**云计算架构实战**

V-0.1.0

修改历史

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本 | 描述 | 作者 |
| 2015-11-26 | V-0.1.0 | 创建文档 | 刘彬 |
|  |  |  |  |
|  |  |  |  |

目录

[1. 大数据 5](#_Toc4794)

[1.1. 术语解释 6](#_Toc10640)

[1.1.1. 什么是数据挖掘 6](#_Toc10467)

[1.1.2. 数据挖掘要解决什么问题 6](#_Toc6810)

[1.1.3. 数据挖掘的任务 6](#_Toc1233)

[1.2. Hadoop 6](#_Toc7364)

[1.3. NoSql 7](#_Toc21268)

[1.3.1. leveldb 7](#_Toc28634)

[1.3.2. mongodb 7](#_Toc27607)

[1.3.3. redis 7](#_Toc3235)

[1.3.4. cassandra 7](#_Toc2280)

[1.4. Kettle 7](#_Toc24909)

[1.5. mahout 7](#_Toc29931)

[1.6. Lustre  7](#_Toc5723)

[1.7. OCFS2  7](#_Toc17695)

[1.8. mpfs 8](#_Toc18375)

[1.9. Gluster 8](#_Toc31333)

[2. 虚拟化 8](#_Toc15481)

[2.1. Exsi 8](#_Toc2920)

[2.2. Xen 8](#_Toc13895)

[2.3. KVM 11](#_Toc10518)

[2.3.1. KVM环境构建 11](#_Toc1812)

[2.3.2. KVM源代码 14](#_Toc22473)

[2.3.3. Qemu-kvm 19](#_Toc15308)

[2.3.4. libvirt 19](#_Toc2769)

[2.3.5. kvm与qemu的关系 19](#_Toc29768)

[2.3.6. kvm 存储栈分析  20](#_Toc7488)

[2.4. VirtualBox 20](#_Toc30570)

[2.5. Hyper-V 20](#_Toc6697)

[3. 数据分析 20](#_Toc19687)

[3.1. WEB爬虫 20](#_Toc6059)

[3.2. Heritrix  20](#_Toc12956)

[3.3. 搜索引擎 20](#_Toc17277)

[3.4. teradata 20](#_Toc12650)

[3.5. sybase 20](#_Toc29925)

[3.6. greenplum 20](#_Toc30172)

[3.7. hana 20](#_Toc30514)

[4. 云存储 20](#_Toc29486)

[4.1. 存储系统设计原理 20](#_Toc6450)

[4.2. 基本存储设备 20](#_Toc19313)

[4.3. 磁盘阵列 21](#_Toc12608)

[4.4. 对象存储系统研究 21](#_Toc3549)

[4.5. 可靠性和可用性研究 21](#_Toc733)

[4.6. 存储安全性研究 21](#_Toc9353)

[4.7. SAN存储 21](#_Toc6640)

[4.8. NAS存储 21](#_Toc21801)

[4.9. Sheepdog 21](#_Toc4383)

[4.10. KFS 24](#_Toc5255)

[4.11. CEPH 24](#_Toc9188)

[5. SDN 24](#_Toc10945)

[6. 云平台 24](#_Toc12867)

[6.1. cloudfoudry 24](#_Toc22174)

[6.2. opennebula 24](#_Toc19287)

[6.3. cloudstack 24](#_Toc20605)

[6.3.1. Api 认证 25](#_Toc15624)

[6.3.2. jclouds 25](#_Toc20895)

[6.4. openstack 26](#_Toc23206)

[6.4.1. Conceptual Diagram（概念图） 27](#_Toc10972)

[6.4.2. Logical Diagram（逻辑图） 28](#_Toc31047)

[6.4.3. Openstack Swift 28](#_Toc3599)

[6.5. Docker 29](#_Toc22543)

[6.5.1. 常用组件介绍 29](#_Toc1969)

[6.5.2. 容器化的综述 29](#_Toc7125)

[6.5.3. 服务发现和分布式配置存储 29](#_Toc21117)

[6.5.4. 网络与通信 29](#_Toc185)

[6.5.5. 集群 29](#_Toc15641)

[6.5.6. Flynn 31](#_Toc14763)

[6.5.7. Deis 31](#_Toc23058)

[6.5.8. Dokku 31](#_Toc29941)

[6.5.9. CoreOS 31](#_Toc13488)

[6.5.10. Fig 32](#_Toc30276)

[6.5.11. Atomic 32](#_Toc22890)

[6.5.12. Drone 32](#_Toc31172)

[6.5.13. Kubernetes 32](#_Toc14085)

[6.5.14. Panamax 32](#_Toc28999)

[6.5.15. Swarm 33](#_Toc17826)

[6.5.16. Boot2Docker 36](#_Toc29593)

[6.5.17. DockerUI 36](#_Toc2891)

[6.5.18. Cgroups 38](#_Toc14182)

[6.5.19. LXC 39](#_Toc4531)

[6.5.20. AUFS  39](#_Toc32352)

[6.6. Mesos 43](#_Toc16209)

[6.6.1. Marathon 43](#_Toc14982)

[6.6.2. Bamboo 43](#_Toc22014)

[6.6.3. Chronos 43](#_Toc18110)

[6.7. AWS 43](#_Toc11188)

[6.7.1. aws sdk 43](#_Toc5408)

[6.7.2. 实例支持REST的action 45](#_Toc5672)

[6.7.3. jclouds 45](#_Toc2203)

[6.8. Aliyun 45](#_Toc2131)

[6.9. 云平台架构 46](#_Toc7399)

[6.9.1. 私有云 46](#_Toc6439)

[6.9.2. 混合云 46](#_Toc22238)

[6.9.3. 专有云 46](#_Toc9475)

[6.9.4. 桌面云 46](#_Toc2219)

[6.9.5. overview 46](#_Toc1949)

[7. 资源监控 47](#_Toc4215)

[7.1. zabbix 47](#_Toc6788)

[7.2. nagios 47](#_Toc11645)

[7.3. ganglia 92](#_Toc17509)

[8. 自动化管理 92](#_Toc9891)

[8.1. Cmdb 92](#_Toc1972)

[8.2. Puppet 92](#_Toc5908)

[8.3. chef 96](#_Toc13164)

[9. SOA架构与实施 97](#_Toc6251)

[9.1. HTTP 97](#_Toc12035)

[9.2. SOA基本概念与术语 97](#_Toc32623)

[9.3. REST 98](#_Toc25283)

[9.3.1. JAX-RS规范 98](#_Toc18286)

[9.4. 微服务 99](#_Toc2087)

[9.4.1. IT基础架构库(Information Technology Infrastructure Library) ITIL 99](#_Toc27068)

[9.4.2. 基于J2EE构建 101](#_Toc18930)

[9.4.3. 基于GO构建 102](#_Toc2529)

[9.4.4. 基于消息传递 102](#_Toc6673)

[9.5. UML 103](#_Toc25721)

[9.5.1. 模型是什么 103](#_Toc5006)

[9.5.2. 为什么建模 103](#_Toc7636)

[9.5.3. 建模原理 103](#_Toc12575)

[9.6. 设计模式 103](#_Toc14041)

[9.6.1. 创建模式 104](#_Toc9486)

[9.6.2. 结构模式 104](#_Toc25015)

[9.6.3. 行为模式 104](#_Toc17811)

[9.7. 反模式 105](#_Toc32105)

[9.8. 持续交付 106](#_Toc22746)

[9.8.1. 软件交付的问题 107](#_Toc30236)

[9.8.2. 配置管理 107](#_Toc9796)

[9.8.3. 持续集成 107](#_Toc31219)

[9.8.4. 测试策略的实现 107](#_Toc7845)

[9.8.5. 部署流水线解析 107](#_Toc11455)

[9.8.6. 构建与部署脚本化 107](#_Toc15251)

[9.8.7. 提交阶段 107](#_Toc6818)

[9.8.8. 自动化验收测试 107](#_Toc30205)

[9.8.9. 非功能需求测试 107](#_Toc26819)

[9.8.10. 应用程序的部署与发布 107](#_Toc30369)

[10. 解决方案 107](#_Toc31848)

# 大数据

**人工智能**

**机器学习**

**模式识别**

**统计学**

**数据挖掘**

**数据库技术、并行计算、分布式计算**

数据挖掘学科之间的交叉

## 术语解释

* R、SAS、SPSS、S-PLUS、MATLAB中的一种数据分析环境
* OLAP技术，包括关系型OLAP技术，以及立方、切片等多维OLAP技术
* 流式处理的相关技术原理，了解CEP的基本原理
* 数据呈现技术（特别是报表设计）
* 运用回归、分类、聚类、关联规则、图计算的常见算法
* 数学模型和统计模型，熟练掌握常用数学建模方法和统计分析方法
* 静态页面生成技术、Google Web Toolkit
* 分布式列存数据库、ETL、数据质量、商业智能应用构建工具、大规模数据云计算
* 数据库基础、具有数据库索引优化、查询优化和存储优化经验
* 大规模系统的负载均衡、网络存储、网络安全、数据库高可用设计及性能评估机制
* 掌握主流应用服务器框架、分布式数据库、缓存、文件系统、消息系统等技术
* OOA,OOP,OOD的软件编程设计
* android系统的Custom Building System

### 什么是数据挖掘

1. 【数据挖掘是在大型数据存储库中，自动地发现有用信息的过程】
2. 【数据挖掘技术用来探查大型数据库，发现先前未知的用用模式】
3. 【数据挖掘又是数据库知识发现（KDD）的重要组成部分】

### 数据挖掘要解决什么问题

伸缩性

高纬性

异种数据和复杂数据

数据的所有权与分布

非全统分析

### 数据挖掘的任务

预测任务：任务的目标是根据其他属性的值，预测特定属性的值。被预测的属性一般称目标变量或者因变量，因而用来做预测的属性称为明变量或自变量。

描述任务：其目标是导出概括数据中潜在联系的模式(相关、趋势、聚类、轨迹和异常

## Hadoop

## NoSql

Nosql简介<http://www.nosql-database.org/>

### leveldb

<https://code.google.com/p/leveldb/>

### mongodb

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

### redis

<http://www.redis.io/>

### cassandra

## Kettle

<http://www.weka.nl/>

## mahout

数据切分是从逻辑上或物理上将数据分割成数个更好维护或访问的部分，同时很多流行的开源关系型数据库都支持分片（比如MySQL Partitioning及Postgres Partitionging）。

## Lustre

<http://wiki.lustre.org/index.php/Main_Page>

## OCFS2

<https://oss.oracle.com/projects/ocfs2/>

## mpfs

<http://www.mpfs.de/>

## Gluster

# 虚拟化

## Exsi

## Xen

Xen(发音为Zen)是由XenSource所管理的一个开源GPL项目。Xen是openSuse和Novell主要支持的虚拟化技术。

Xen服务器上的虚拟机以两种模式运行：全虚拟化或半虚拟化。

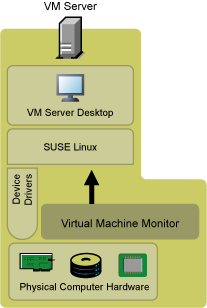
**全虚拟化：**是一种完全模拟所有硬件设备的虚拟化模式。由于VMM（虚拟机监控系统）模拟了所有的计算机设备使得操作系统好像独自占用了所有计算机的资源，所以尽管需要特定的计算机硬件，大多数操作系统都可以以全虚拟化模式运行。完全模拟计算机硬件需要消耗虚拟机服务器更多地CPU资源。所以全虚拟化模式操作系统运行速度会比较慢。

**半虚拟化：**是一种选择性的模拟硬件设备的虚拟化模式。以半虚拟化模式运行的半虚拟化操作系统并不需要完全模拟硬件，因此只需付出更少的管理代价。例如，VMware系统运行时不需要模拟显卡，因此虚拟机服务器就不用模拟视频数据了。所以，以半虚拟化模式运行的操作系统消耗更少的CPU资源并且执行地更好。它也并不需要特定的计算机硬件。

虚拟机监控系统（VMM）在服务器硬件和Suse linux系统内核之间运行。计算机启动时会首先加载VMM,然后以特权模式启动虚拟机服务器。特权模式指的是虚拟机服务器可以创建并控制虚拟机，而且还可对计算机硬件进行直接存取。你需要设置虚拟机服务器与本地设备驱动程序来匹配实际的计算机设备。例如，如果计算机有一个物理网卡e1000，就要设置虚拟机服务器去加载运行e1000的Suse linux驱动程序。

SUSE®Linux Enterprise Server 11 Service Pack 3 (SP3) 随附最新的开放源代码虚拟化技术 Xen 和 KVM。有了这些超级管理程序，SUSE Linux Enterprise Server 就可用于在单个物理系统上安装、监视和管理多个虚拟机 (VM)。

