
sudo lxc-start -n lxc -d
```

“-d”选项将容器作为守护程序来启动

```
sudo lxc-console -n lxc
```

访问容器控制台，键入`<Ctrl+a q>`组合键，  
退出控制台

```
sudo lxc-stop -n lxc  
sudo lxc-destroy -n lxc
```

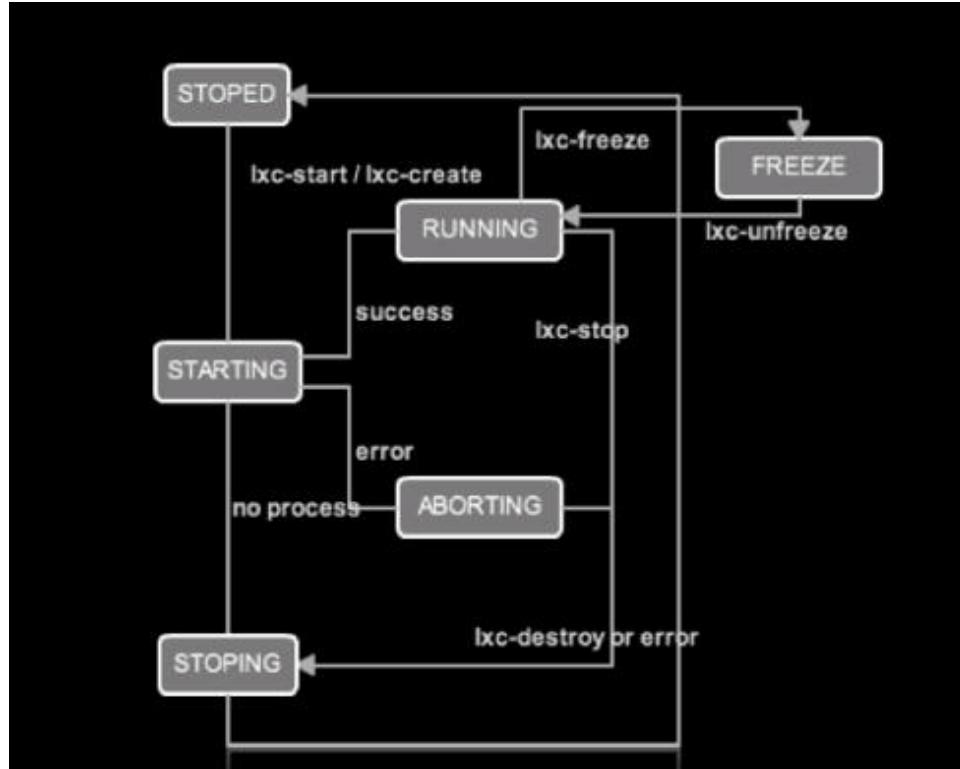
停止容器  
销毁容器

```
sudo lxc-stop -n lxc  
sudo lxc-clone -o -n lxc
```

克隆容器

```
xiaodong@ubuntu:/var/lib$ sudo lxc-clone -o lxc -n ubuntu  
Created container ubuntu as copy of lxc
```

# LXC Lifecycle



# Docker API

By default, Docker daemon listens on `unix:///var/run/docker.sock`, you can use the following command to get the image list (JSON format).

```
echo -e "GET /images/json HTTP/1.0\r\n" | nc -U /var/run/docker.sock
HTTP/1.0 200 OK
Content-Type: application/json
Server: Docker/1.8.2 (linux)
Date: Wed, 16 Sep 2015 14:08:10 GMT
Content-Length: 530

[ {"Id": "7eee2d462c8f6ffacfb908cc930559e21778f60afdb2d7e9cf0f3025274d7ea8", "ParentId": "15a3cddfc178c4dbaa8f56142d4eebef6d22a3cd1842820844cf815992fe5a13", "RepoTags": ["mysql:latest"], "RepoDigests": [], "Created": 1440453313, "Size": 0, "VirtualSize": 283598751, "Labels": {}}, {"Id": "b39b81afc8cae27d6fc7ea89584bad5e0ba792127597d02425eaee9f3aaaa462", "ParentId": "615c102e2290b70d38d89c03a1ad263da8bd8b05fb7fc8479174e5fd2215520e", "RepoTags": ["ubuntu:14.04"], "RepoDigests": [], "Created": 1421443530, "Size": 0, "VirtualSize": 188305056, "Labels": null} ]
```

# Docker API

But, for the purpose of interaction using HTTP protocol, we must modify the **/etc/default/docker** file and add the red line.