目前Xen开源社区开发和维护着涉及软件层的虚拟机监控程序(VMM) 和 hypervisor，同时它也提供功能性的虚拟机服务器。

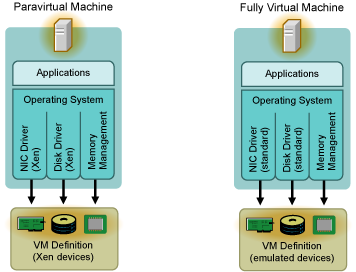
[](https://cn.opensuse.org/File:Xen_arch_10_a.png)

***Figure 2-1*** *虚拟机与设备驱动*

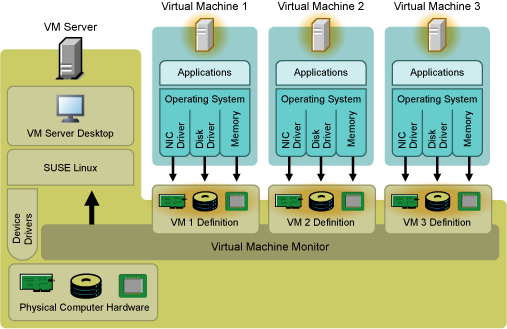
如何定义一个xen虚拟机服务器(Defining a VM),你可以在存放虚拟机镜像的虚拟机服务器上定义一个虚拟机，在配置文件/etc/xen/vm/vm\_name里面去定义。配置文件里面定义了虚拟资源，比如在虚拟机上安装和启动操作系统时你所看到的CPU、内存、网卡及块设备。例如如下表格中定义xen虚拟机服务器的配置。

|  |  |
| --- | --- |
| name="sles11"  description="None"  uuid="a3d58b25-825c-0aa0-235e-2908d65bbcee"  memory=2048  maxmem=2048  vcpus=4  on\_poweroff="destroy"  on\_reboot="restart"  on\_crash="destroy"  localtime=0  keymap="en-us"  builder="hvm"  device\_model="/usr/lib/xen/bin/qemu-dm"  kernel="/usr/lib/xen/boot/hvmloader"  boot="c"  disk=[ 'file:/var/lib/xen/images/sles11/disk0.raw,hda,w', ]  vif=[ 'mac=00:16:3e:21:69:d0,bridge=br0,type=netfront', ]  stdvga=0  vnc=1  vncunused=1  viridian=0  acpi=1  pae=1  serial="pty" | [Xen arch 04 a.png](https://cn.opensuse.org/File:Xen_arch_04_a.png) |

在全虚拟化和半虚拟化两种模式下，虚拟机操作系统都是通过设备驱动与VMM相结合。在全虚拟化模式下，操作系统通过本地标准模拟设备的设备驱动来实现，例如AMD PCNet或者NE2000网卡，IDE磁盘驱动器和VGA显卡。在半虚拟化模式下，含有专有设备驱动（叫Xen驱动)的VM-aware操作系统通过VMM和VM服务器与计算机的物理设备通信。

[](https://cn.opensuse.org/File:Xen_arch_03_a.png)

比如说，如果一个以全虚拟化模式运行的虚拟机操作系统想要保存一个文件到它的20G虚拟磁盘上，操作系统就要通过设备驱动给VMM传送请求。VMM知道了它的500G物理磁盘的哪部分可以给虚拟机进行存取以后，它就会给虚拟机服务器下达指令。虚拟机服务器回去访问磁盘启动器并把文件写入预先确定好的磁盘区域里。 只要有足够的计算能力和可用的计算机资源，你可以再一台虚拟机服务器上创建并同时运行任意多个虚拟机。每一个虚拟机都相互独立地通过VMM和虚拟机服务器平台去使用虚拟或模拟的CPU、内存、块设备和网络资源。

[](https://cn.opensuse.org/File:Xen_arch_11_a.png)

## KVM

KVM（名称来自英语：Kernel-based Virtual Machine的缩写，即基于内核的虚拟机），是一种用于Linux内核中的虚拟化基础设施，可以将Linux内核转化为一个hypervisor。KVM在2007年2月被导入Linux 2.6.20核心中。

### KVM环境构建

我们知道，KVM从诞生第一天就是需要硬件虚拟化扩展的支持，所以这里需要特别讲解一下硬件系统的配置。

KVM最初始的开发是基于**x86**和**x86-64**处理器架构上Linux系统进行的，目前，KVM也被移植到多种不同处理器架构之上，包括：Intel和HP的IA64（安腾）架构、AIM联盟（Apple–IBM–Motorola）的PowerPC架构、IBM的S/390架构、ARM架构（2012年开始[注1]）。其中，在x86-64上面的功能的支持是最完善（主要原因是由于Intel/AMD的x86-64架构在桌面和服务器市场上的主导地位及其架构的开放性，以及它的开发者众多），本书也是采用基于Intel x86-64架构的处理器作为基本的硬件环境[注2]。

在x86-64架构的处理器中，KVM必须的硬件虚拟化扩展分别为：英特尔的虚拟化技术（Intel VT）和AMD的AMD-V技术。其中，英特尔在2005年11月发布的奔腾四处理器（型号：662 和672）第一次正式支持VT技术，而在2006年5月AMD也发布了支持AMD-V的处理器。现在比较流行的针对服务器和桌面的Intel处理器多数都是支持VT技术的，下面主要解释英特尔的VT（Virtualization Technology）技术相关的硬件设置。

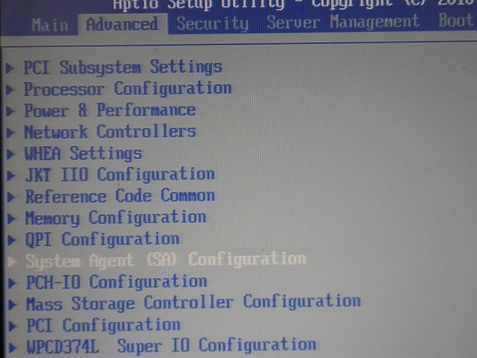
首先处理器（CPU）在硬件上支持了VT技术，还需要在BIOS中将其功能打开，KVM才能使用到。目前，多数流行的服务器和部分桌面处理器的BIOS都是默认将VT打开了的。

VT的选项，一般在BIOS中的“Advanced à Processor Configuration”这里来查看和设置，它的标识一般为“Intel(R) Virtualization Technology”或者“Intel VT”等类似的文字说明。

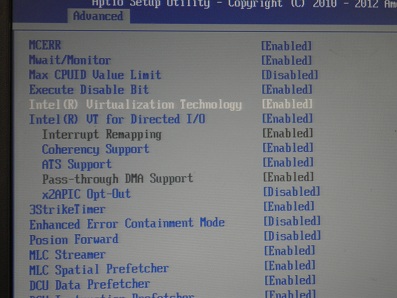
除了必须的处理器虚拟化扩展支持以外，如果服务器芯片还支持VT-d，也建议在BIOS中将其打开，因为后面的一些相对高级的设备直接分配功能会需要硬件VT-d技术的支持。VT-d（Virtualization Technology for Directed I/O）是对设备I/O的虚拟化硬件支持，在BIOS的位置可能为“Advanced à Processor Configuration”或者“Advanced à System Agent (SA) Configuration”，它一般在BIOS中的标志一般为“Intel(R) VT for Directed I/O”或者“Intel VT-d”等。

下面以一台Intel Romley-EP平台的服务器为例来说明BIOS中的设置。

1. BIOS中的Advanced选项：



2. BIOS中的VT和VT-d选项，如图3-2所示：



对于不同平台或不同厂商的BIOS，VT和VT-d等设置位置可能是不一样的，需要根据实际的硬件情况和BIOS中选项来灵活设置。

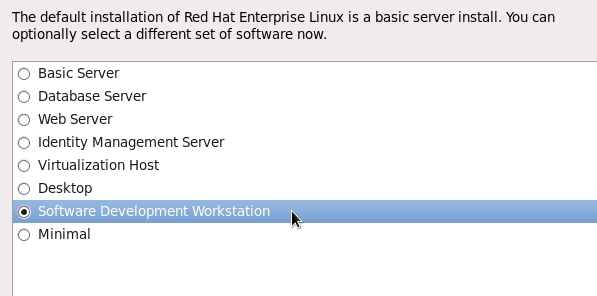
设置好了VT和VT-d的相关选项，保存BIOS的设置并退出，系统重启后会生效。

KVM是基于内核的虚拟化技术，要运行KVM虚拟化环境，安装一个Linux操作系统的宿主机（Host）是必需的。由于Redhat公司是目前对KVM项目投入最多的企业之一，从RHEL6开始，其系统自带的虚拟化方案就采用了KVM，而且RHEL（Red Hat Enterprise Linux）也是最流行的企业级Linux发行版之一，所以本节选用RHEL来讲解Linux系统的安装步骤和过程，并且本章后面的编译和运行都是在这个系统之上的。

当然，KVM作为流行的开源虚拟机之一，它可以在绝大多数流行的Linux系统上编译和运行，所以你依然可以选择RHEL之外的其他Linux发行版，比如CentOS、Fedora、Ubuntu、Debian、OpenSuse等系统都是不错的选择。

本节内容基于目前最新的RHEL版本——RHEL6.3 Server版的系统来简单介绍，普通Linux安装的基本过程这里就不再详细描述，本节主要说明一下安装过程中一些值得注意的地方。

在选择哪种类型的服务器时，选择“Software Development Workstation”即可（如图3-3所示），然后选中当前页面的“Customize now”，点击“Next”按钮进入下一步去选择具体需要安装的组件和设置所需要的各个RPM包。

[](http://smilejay.com/wp-content/uploads/2012/06/3-3.jpg)

RHEL6.3安装中选择服务器类型

在选择了“Software Development Workstation”之后，在具体组件的选择界面，就可以看到已经默认选择好了很多的组件（如图3-4所示），这里主要需要检查一下Development这个选项这里已默认勾选了很多的开发组件。其中，Development选项中的Development tools和Additional Development这两个组件最好被选中，因为在本书中的编译以及其他实验中可能会用到，其中包括一些比较重要的软件包，比如：gcc、git、make等（它们一般被默认选中）。另外可以点击下方的“Optional packages”按钮根据需要选择一些可选的软件包。图3-4中的Virtualization选项中，我们可以先不选中其中的任何组件，因为本章会自己编译KVM和qemu-kvm，而在后面第7章讲发行版中的KVM时，一般会安装Virtualization组件并使用发行版中自带的KVM Virtualization功能。



Software Development Workstation 的具体组件的选择

然后，继续进行后面安装流程，可以安装相应的软件包，安装过程的一个快照如图3-5所示。



RHEL6.3安装过程快照

在安装完所有软件包后，系统会提示安装完成需要重启，重启系统之后，即可进入到RHEL6.3系统中。至此，Linux系统就安装完毕了，这就是在本书中作为宿主机（Host）的操作系统，后面的编译和做实验都是在这个宿主机上进行的（当然，我们会使用本章中讲述的自己编译的kernel和qemu-kvm来做实验）。

### KVM源代码

KVM作为Linux kernel中的一个module而存在，是从Linux 2.6.20版本开始被完全正式加入到内核的主干开发和正式发布代码中。所以，只需要下载2.6.20版本之后Linux kernel的代码即可。当然，如果是学习KVM，推荐使用最新正式发布或者开发中的kernel版本，如果是实际部署到生产环境中，还需要自己选择适合的稳定版本进行足够的功能和性能测试。如果你想使用最新的处于开发中KVM代码，你需要自己下载KVM的代码仓库，本节中就是这样来讲解的。

总的来说，下载最新KVM源代码，主要有如下三种方式：

* 下载KVM项目开发中的代码仓库kvm.git
* 下载Linux内核的代码仓库linux.git
* 打包下载Linux内核的源代码（Tarball[注3]格式）

1.  下载kvm.git

KVM项目的代码是托管在Linux内核官方源码网站[http://git.kernel.org](http://git.kernel.org/)上的，可以到上面去查看和下载。该网页上virt/kvm/kvm.git 即是KVM项目的代码，它是最新的功能最丰富的KVM源代码库（尽管并非最稳定的）。目前，kvm.git的最主要维护者（maintainer）是来自Redhat公司的Avi Kivity和Marcelo Tosatti。从<http://git.kernel.org/?p=virt/kvm/kvm.git>网页可以看到，kvm.git下载链接有如下3个URL可用于下载最新的kvm的开发代码仓库。

git://git.kernel.org/pub/scm/virt/kvm/kvm.git

http://git.kernel.org/pub/scm/virt/kvm/kvm.git

https://git.kernel.org/pub/scm/virt/kvm/kvm.git

这三个URL下载的内容都是完全一致的，根据自己实际情况选择一个下载即可。Linux内核相关的项目一般都使用Git[注4]作为源代码管理工具，KVM当然也是用Git管理源码的。可以使用git clone命令来下载kvm的源代码，也可以使用Git工具的其他命令对源码进行各种管理，这里不详述Git的各种命令，有兴趣的读者可以参考后面的附录“git 常用命令”。

kvm.git的下载方式和过程，为如下命令行所示：

[root@jay-linux ~] cd kvm\_demo

[root@jay-linux kvm\_demo]# git clone git://git.kernel.org/pub/scm/virt/kvm/kvm.git kvm.git

[root@jay-linux kvm\_demo]# cd kvm.git

[root@jay-linux kvm.git]# pwd

2. 下载linux.git

Linux内核的官方网站为[http://kernel.org](http://kernel.org/)，其中源代码管理网为[http://git.kernel.org](http://git.kernel.org/)，可以在那里找到最新的linux.git代码。在源码管理网站上，我们看到有多个linux.git，我们选择Linus Torvalds[注5]的源码库（也即是Linux内核的主干）。在源码的网页<http://git.kernel.org/?p=linux/kernel/git/torvalds/linux.git> 中可以看到，其源码仓库可以也有如下3个链接可用：

git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

http://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

这三个URL中源码内容是完全相同的，可以用用git clone命令复制到本地，其具体操作方式与前一种（kvm.git）的下载方式完全一样。

3. 下载Linux的Tarball

在Linux官方网站（<http://kernel.org/>）上，也提供Linux内核的Tarball文件下载；除了在其首页上点击一些Tarball之外，也可以到如下网址下载Linux内核的各个版本的Tarball。

<ftp://ftp.kernel.org/pub/linux/kernel/>

<http://www.kernel.org/pub/linux/kernel/>

kernel.org还提供一种rsync的方式下载，此处不详细叙述，请参见其官网首页的提示。

以用wget下载linux-3.4.1.tar.gz 为例，有如下的命令行代码：

[root@jay-linuxkvm\_demo]# wget http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.4.1.tar.gz

[root@jay-linux kvm\_demo]# ls -l

[root@jay-linux kvm\_demo]# tar -zxf linux-3.4.1.tar.gz

4. 通过kernel.org的镜像站点下载：

由于Linux的源代码量比较大，如果只有美国一个站点可供下载，那么可能会速度较慢，服务器压力也较大。所以，kernel.org在世界上多个国家和地区都有一些镜像站点，而且，一些Linux开源社区的爱好者们也自发建立了不少kernel.org的一些镜像，在中国的镜像站点中，如下两个镜像站点推荐给大家下载Linux相关的代码及其他源码（本书写作之时，它们并没有提供git形式的代码仓库kvm.git或linux.git的下载），访问速度比较快。

其中一个是清华大学开源镜像站：<http://mirror.tuna.tsinghua.edu.cn/>；其中的如下链接地址与<http://www.kernel.org/pub/linux/kernel/>就是同步的，用起来比较方便。

<http://mirror.tuna.tsinghua.edu.cn/kernel/linux/kernel/>

另外一个推荐镜像是北京交通大学的一个开源镜像站，链接如下：

<http://mirror.bjtu.edu.cn/kernel/linux/kernel/>

另外，还有如下两个镜像站可以推荐给大家参考：

网易开源镜像站：<http://mirrors.163.com/>

搜狐开源镜像站：<http://mirrors.sohu.com/>

上面三种方式下载的源代码都是可以同样地进行配置和编译，本章中以开发中的最新源代码仓库kvm.git来讲解KVM的配置和编译等。KVM是作为Linux内核中的一个module存在的，而kvm.git是一个包含了最新的KVM模块开发中代码的完整的Linux内核源码仓库。它的配置方式，与普通的Linux内核配置完全一样，只是需要注意将KVM相关的配置选择为编译进内核或者编译为模块。

在kvm.git（Linux kernel）代码目录下，运行“make help”命令可以得到一些关于如何配置和编译kernel的帮助手册，如下面命令行所示：

[root@jay-linux kvm.git]# make help

Cleaning targets:

clean           – Remove most generated files but keep the config and

enough build support to build external modules

mrproper        – Remove all generated files + config + various backup files

distclean       – mrproper + remove editor backup and patch files

Configuration targets:

config          – Update current config utilising a line-oriented program

nconfig         – Update current config utilising a ncurses menu based program

menuconfig      – Update current config utilising a menu based program

xconfig         – Update current config utilising a QT based front-end

<!- 此处省略数十行帮助信息 ->

make W=n   [targets] Enable extra gcc checks, n=1,2,3 where

1: warnings which may be relevant and do not occur too often

2: warnings which occur quite often but may still be relevant

3: more obscure warnings, can most likely be ignored

Multiple levels can be combined with W=12 or W=123

Execute “make” or “make all” to build all targets marked with [\*]

For further info see the ./README file

对KVM或者Linux内核配置时常用的配置命令，对其如下的一些解释：

q   **make config**

基于文本的最为传统的也是最为枯燥的一种配置方式，但是它可以适用于任何情况之下，这种方式会为每一个内核支持的特性向用户提问，如果用户回答“y”，则把特性编译进内核；回答“m”，则它特性作为模块进行编译；回答“n”，则表示不对该特性提供支持。输入“？”则显示该选项的帮助信息，在了解之后再做决定处理该选项的方式；在回答每个问题前，必须考虑清楚，如果在配置过程中因为失误而给了错误的回答，就只能按“ctcl+c”强行退出然后重新配置了。

q   **make oldconfig**

make oldconfig和make config类似，但是它的作用是在现有的内核设置文件基础上建立一个新的设置文件，只会向用户提供有关新内核特性的问题，在新内核升级的过程中，make oldconfig非常有用，用户将现有的配置文件.config复制到新内核的源码中，执行make oldconfig，此时，用户只需要回答那些针对新增特性的问题。

q   **make silentoldconfig**

和上面make oldconfig一样，但在屏幕上不再出现已在.config中配置好的选项。

q   **make menuconfig**

基于终端的一种配置方式，提供了文本模式的图形用户界面，用户可以通过光标移动来浏览所支持的各种特性。使用这用配置方式时，系统中必须安装有ncurses库，否则会显示“Unable to find the ncurses libraries”的错误提示。其中“Y”、“N”、“M”、“？”键的选择功能与前面make config中介绍的一致。

q   **make xconfig**

基于X Window的一种配置方式，提供了漂亮的配置窗口，不过只有能够在X Server上使运行X桌面应用程序时才能够使用，它依赖于QT，如果系统中没有安装QT库，则会出现“Unable to find any QT installation”的错误提示。

q   **make gconfig**

与make xconfig类似，不同的是make gconfig依赖于GTK库。

q   **make defconfig**

按照内核代码中提供的默认配置文件对内核进行配置（在Intel x86\_64平台上，默认配置为arch/x86/configs/x86\_64\_defconfig），生成.config文件可以用作初始化配置，然后再使用make menuconfig进行定制化配置。

q   **make allyesconfig**

尽可能多地使用“y”设置内核选项值，生成的配置中包含了全部的内核特性。

q   **make allnoconfig**

除必须的选项外，其它选项一律不选 (常用于嵌入式Linux系统的编译)。

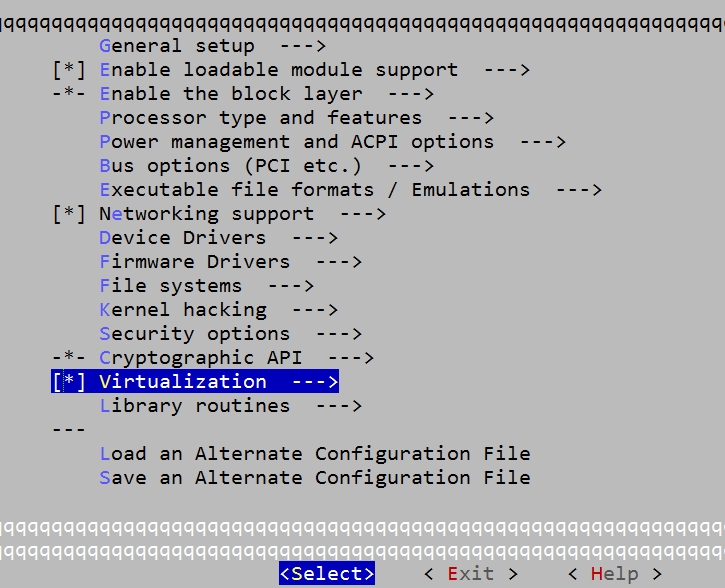
q   **make allmodconfig**

尽可能多的使用“m”设置内核选项值来生成配置文件。

q   **make localmodconfig**

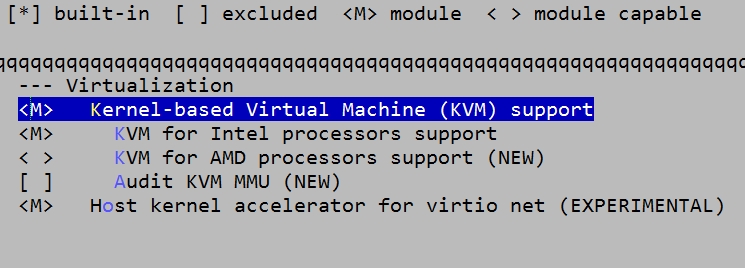
会执行 lsmod 命令查看当前系统中加载了哪些模块 (Modules)，并最后将原来的 .config 中不需要的模块去掉，仅保留前面 lsmod 命令查出来的这些模块，从而简化了内核的配置过程。 这样做确实方便了很多，但是也有个缺点：该方法仅能使编译出的内核支持当前内核已经加载的模块。因为该方法使用的是 lsmod 查询得到的结果，如果有的模块当前没有加载，那么就不会编到新的内核中。

下面以make menuconfig为例讲述一下如何选择KVM相关的配置；运行make menuconfig后显示的界面如图3-6所示：



make menuconfig 命令的选择界面

选择了Virtualization之后，进入里面进行详细的配置，包括选中KVM，选中对处理器的支持（比如：KVM for Intel processors support，KVM for AMD processors support），如图3-7所示：



Virtualization中的配置选项

在配置完成之后，就会在kvm.git的目录下面生成一个.config文件，最好检查一下KVM相关的配置是否正确，与KVM直接相关的配置项主要有如下几个，本次配置的情况如下：

CONFIG\_HAVE\_KVM=y

CONFIG\_HAVE\_KVM\_IRQCHIP=y

CONFIG\_HAVE\_KVM\_EVENTFD=y

CONFIG\_KVM\_APIC\_ARCHITECTURE=y

CONFIG\_KVM\_MMIO=y

CONFIG\_KVM\_ASYNC\_PF=y

CONFIG\_HAVE\_KVM\_MSI=y

CONFIG\_VIRTUALIZATION=y

CONFIG\_KVM=m

CONFIG\_KVM\_INTEL=m

# CONFIG\_KVM\_AMD is not set

CONFIG\_KVM\_MMU\_AUDIT=y

### Qemu-kvm

### libvirt

libvirt 提供一种虚拟机监控程序不可知的 API 来安全管理运行于主机上的来宾操作系统。libvirt 本身不是一种工具， 它是一种可以建立工具来管理来宾操作系统的 API。libvirt 本身构建于一种抽象的概念之上。它为受支持的虚拟机监控程序实现的常用功能提供通用的 API。libvirt 起初是专门为 Xen 设计的一种管理 API，后来被扩展为可支持多个虚拟机监控程序。

### kvm与qemu的关系

qemu：it is a emulator, 用来仿真cpu，设备，总线等设备

kvm： it's a kenerl module, 用来处理qemu向内核发起的请求

qemu是kvm的客户端展现，kvm是内核为qemu提供的服务代理，用来处理qemu的请求

### kvm 存储栈分析

## VirtualBox

## Hyper-V

# 数据分析

## WEB爬虫

## Heritrix

<http://crawler.archive.org/index.html>

## 搜索引擎

## teradata

## sybase

## greenplum

## hana

# 云存储

## 存储系统设计原理

## 基本存储设备

磁盘

固态盘

磁带

## 磁盘阵列

基本原理

RAID系统设计

磁盘阵列系统性能评价

磁盘阵列在线扩容技术

多协议磁盘阵列技术

磁盘阵列预警技术

## 对象存储系统研究

对象存储设备的指令集

对象存储文件系统

存储服务质量

## 可靠性和可用性研究

## 存储安全性研究

揭开存储系统的抽象“外壳”，

## SAN存储

## NAS存储

iSCSI initiator和target，对CDP、snapshot、云备份

## 分布式存储系统

## Sheepdog

sheepdog是作为虚拟机kvm的volume使用的，是非普通的文件系统，这点和Amazon的EBS(Elastic Block Store)比较类似

sheepdog是一种对称(symmetric)的结构，各个节点的地位相同，没有中心节点，没有meta-server，使用Corosync 对物理节点进行管理

sheepdog中的对象存储分为两类，其一是One reader One Writer 其二是No writer multiple reader 而且对象是4M大小分片的，使用“一致性哈希”算法来确定对象存储位置，多副本存储