```
# Docker Upstart and SysVinit configuration file

# Customize location of Docker binary (especially for development testing) .
#DOCKER="/usr/local/bin/docker"

# Use DOCKER_OPTS to modify the daemon startup options.
#DOCKER_OPTS="--dns 8.8.8.8 --dns 8.8.4.4"
DOCKER_OPTS="-H=unix:///var/run/docker.sock -H=0.0.0.0:6732"
# If you need Docker to use an HTTP proxy, it can also be specified here.
#export http_proxy="http://127.0.0.1:3128/"

# This is also a handy place to tweak where Docker's temporary files go.
#export TMPDIR="/mnt/bigdrive/docker-tmp"
```

```
sudo service docker restart
```

# Docker API

## List all the running containers

```
xiaodong@ubuntu:~$ curl http://localhost:6732/containers/json | jq .
[
  {
    "HostConfig": {
      "NetworkMode": "default"
    },
    "Id": "5265d540c7b37c6ec1b525e6a2e4e3f4d1bd366cfda924a69b793d10d3f29288",
    "Names": [
      "/reverent_kirch"
    ],
    "Image": "ubuntu:14.04",
    "Command": "/bin/sh -c 'while true; do echo hello world; sleep 2; done'",
    "Created": 1441771858,
    "Ports": [],
    "Labels": {},
    "Status": "Up 13 minutes"
  }
]
```

Query  
Parameters:

- all
- limit
- since
- before
- size

# Docker API

## Create a container

```
xiaodong@ubuntu:~$ curl --request POST --header "Content-Type:  
application/json" --data '{"Hostname": "Machine-JSON", "Image":  
"ubuntu:14.04", "Tty": true, "OpenStdin": true, "Cmd": ["/bin/bash"]}'  
http://localhost:6732/containers/create  
{"Id":"2b11bc47057a536bc8031a5291d60ef2eedebb66260e3ca363eea4af93e20090","War  
nings":null}  
xiaodong@ubuntu:~$ curl --request POST  
http://localhost:6732/containers/2b11bc47057a536bc8031a5291d60ef2eedebb66260e  
3ca363eea4af93e20090/start
```

## Start the container (use container id)

```
xiaodong@ubuntu:~$ curl --request POST  
http://localhost:6732/containers/2b11bc47057a536bc8031a5291d60ef2eedebb66260e
```

# Docker API

You can put json strings into a file (e.g. docker.json)

```
xiaodong@ubuntu:~$ cat docker.json
{
  "Hostname": "Machine-JSON2",
  "Image": "ubuntu:14.04",
  "Tty": true,
  "OpenStdin": true,
  "Cmd": ["/bin/bash"]
}

xiaodong@ubuntu:~$ curl --request POST --header "Content-Type:
application/json" --data-binary @docker.json
http://localhost:6732/containers/create
```

# Docker - Python

docker-py, an API client for docker written in Python.

```
xiaodong@ubuntu:~$ pip install docker-py -user
xiaodong@ubuntu:~$ ls .local/lib/python2.7/site-packages/
backports
backports.ssl_match_hostname-3.4.0.2.dist-info
docker
docker_py-1.4.0.dist-info
requests
requests-2.7.0.dist-info
websocket
websocket_client-0.32.0.dist-info
```

# References

1. <https://en.wikipedia.org/wiki/Virtualization>
2. <https://linuxcontainers.org>
3. Docker容器内多进程的管理方案. <http://www.tuicool.com/articles/2677VfR>
4. 如何在Docker容器内外拷贝数据. <http://blog.csdn.net/yangzhenping/article/details/43667785>
5. 深入浅出Docker（一）：Docker核心技术预览<http://www.infoq.com/cn/articles/docker-core-technology-preview/>