## KFS

## CEPH

<http://ceph.com/>

# SDN

# 云平台

## cloudfoudry

## opennebula

## cloudstack

CloudStack是一个开源云计算解决方案。可以加速高伸缩性的公共和私有云（IaaS）的部署、管理、配置。使用CloudStack作为基础，数据中心操作者可以快速方便的通过现存基础架构创建云服务。

### Api 认证

<http://x.x.x.x:8080/client/api?response=json&command=login&username=admin&domain=&password=?Cloud.123>

### jclouds

认证cloudstack,包含两种认证方式，其配置文件是在org.jclouds.cloudstack.config包中。

CredentialType

CloudStackProperties

CloudStackParserModule

CloudStackHttpApiModule

## openstack

供OpenStack 是一个旨在为公共及私有云的建设与管理提软件的开源项目。它的社区拥有超过130家企业及1350位开发者，这些机构与个人都将OpenStack作为基础设施即服务（简称IaaS）资源的通用前端。OpenStack项目的首要任务是简化云的部署过程并为其带来良好的可扩展性。本文希望通过提供必要的指导信息，帮助大家利用OpenStack前端来设置及管理自己的公共云或私有云。

OpenStack是IaaS(基础设施即服务)组件，让任何人都可以自行建立和提供云端运算服务。

### Conceptual Diagram（概念图）

### Logical Diagram（逻辑图）

### Openstack Swift

## Docker

Docker的主要优点

* 轻量级资源使用：容器在进程级别隔离并使用宿主机的内核，而不需要虚拟化整个操作系统。
* 可移植性：一个容器应用所需要的依赖都在容器中，这就让它可以在任意一台Docker主机上运行。
* 可预测性：宿主机不需要关心容器内运行的是什么，同样，容器也不需要关心是在哪个宿主机上运行。所需要的接口都是标准化的，并且交互也都是可预测的。

### 常用组件介绍

### 容器化的综述

### 服务发现和分布式配置存储

### 网络与通信

### 集群

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 项目名称 | 中文描述 | 英文描述 | 开源证书 | 开发语言 | Like |
| fleet | 调度器和集群管理工具 | Base coreos ,  A Distributed init System | Apache 2.0 license | Go | / |
| marathon | 调度器和集群管理工具 | Deploy and manage containers (including Docker) on top of Apache Mesos at scale. | Apache 2.0 license | scale | 建议  使用 |
| Swarm | 调度器和集群管理工具 | Swarm: a Docker-native clustering system | Apache 2.0 license | Go | / |
| mesos | 宿主机抽象服务，用于为调度器联合宿主机资源 | Apache Mesos is a cluster manager that provides efficient resource isolation and sharing across distributed applications, or frameworks. | Apache 2.0 license | C++ |  |
| kubernetes | 一个管理容器组的工具，具有先进的调度能力 | Kubernetes is an open source system for managing containerized applications across multiple hosts, providing basic mechanisms for deployment, maintenance, and scaling of applications. | Apache 2.0 license | Go | 太复杂 |
| compose | 一个用于创建容器组的容器编排工具 | Define and run multi-container applications with Docker | Apache 2.0 license | python | / |

Marathon 优点

* HA -- run any number of Marathon schedulers, but only one gets elected as leader; if you access a non-leader, your request gets proxied to the current leader
* Constraints - e.g., only one instance of an application per rack, node, etc.
* Service Discovery & Load Balancing via HAProxy or the events API (see below).
* Health Checks: check your application's health via HTTP or TCP checks.
* Event Subscription lets you supply an HTTP endpoint to receive notifications, for example to integrate with an external load balancer.
* Web UI
* JSON/REST API for easy integration and scriptability
* Basic Auth and SSL
* Metrics: available at /metrics in JSON format

### Flynn

Flynn是一个使用Go语言编写的开源PaaS平台，Flynn使用模块化的设计，任何一个模块都可以独立的进行修改、升级和替换。Flynn的目标是简化分布式环境中应用的部署和维护，通过使用git push命令，Flynn就可以将应用部署到Docker，从而省去了复杂的配置和操作。Flynn的架构大致分为两层，Layer 0是底层的资源层，提供分布式配置、任务调度、服务发现、主机隔离等基础功能；Layer 1基于Layer 0构建了一个用于集群中管理、部署、扩展服务的系统，主要包括管理API/客户端、Git接收器、数据存储、路由。Flynn目前仍在开发中，尚未发布稳定版，但已经获得了很多公司的资助，它被称为是下一代的开源PaaS平台。

### Deis

Deis也是一个支持共有云和私有云的开源PaaS系统，Deis基于Docker和CentOS构建了一个类Heroku的PaaS系统。Deis主要设计用来和不同的云提供商进行交互，目前支持 Rackspace、EC2、 DigitalOcean、Google Compute Engine、Bare-Metal。Deis使用out-of-the-box的方式支持Ruby、Python、Node.js、Java、Clojure、Scala、Play、PHP、Perl、Dart和Go语言，同样支持git push部署。Flynn和Deis都是两个基于Docker的云计算微PaaS技术，关于它们的区别，可以参考这篇文章，作者从架构、实现方式等多方面对二者进行了比较，Deis目前也尚未发布1.0版本，但在GitHub上已经有2000+的star量。

### Dokku

Dokku是一个迷你版的Heroku，基于Docker使用100行左右的Bash代码编写，简单的安装和配置后，即可使用Git命令将应用部署到本地的Dokku平台（当使用git push命令的时候，Dokku会使用buildpack检测应用，然后再部署）。Dokku实际上相当于一个单机版的Heroku，它包含4个组件，分别是Docker、Buildstep、pluginhook、sshcommand。Dokku目前支持Node.js、Ruby、Python。

### CoreOS

CoreOS是一种新的、架构体系重新设计的Linux发行版，可以运行在既有的硬件或者云上。CoreOS不提供类似yum或者apt类似的包管理工具，用户不需要在CoreOS中安装软件，而是让程序都在Docker容器中去运行。CoreOS使用systemd和fleet来对容器进行管理，通过etcd进行服务发现和配置信息共享。CoreOS目前风头正劲，目前已经获得融资并于上周宣布收购私有Docker仓库服务商Quay.io，进军企业级的Registry。另外，CoreOS的etcd等组件也获得了社区的认可，并得到了大规模使用。CoreOS已经发布首个稳定版本，目前主流的云服务商都提供了对CoreOS的支持。

### Fig

Fig是一个基于Docker的用于快速搭建开发环境的工具，目前Fig团队已经加入Docker公司。Fig通过一个配置文件来管理多个Docker容器，非常适合组合使用多个容器进行开发的场景。Fig可以和Docker一起来构建基于Docker的复杂应用，CoreOS的功能强大但是配置比较复杂，而Fig相对而言比较简单，但是很难在多台服务器上做扩展，如何使用Fig构建多个容器的复杂应用并且把这些应用部署到基于CoreOS 的生产环境可以参考这篇文章。从Docker官方收购Fig也可以看到该项目的重要性，Fig目前尚未发布1.0版本。

### Atomic

Atomic是一个基于Fedora的用于运行Docker容器的原型系统，它从Fedora（在将来可能会是从另一个发行版）获取软件包，并扮演了一个“操作系统二进制文件的Git”的角色，允许对不同集合的软件包进行操作安装，并在它们之间进行原子的整体切换。因为Atomic项目与它的上游的Red Hat产品几乎承载着一样的二进制文件和内核，所以它同样会从上游的工程工作和支持中受益，这成为了它发展过程中的一项绝对优势。Atomic项目目前还没有达到用于生产环境的要求，仍在开发中。

### Drone

Drone是一个使用Go语言编写的基于Docker的持续集成系统。Drone可以快速提供隔离的虚拟环境编译测试，而且根据需要保留结果，比使用VM更加简洁有效。如何使用 Drone和Docker搭建全功能的CI服务器可以参考此文。使用Drone搭建CI服务器后，代码可以不离开公司网络即可测试，这非常适合大公司的保密原则，另外，由于Drone基于Docker使用，所以部署到生产环境也非常容易。

### Kubernetes

Kubernetes是来自Google的容器集群管理工具。Kubernetes支持跨平台，它可以在除Google以外的其它云平台中运行，比如AWS。基于Docker之上的Kubernetes可以构建一个容器的调度服务，其目的是让用户通过Kubernetes集群来管理云端容器的集群，而无需用户进行复杂的设置工作。系统会自动选取合适的工作节点来执行具体的容器集群调度处理工作。目前Kubernetes已经得到得到微软、IBM、红帽、CoreOS等公司的支持。

### Panamax

Panamax是一个CenturyLink开源的Docker管理工具，用户可以把多个Docker容器组合为模板并分享到GitHub。Panamax中的应用是由基于Docker镜像的独立服务组合而成，这些Docker镜像来自Docker Hub或者其它的Docker registry。Web的用户界面允许每个服务可以连接到其他服务，并可以配置环境变量、端口绑定、卷。另外也可以添加自定义的Docker运行命令。当这些服务组合在一起成为一个具备完整功能的应用后就可以作为一个模板保存到GitHub。Panamax的最初版本运行在由Vagrant管理的VirtualBox上，由于Vagrant的限制，目前Panamax仅可运行在Mac和Linux的VirtualBox上，并不支持其他虚拟化平台。CenturyLink的云平台也将会支持Panamax。

### Swarm

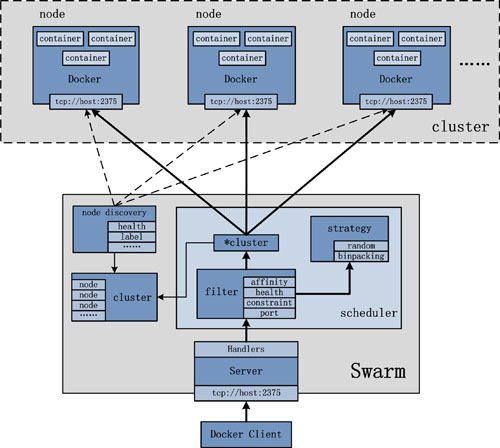
Docker自诞生以来，其容器特性以及镜像特性给DevOps爱好者带来了诸多方便。然而在很长的一段时间内，Docker只能在单host上运行，其跨host的部署、运行与管理能力颇受外界诟病。跨host能力的薄弱，直接导致Docker容器与host的紧耦合，这种情况下，Docker容器的灵活性很难令人满意，容器的迁移、分组等都成为很难实现的功能点。

Swarm是Docker公司在2014年12月初新发布的容器管理工具。和Swarm一起发布的Docker管理工具还有Machine以及Compose。

Swarm是一套较为简单的工具，用以管理Docker集群，使得Docker集群暴露给用户时相当于一个虚拟的整体。Swarm使用标准的Docker API接口作为其前端访问入口，换言之，各种形式的Docker Client(dockerclient in go, docker\_py, docker等)均可以直接与Swarm通信。Swarm几乎全部用Go语言来完成开发，并且还处于Alpha版本，目前在github上发布的版本仅有v0.1.0-rc1。然而Swarm的发展十分快速，功能和特性的变更迭代还非常频繁。因此，可以说Swarm还不推荐被用于生产环境中，但可以肯定的是Swarm是一项很有前途的技术。Swarm的设计和其他Docker项目一样，遵循“batteries included but removable”原则。笔者对该原则的理解是：batteries included代表设计Swarm时，为了完全体现分布式容器集群部署、运行与管理功能的完整性，Swarm和Docker协同工作，以达到效果；“but removable”意味着Swarm与Docker并非紧耦合，这套管理引擎并未侵入Docker的使用，同时这套机制也可以用以其他容器技术的集群部署、运行与管理。

#### Swarm架构

Swarm作为一个管理Docker集群的工具，首先需要将其部署起来，可以单独将Swarm部署于一个节点。另外，自然需要一个Docker集群，集群上每一个节点均安装有Docker。具体的Swarm架构图可以参照下图：



**图5.5-1**

Swarm架构中最主要的处理部分自然是Swarm节点，Swarm管理的对象自然是Docker Cluster，Docker Cluster由多个Docker Node组成，而负责给Swarm发送请求的是Docker Client。

#### Swarm命令

Swarm架构图可以让大家对Swarm有一个初步的认识，比如Swarm的具体工作流程：Docker Client发送请求给Swarm；Swarm处理请求并发送至相应的Docker Node；Docker Node执行相应的操作并返回响应。除此之外，Swarm的工作原理依然还不够明了。

深入理解Swarm的工作原理，可以先从Swarm提供的命令入手。Swarm支持的命令主要有4个：

* swarm create
* swarm manage
* swarm join
* swarm list

##### swarm create

Swarm中swarm create命令用于创建一个集群标志，用于Swarm管理Docker集群时，Docker Node的节点发现功能。发起该命令之后，Swarm会前往Docker Hub上内建的发现服务中获取一个全球唯一的token，用以唯一的标识Swarm管理的Docker集群。

**注：**Swarm的运行需要使用服务发现，目前该服务内建与Docker Hub，该服务发现机制目前还在alpha版本，站点为：http://discovery-stage.hub/docker.com 。

##### swarm manage

Swarm中swarm manage是最为重要的管理命令。一旦swarm manage命令在Swarm节点上被触发，则说明用户需要swarm开始管理Docker集群。从运行流程的角度来讲，swarm经历的阶段主要有两点：启动swarm、接收并处理Docker集群管理请求。

Swarm启动的过程包含三个步骤：

发现Docker集群中的各个节点，收集节点状态、角色信息，并监视节点状态的变化；

初始化内部调度（scheduler）模块；

创建并启动API监听服务模块；

**第一个步骤**，Swarm发现Docker集群中的节点。发现（discovery）是Swarm中用于维护Docker集群状态的机制。既然涉及到发现（discovery），那在这之前必须先有注册（register）。Swarm中有专门负责发现（discovery）的模块，而关于注册（register）部分，不同的discovery模式下，注册（register）也会有不同的形式。

目前，Swarm中提供了5种不同的发现（discovery）机制：Node Discovery、File Discovery、Consul Discovery、EtcD Discovery和Zookeeper Discovery。

**第二个步骤，**Swarm内部的调度（scheduler）模块被初始化。swarm通过发现机制发现所有注册的Docker Node，并收集到所有Docker Node的状态以及具体信息。此后，一旦Swarm接收到具体的Docker管理请求，Swarm需要对请求进行处理，并通过所有Docker Node的状态以及具体信息，来筛选（filter）决策到底哪些Docker Node满足要求，并通过一定的策略（strategy）将请求转发至具体的一个Docker Node。

**第三个步骤，**Swarm创建并初始化API监听服务模块。从功能的角度来讲，可以将该模块抽象为Swarm Server。需要说明的是：虽然Swarm Server完全兼容Docker的API，但是有不少Docker的命令目前是不支持的，毕竟管理Docker集群与管理单独的Docker会有一些区别。当Swarm Server被初始化并完成监听之后，用户即可以通过Docker Client向Swarm发送Docker集群的管理请求。

Swarm的swarm manage接收并处理Docker集群的管理请求，即是Swarm内部多个模块协同合作的结果。请求入口为Swarm Server，处理引擎为Scheduler，节点信息依靠Disocovery。

##### swarm join

Swarm的swarm join命令用于将Docker Node添加至Swarm管理的Docker集群中。从这点也可以看出swarm join命令的执行位于Docker Node，因此在Docker Node上运行该命令，首先需要在Docker Node上安装Swarm，由于该Swarm只会执行swarm join命令，故可以将其当成Docker Node上用于注册的agent模块。

功能而言，swarm join可以认为是完成Docker Node在Swarm节点处的注册（register）工作，以便Swarm在执行swarm manage时可以发现该Docker Node。然而，上文提及的5种discovery模式中，并非每种模式都支持swarm join命令。不支持的discovery的模式有Node Discovery与File Discovery。

Docker Node上swarm join执行之后，标志着Docker Node向Swarm注册，请求加入Swarm管理的Docker集群中。Swarm通过注册信息，发现Docker Node，并获取Docker Node的状态以及具体信息，以便处理Docker请求时作为调度依据。

##### swarm list

Swarm中的swarm list命令用以列举Docker集群中的Docker Node。

Docker Node的信息均来源于Swarm节点上注册的Docker Node。而一个Docker Node在Swarm节点上注册，仅仅是注册了Docker Node的IP地址以及Docker监听的端口号。

使用swarm list命令时，需要指定discovery的类型，类型包括：token、etcd、file、zk以及<ip>。而swarm list并未罗列Docker集群的动态信息，比如Docker Node真实的运行状态，或者Docker Node在Docker集群中扮演的角色信息。

Swarm的架构以及命令并没有很复杂，同时也为希望管理Docker集群的Docker爱好者降低了学习和使用门槛。

俗话说得好，没有一种一劳永逸的工具，有效的管理Docker集群同样也是如此。**缺乏场景来谈论Swarm的价值，意义并不会很大。相反，探索和挖掘Swarm的特点与功能，并为Docker集群的管理提供一种可选的方案，是Docker爱好者更应该参与的事。**

[Swarm: a Docker-native clustering system](http://github.com/docker/swarm" \t "_blank)

[Intro to Docker Swarm: Part 1 - Overview](http://technolo-g.com/intro-to-docker-swarm-pt1-overview/" \t "_blank)

[Intro to Docker Swarm: Part 2 - Configuration Options and Requirements](http://technolo-g.com/intro-to-docker-swarm-pt2-config-options-requirements/)

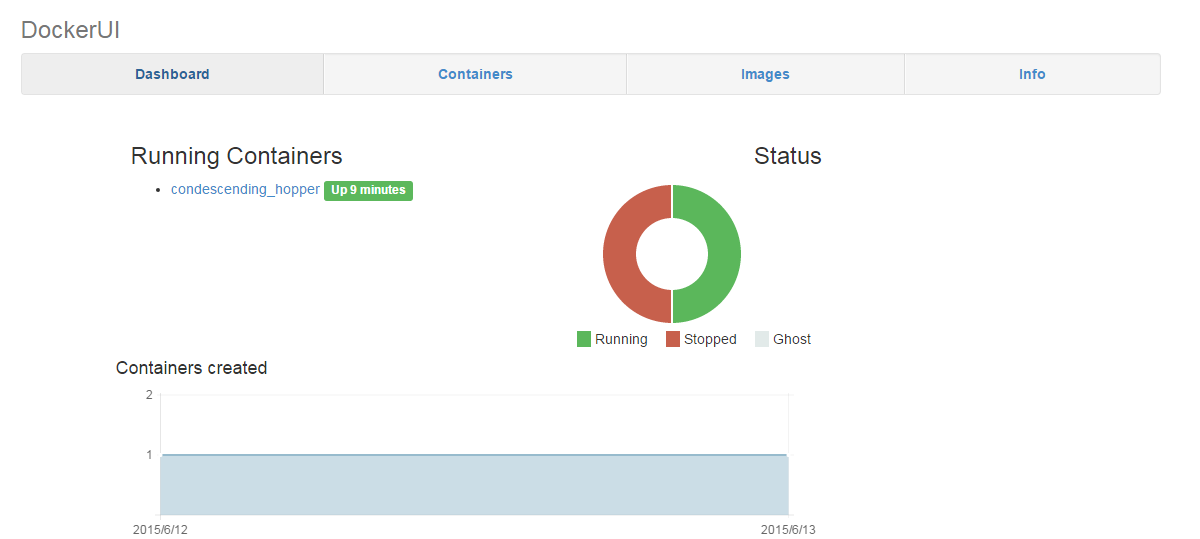
### Boot2Docker

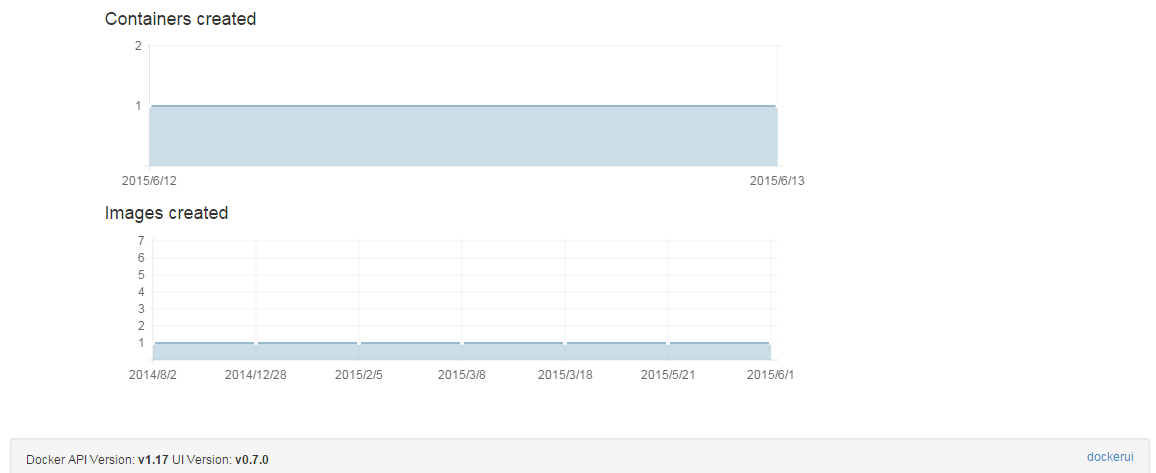
Boot2Docker是一个专为Docker而设计的轻量级Linux发型包，解决Windows或者OS X用户不能安装Docker的问题。Boot2Docker完全运行于内存中，24M大小，启动仅5-6秒。Boot2Docker需要运行在VirtualBox中，具体的安装可以参考官方文档。

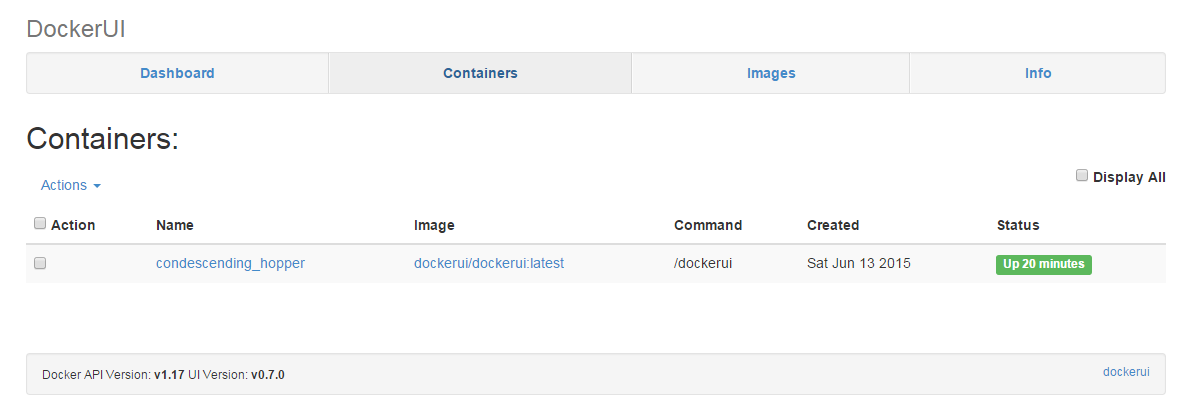
### DockerUI

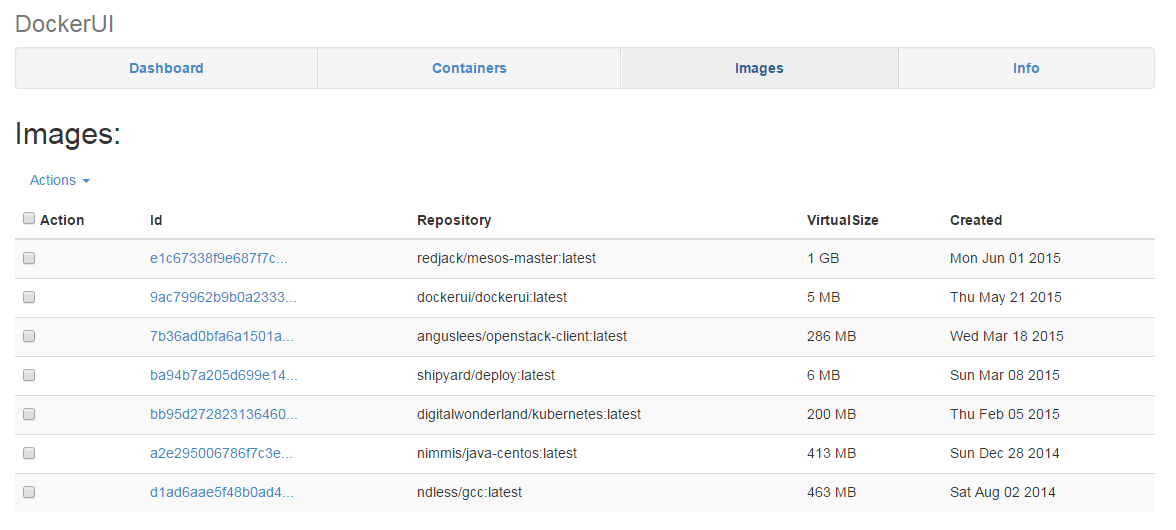
docker pull dockerui/dockerui

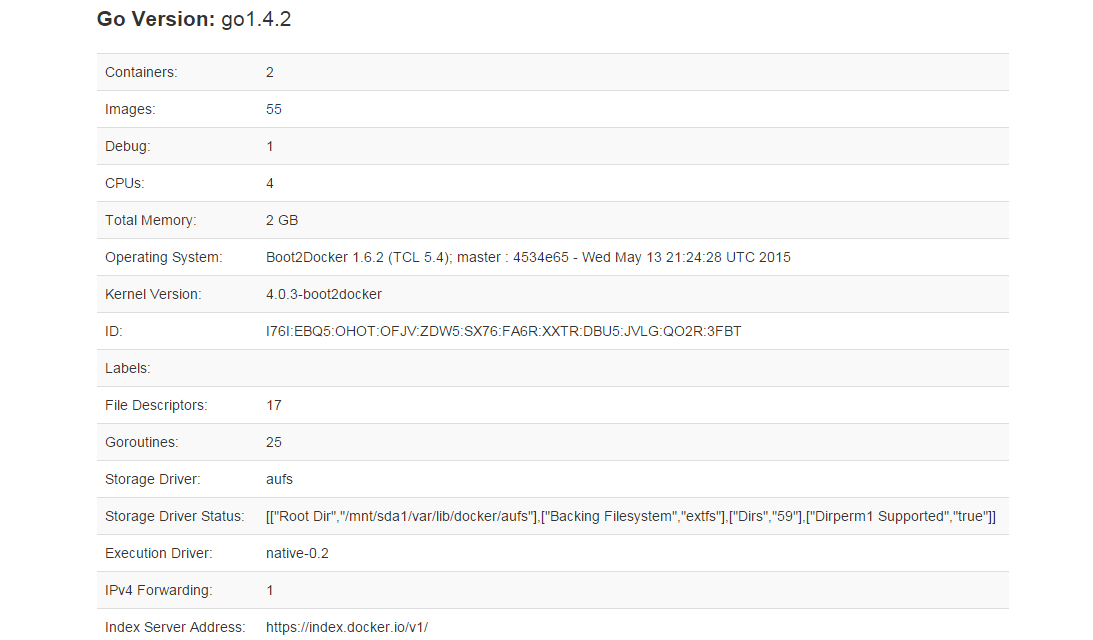
docker run -d -p 9000:9000 --privileged -v /var/run/docker.sock:/var/run/docker.sock dockerui/dockerui











### Cgroups

cgroups 实现了对资源的配额和度量。 cgroups 的使用非常简单，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九大子系统的资源。

1. 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，光盘以及usb等。
2. cpu 这个子系统使用调度程序为cgroup任务提供cpu的访问。
3. cpuacct 产生cgroup任务的cpu资源报告。
4. 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。
5. devices 允许或拒绝cgroup任务对设备的访问。
6. freezer 暂停和恢复cgroup任务。
7. memory 设置每个cgroup的内存限制以及产生内存资源报告。
8. net\_cls 标记每个网络包以供cgroup方便使用。
9. ns 名称空间子系统。

### LXC

### AUFS

AUFS (AnotherUnionFS) 是一种 Union FS, 简单来说就是支持将不同目录挂载到同一个虚拟文件系统下(unite several directories into a single virtual filesystem)的文件系统, 更进一步的理解, AUFS支持为每一个成员目录(类似Git Branch)设定readonly、readwrite 和 whiteout-able 权限, 同时 AUFS 里有一个类似分层的概念, 对 readonly 权限的 branch 可以逻辑上进行修改(增量地, 不影响 readonly 部分的)。通常 Union FS 有两个用途, 一方面可以实现不借助 LVM、RAID 将多个disk挂到同一个目录下, 另一个更常用的就是将一个 readonly 的 branch 和一个 writeable 的 branch 联合在一起，Live CD正是基于此方法可以允许在 OS image 不变的基础上允许用户在其上进行一些写操作。Docker 在 AUFS 上构建的 container image 也正是如此，接下来我们从启动 container 中的 linux 为例来介绍 docker 对AUFS特性的运用。

典型的启动Linux运行需要两个FS: bootfs + rootfs:典型的启动Linux运行需要两个FS: bootfs + rootfs:



图5-1

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了. rootfs (root file system) 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, 因此不同的发行版可公用bootfs 如下图:

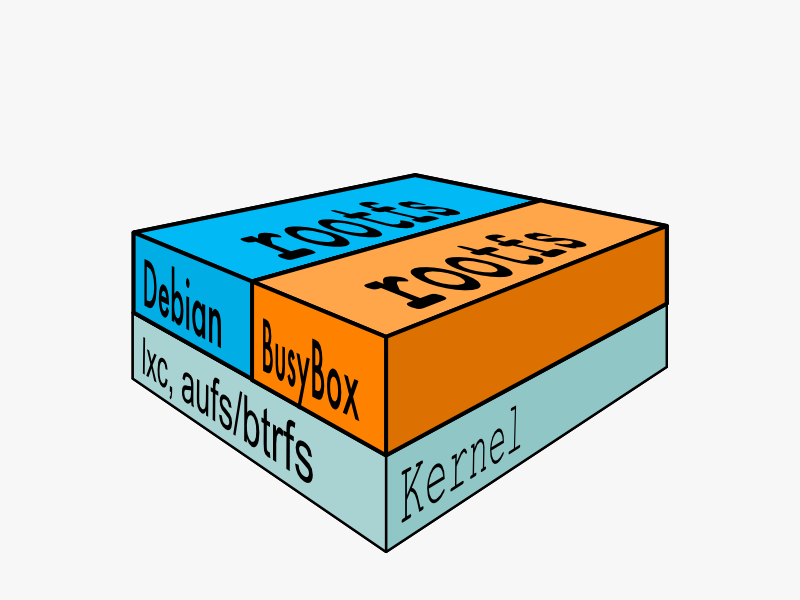


图5-2

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(file system) 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。如下图:

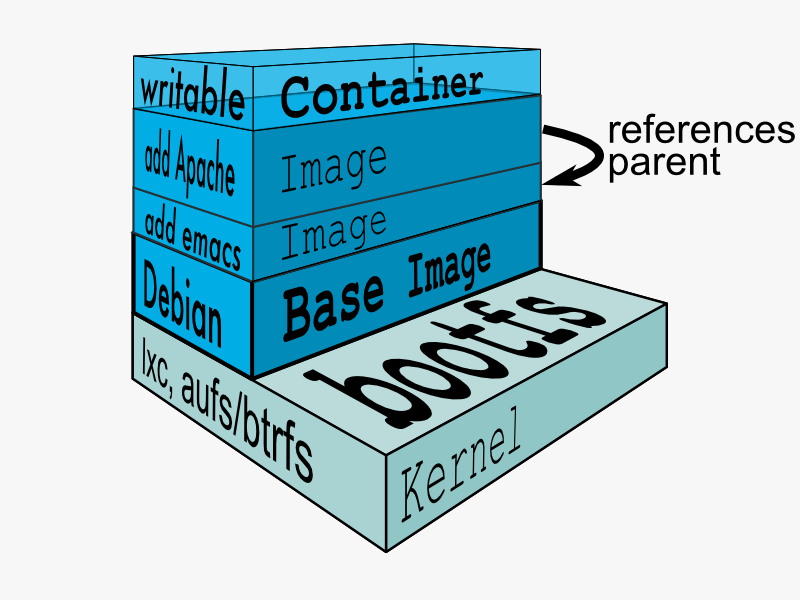


图5-3

得益于AUFS的特性, 每一个对readonly层文件/目录的修改都只会存在于上层的writeable层中。这样由于不存在竞争, 多个container可以共享readonly的FS层。 所以Docker将readonly的FS层称作 "image" - 对于container而言整个rootfs都是read-write的，但事实上所有的修改都写入最上层的writeable层中, image不保存用户状态，只用于模板、新建和复制使用。

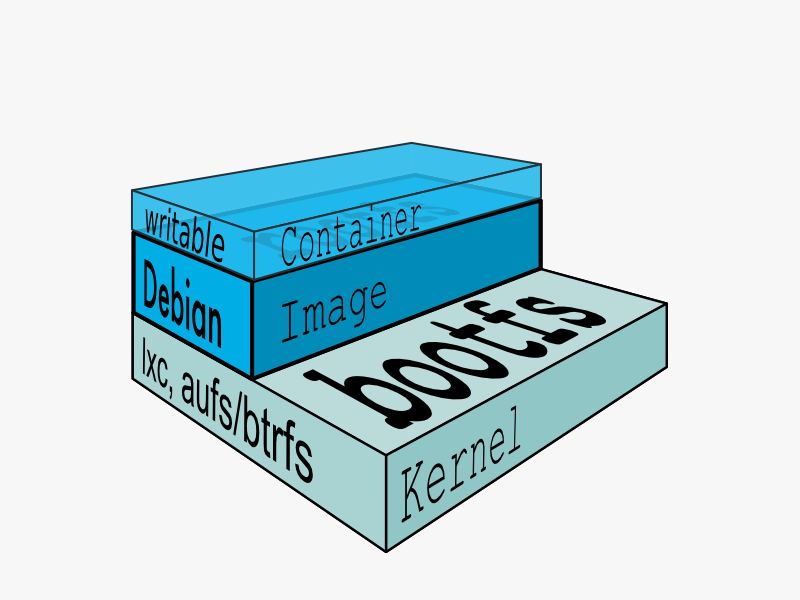


图5-5

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。因此想要从一个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。如下图:

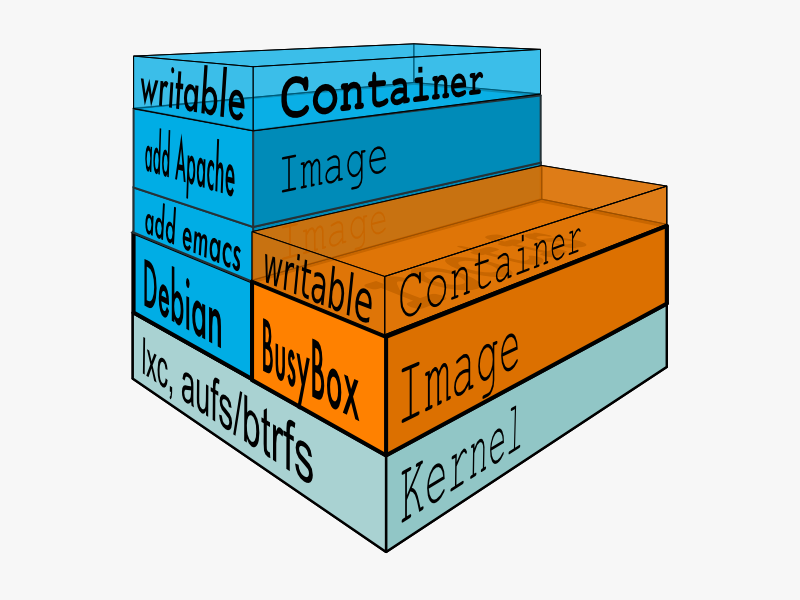


图5-6

安全永远是相对的，这里有三个方面可以考虑Docker的安全特性:

1. 由kernel namespaces和cgroups实现的Linux系统固有的安全标准;
2. Docker Deamon的安全接口;
3. Linux本身的安全加固解决方案,类如AppArmor, SELinux;

由于安全属于非常具体的技术，这里不在赘述，请直接参阅[Docker官方文档](http://blog.dataman-inc.com/untitled-2/docs.docker.com/articles/security/)。

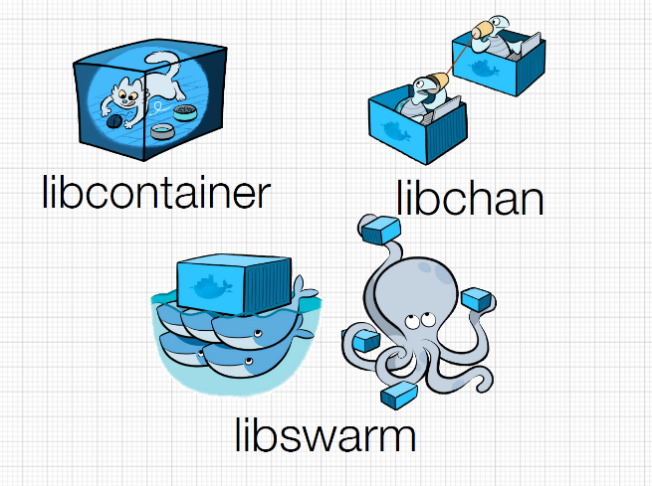


图5-7

#### Libswarm

Libswarm，是Solomon Hykes (Docker的CTO) 在DockerCon 2014峰会上向社区介绍的新“乐高积木”工具: 它是用来统一分布式系统的网络接口的API。Libswarm要解决的问题是，基于Docker构建的分布式应用已经催生了多个基于Docker的服务发现(Serivce Discovery)项目，例如etcd, fleet, geard, mesos, shipyard, serf等等，每一套解决方案都有自己的通讯协议和使用方法，使用其中的任意一款都会局限在某一个特定的技术范围內。所以Docker的CTO就想用libswarm暴露出通用的API接口给分布式系统使用，打破既定的协议限制。目前项目还在早期发展阶段，值得参与。

#### Libchan

Libchan，是一个底层的网络库，为上层 Libswarm 提供支持。相当于给Docker加上了ZeroMQ或RabbitMQ，这里自己实现网络库的好处是对Docker做了特别优化，更加轻量级。一般开发者不会直接用到它，大家更多的还是使用Libswarm来和容器交互。喜欢底层实现的网络工程师可能对此感兴趣，不妨一看。

#### Libcontainer

Libcontainer，Docker技术的核心部分，单独列出来也是因为这一块的功能相对独立，功能代码的迭代升级非常快。想了解Docker最新的支持特性应该多关注这个模块。

## Mesos

### Marathon

### Bamboo

### Chronos

## AWS

<http://aws.amazon.com/cn/>

### aws sdk

http://webservices.amazon.com/onca/xml?Service=AWSECommerceService&AWSAccessKeyId=**AKIAOU2W76C6ADD4YTZQ**&AssociateTag=mytag-20&Operation=ItemLookup&ItemId=0679722769&ResponseGroup=Images,ItemAttributes,Offers,Reviews&Version=2013-08-01&Timestamp=2014-08-18T12:00:00Z

https://elasticmapreduce.amazonaws.com?

Action=DescribeJobFlows

&Version=2009-03-31

&AWSAccessKeyId= **AKIAOU2W76C6ADD4YTZQ**&SignatureVersion=2

&SignatureMethod=HmacSHA256

&Timestamp=2011-10-03T15:19:30

Web 服务请求通过 Internet 发送，易被篡改。为了确保请求未被更改，AWS 会计算签名，以确定任何参数或参数值在传输途中是否发生了更改。AWS 要求把签名作为每个请求的一部分。

**Note**

请确保对 请求进行 URI 编码。例如，您 请求中的空白应编码为 %20。虽然 HTTP 协议规范通常允许未编码的空格，但使用未编码的字符会使查询请求中的签名无效。请勿将空格编码为加号 (+)，因为这会导致错误。

您可以使用 AWS Security Token Service (AWS STS) 提供的临时安全证书对请求进行签名。此过程与使用长期证书相同。有关更多信息，请参阅[将临时安全证书用于 API](http://docs.aws.amazon.com/STS/latest/UsingSTS/using-temp-creds.html" \l "RequestWithSTS" \t "_blank)。

签名示范

http://docs.aws.amazon.com/zh\_cn/general/latest/gr/signature-v4-examples.html#signature-v4-examples-java

**AWS services 支持Signature Version 2版本的服务有**

|  |  |
| --- | --- |
| Auto Scaling | [Auto Scaling API Reference](http://docs.aws.amazon.com/AutoScaling/latest/APIReference/" \t "_blank) |
| AWS CloudFormation | [AWS CloudFormation API Reference](http://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/" \t "_blank) |
| Amazon CloudWatch | [Amazon CloudWatch API Reference](http://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/" \t "_blank) |
| AWS Elastic Beanstalk | [Elastic Beanstalk API Reference](http://docs.aws.amazon.com/elasticbeanstalk/latest/api/" \t "_blank) |
| Amazon Elastic Compute Cloud | [Amazon EC2 API Reference](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/" \t "_blank) |
| Elastic Load Balancing | [Elastic Load Balancing API Reference](http://docs.aws.amazon.com/ElasticLoadBalancing/latest/APIReference/" \t "_blank) |
| Amazon Elastic MapReduce | [Amazon Elastic MapReduce API Reference](http://docs.aws.amazon.com/ElasticMapReduce/latest/API/" \t "_blank) |
| Amazon ElastiCache | [Amazon ElastiCache API Reference](http://docs.aws.amazon.com/AmazonElastiCache/latest/APIReference/" \t "_blank) |
| AWS Identity and Access Management | [IAM API Reference](http://docs.aws.amazon.com/IAM/latest/APIReference/" \t "_blank) |
| AWS Import/Export | [AWS Import/Export API Reference](http://docs.aws.amazon.com/AWSImportExport/latest/DG/" \t "_blank) |
| Amazon Relational Database Service | [Amazon Relational Database Service API Reference](http://docs.aws.amazon.com/AmazonRDS/latest/APIReference/" \t "_blank) |
| Amazon Simple Notification Service | [Amazon Simple Notification Service API Reference](http://docs.aws.amazon.com/sns/latest/api/" \t "_blank) |
| Amazon Simple Queue Service | [Amazon Simple Queue Service API Reference](http://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/" \t "_blank) |
| Amazon SimpleDB | [Amazon SimpleDB API Reference](http://docs.aws.amazon.com/AmazonSimpleDB/latest/DeveloperGuide/SDB_API.html" \t "_blank) |

### 实例支持REST的action

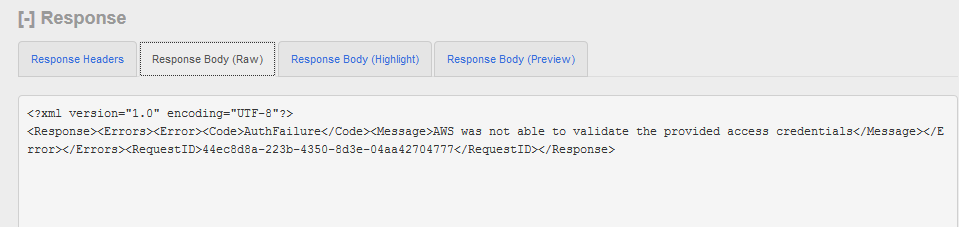
**Instances**

* [DescribeInstanceAttribute](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_DescribeInstanceAttribute.html" \o "DescribeInstanceAttribute)
* [DescribeInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_DescribeInstances.html" \o "DescribeInstances)
* [DescribeInstanceStatus](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_DescribeInstanceStatus.html" \o "DescribeInstanceStatus)
* [GetConsoleOutput](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_GetConsoleOutput.html" \o "GetConsoleOutput)
* [GetPasswordData](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_GetPasswordData.html" \o "GetPasswordData)
* [ModifyInstanceAttribute](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_ModifyInstanceAttribute.html" \o "ModifyInstanceAttribute)
* [MonitorInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_MonitorInstances.html" \o "MonitorInstances)
* [RebootInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_RebootInstances.html" \o "RebootInstances)
* [ReportInstanceStatus](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_ReportInstanceStatus.html" \o "ReportInstanceStatus)
* [ResetInstanceAttribute](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_ResetInstanceAttribute.html" \o "ResetInstanceAttribute)
* [RunInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_RunInstances.html" \o "RunInstances)
* [StartInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_StartInstances.html" \o "StartInstances)
* [StopInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_StopInstances.html" \o "StopInstances)
* [TerminateInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_TerminateInstances.html" \o "TerminateInstances)
* [UnmonitorInstances](http://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_UnmonitorInstances.html" \o "UnmonitorInstances)

列出实例的案例

<http://ec2.cn-north-1.amazonaws.com.cn/?AWSAccessKeyId=AKIAOTSJPWFN3V5LXRJQ&Action=DescribeInstances&SignatureMethod=HmacSHA256&SignatureVersion=2&Timestamp=2015-11-09T09%3A59%3A12Z&Version=2015-04-15&Signature=%2FNRtYAo5lvvC5p%2F0XkVrrkcRjta0HOR0Lnzo5urgsJQ%3D>

无法认证错误信息：



### jclouds

## Aliyun

<http://www.aliyun.com/>

## 云平台架构

### 私有云

### 混合云

### 专有云

### 桌面云

#### Spice

### overview

云推送系统，云消息系统，底层协议

高性能服务器软件开发

OSGI、AXIS2

penshift/cloudfoundry/Appscale/stratos

Portal、BI、EAI

计算机网络协议及大并发量、网络服务器端开发

虚拟化、操作系统内核

数据中心资源管理、监控、调度等系统研

1、负责云系统的架构安全设计

2、云系统的均衡负载设计

3、云的优化存储设计

Linux/Unix操作系统，PBS、LSF等排队系统

Fortran、C/C++、Java

Python、Perl、TK/TCL

VASP、Gaussian、CP2K、CPMD、ADF、Amber、CHARMM、Gromacs、Molpro、Vitual Docker、Schrodinger、Sybyl、Materials Studio、Discover Studio

快速部署、资源调度、多租户技术、海量数据处理技术、大规模消息通信技术、大规模分布式存储技术、许可证管理及收费；

算法、数据架构和存储技术（SAN、scale-out NAS、iscsi、FC）

网络交换、路由协议开发或访问控制、AAA开发’

多租户技术

# 资源监控

## zabbix

<https://www.zabbix.org/wiki/Main_Page>

## nagios

<http://sourceforge.net/projects/nagios/?source=dlp>

<http://www.ibm.com/developerworks/cn/linux/l-ganglia-nagios-1/index.html>

[Linux下Nagios的安装与配置](http://www.cnblogs.com/mchina/archive/2013/02/20/2883404.html)

一、Nagios简介

[Nagios logo](http://images.cnitblog.com/blog/370046/201301/30172010-1a4517cdb03747cc8d37837d6f550d8a.png)

　　Nagios是一款开源的电脑系统和网络监视工具，能有效监控Windows、Linux和Unix的主机状态，交换机路由器等网络设置，打印机等。在系统或服务状态异常时发出邮件或短信报警第一时间通知网站运维人员，在状态恢复后发出正常的邮件或短信通知。

　　Nagios原名为NetSaint，由Ethan Galstad开发并维护至今。NAGIOS是一个缩写形式: "Nagios Ain't Gonna Insist On Sainthood" Sainthood 翻译为圣徒，而"Agios"是"saint"的希腊表示方法。Nagios被开发在Linux下使用，但在Unix下也工作得非常好。

主要功能

网络服务监控（SMTP、POP3、HTTP、NNTP、ICMP、SNMP、FTP、SSH）

主机资源监控（CPU load、disk usage、system logs），也包括Windows主机（使用NSClient++ plugin）

可以指定自己编写的Plugin通过网络收集数据来监控任何情况（温度、警告……）

可以通过配置Nagios远程执行插件远程执行脚本

远程监控支持SSH或SSL加通道方式进行监控

简单的plugin设计允许用户很容易的开发自己需要的检查服务，支持很多开发语言（shell scripts、C++、Perl、ruby、Python、PHP、C#等）

包含很多图形化数据Plugins（Nagiosgraph、Nagiosgrapher、PNP4Nagios等）

可并行服务检查

能够定义网络主机的层次，允许逐级检查，就是从父主机开始向下检查

当服务或主机出现问题时发出通告，可通过email, pager, sms 或任意用户自定义的plugin进行通知

能够自定义事件处理机制重新激活出问题的服务或主机

自动日志循环

支持冗余监控

包括Web界面可以查看当前网络状态，通知，问题历史，日志文件等

二、Nagios工作原理

　　Nagios的功能是监控服务和主机，但是他自身并不包括这部分功能，所有的监控、检测功能都是通过各种插件来完成的。

　　启动Nagios后，它会周期性的自动调用插件去检测服务器状态，同时Nagios会维持一个队列，所有插件返回来的状态信息都进入队列，Nagios每次都从队首开始读取信息，并进行处理后，把状态结果通过web显示出来。

　　Nagios提供了许多插件，利用这些插件可以方便的监控很多服务状态。安装完成后，在nagios主目录下的/libexec里放有nagios自带的可以使用的所有插件，如，check\_disk是检查磁盘空间的插件，check\_load是检查CPU负载的，等等。每一个插件可以通过运行./check\_xxx –h 来查看其使用方法和功能。

　　Nagios可以识别4种状态返回信息，即 0(OK)表示状态正常/绿色、1(WARNING)表示出现警告/黄色、2(CRITICAL)表示出现非常严重的错误/红色、3(UNKNOWN)表示未知错误/深黄色。Nagios根据插件返回来的值，来判断监控对象的状态，并通过web显示出来，以供管理员及时发现故障。

四种监控状态

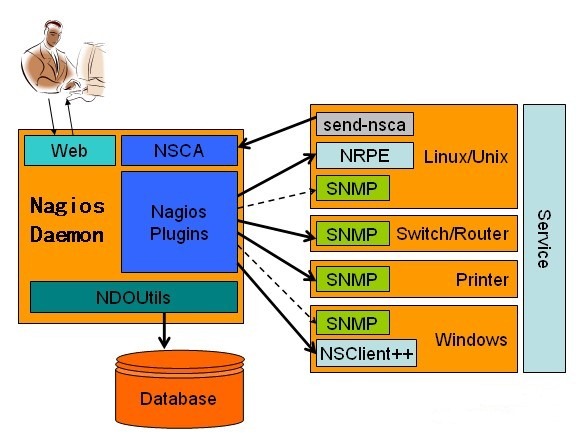
[](http://images.cnitblog.com/blog/370046/201301/30172012-5e48d4a0cbc542b88faf0cdaab257bef.jpg)

　　再说报警功能，如果监控系统发现问题不能报警那就没有意义了，所以报警也是nagios很重要的功能之一。但是，同样的，Nagios 自身也没有报警部分的代码，甚至没有插件，而是交给用户或者其他相关开源项目组去完成的。

　　Nagios 安装，是指基本平台，也就是Nagios软件包的安装。它是监控体系的框架，也是所有监控的基础。

　　打开Nagios官方的文档，会发现Nagios基本上没有什么依赖包，只要求系统是Linux或者其他Nagios支持的系统。不过如果你没有安装apache（http服务），那么你就没有那么直观的界面来查看监控信息了，所以apache姑且算是一个前提条件。关于apache的安装，网上有很多，照着安装就是了。安装之后要检查一下是否可以正常工作。

　　知道Nagios 是如何通过插件来管理服务器对象后，现在开始研究它是如何管理远端服务器对象的。Nagios 系统提供了一个插件NRPE。Nagios 通过周期性的运行它来获得远端服务器的各种状态信息。它们之间的关系如下图所示：

[](http://images.cnitblog.com/blog/370046/201301/30172017-a0cab434c3184ee48163b3c2f86d0299.jpg)

Nagios 通过NRPE 来远端管理服务

1. Nagios 执行安装在它里面的check\_nrpe 插件，并告诉check\_nrpe 去检测哪些服务。

2. 通过SSL，check\_nrpe 连接远端机子上的NRPE daemon

3. NRPE 运行本地的各种插件去检测本地的服务和状态(check\_disk,..etc)

4. 最后，NRPE 把检测的结果传给主机端的check\_nrpe，check\_nrpe 再把结果送到Nagios状态队列中。

5. Nagios 依次读取队列中的信息，再把结果显示出来。

三、实验环境

|  |  |  |  |
| --- | --- | --- | --- |
| Host Name | OS | IP | Software |
| Nagios-Server | CentOS release 6.3 (Final) | 192.168.1.108 | Apache、Php、Nagios、nagios-plugins |
| Nagios-Linux | CentOS release 5.8 (Final) | 192.168.1.111 | nagios-plugins、nrpe |
| Nagios-Windows | Windows XP | 192.168.1.113 | NSClient++ |

Server 安装了nagios软件，对监控的数据做处理，并且提供web界面查看和管理。当然也可以对本机自身的信息进行监控。

Client 安装了NRPE等客户端，根据监控机的请求执行监控，然后将结果回传给监控机。

防火墙已关闭/iptables: Firewall is not running.

SELINUX=disabled

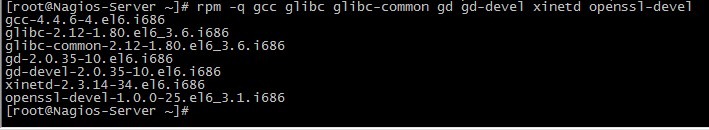
四、实验目标



五、Nagios服务端安装

5.1 基础支持套件：gcc glibc glibc-common gd gd-devel xinetd openssl-devel

# rpm -q gcc glibc glibc-common gd gd-devel xinetd openssl-devel

[](http://images.cnitblog.com/blog/370046/201301/30172023-cbd356b06f1c4fb796b99237b3554b2f.jpg)

如果系统中没有这些套件，使用yum 安装

# yum install -y gcc glibc glibc-common gd gd-devel xinetd openssl-devel

5.2 创建nagios用户和用户组

# useradd -s /sbin/nologin nagios

# mkdir /usr/local/nagios

# chown -R nagios.nagios /usr/local/nagios

[](http://images.cnitblog.com/blog/370046/201301/30172026-dd983461e7e24d3fa12c39d213616afc.jpg)

查看nagios 目录的权限

# ll -d /usr/local/nagios/

[7](http://images.cnitblog.com/blog/370046/201301/30172027-2efbcf5f62f34eae857b7c48c0f94d02.jpg)

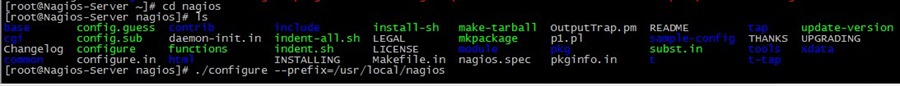
5.3 编译安装Nagios

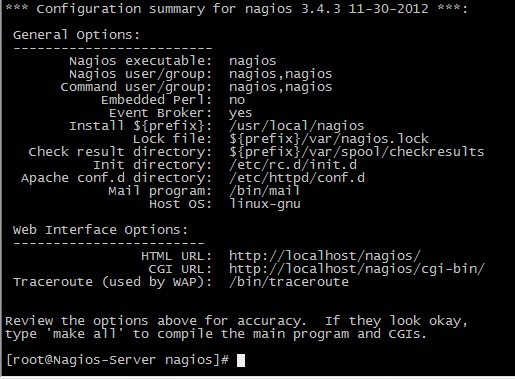
# wget <http://prdownloads.sourceforge.net/sourceforge/nagios/nagios-3.4.3.tar.gz>

# tar zxvf nagios-3.4.3.tar.gz

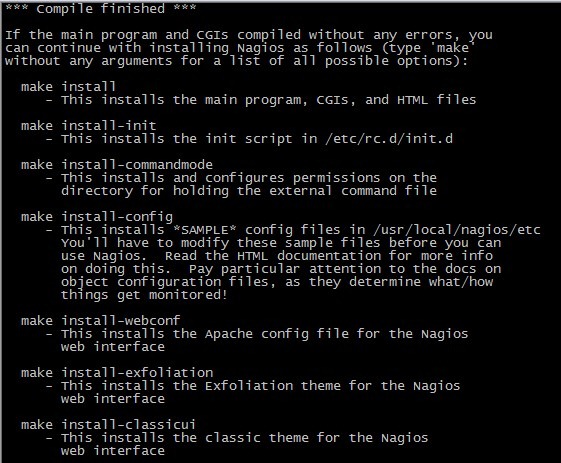
# cd nagios

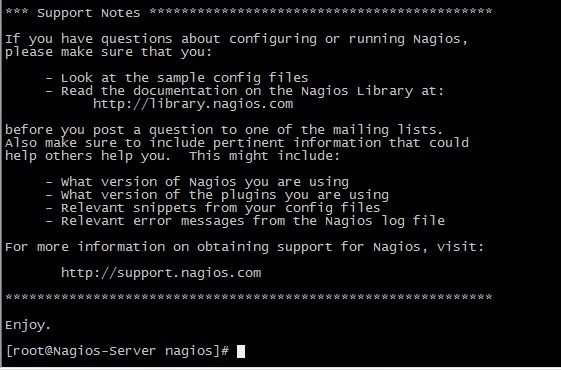
# ./configure --prefix=/usr/local/nagios

[](http://images.cnitblog.com/blog/370046/201301/30172029-51aac459f3a34106af7db6aaaa403d9e.jpg)

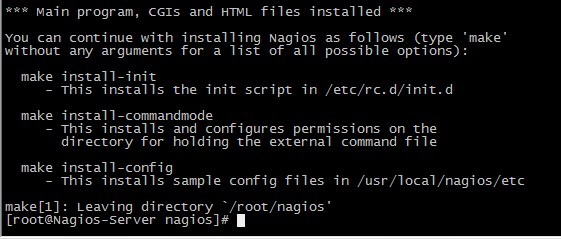
[](http://images.cnitblog.com/blog/370046/201301/30172030-18fdb7961c474b75a2b033e2a4171519.jpg)

# make all

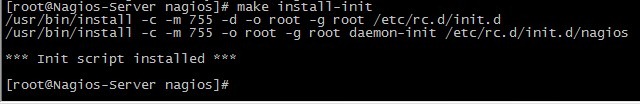
[](http://images.cnitblog.com/blog/370046/201301/30172031-5c6a7463c11d4544bee8f2eed12d7647.jpg)

[](http://images.cnitblog.com/blog/370046/201301/30172034-ebe3fcafffe94564b8161e6b78fe64c8.jpg)

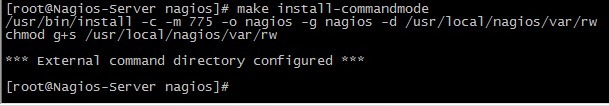
# make install

[](http://images.cnitblog.com/blog/370046/201301/30172035-b01026f0079c477c815351fb70635f40.jpg)

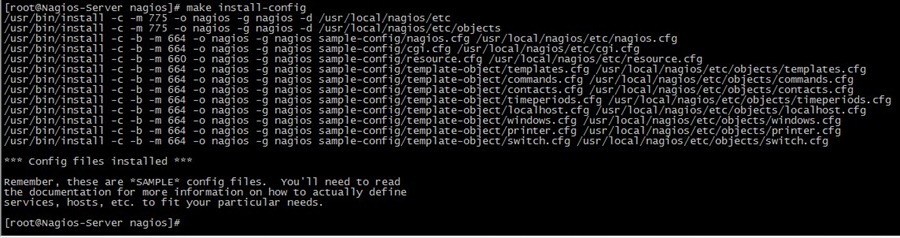
# make install-init

[](http://images.cnitblog.com/blog/370046/201301/30172038-629553af940645d3a1e1b34db39106e7.jpg)

# make install-commandmode



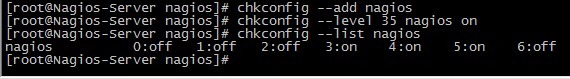
# make install-config



# chkconfig --add nagios

# chkconfig --level 35 nagios on

# chkconfig --list nagios



5.4 验证程序是否被正确安装

切换目录到安装路径（这里是/usr/local/nagios），看是否存在etc、bin、sbin、share、var 这五个目录，如果存在则可以表明程序被正确的安装到系统了。Nagios 各个目录用途说明如下：

|  |  |
| --- | --- |
| bin | Nagios 可执行程序所在目录 |
| etc | Nagios 配置文件所在目录 |
| sbin | Nagios CGI 文件所在目录，也就是执行外部命令所需文件所在的目录 |
| share | Nagios网页文件所在的目录 |
| libexec | Nagios 外部插件所在目录 |
| var | Nagios 日志文件、lock 等文件所在的目录 |
| var/archives | Nagios 日志自动归档目录 |
| var/rw | 用来存放外部命令文件的目录 |

5.5 安装Nagios 插件

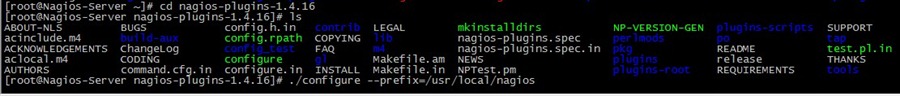
# wget <http://prdownloads.sourceforge.net/sourceforge/nagiosplug/nagios-plugins-1.4.16.tar.gz>

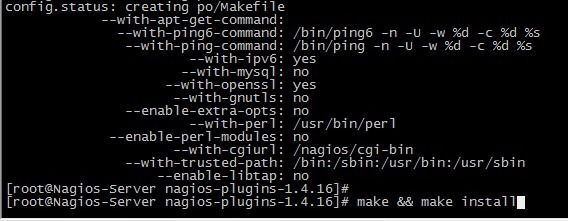
# tar zxvf nagios-plugins-1.4.16.tar.gz

# cd nagios-plugins-1.4.16

# ./configure --prefix=/usr/local/nagios

# make && make install





5.6 安装与配置Apache和Php

Apache 和Php 不是安装nagios 所必须的，但是nagios提供了web监控界面，通过web监控界面可以清晰的看到被监控主机、资源的运行状态，因此，安装一个web服务是很必要的。   
需要注意的是，nagios在nagios3.1.x版本以后，配置web监控界面时需要php的支持。这里我们下载的nagios版本为nagios-3.4.3，因此在编译安装完成apache后，还需要编译php模块，这里选取的php版本为php5.4.10。

a. 安装Apache

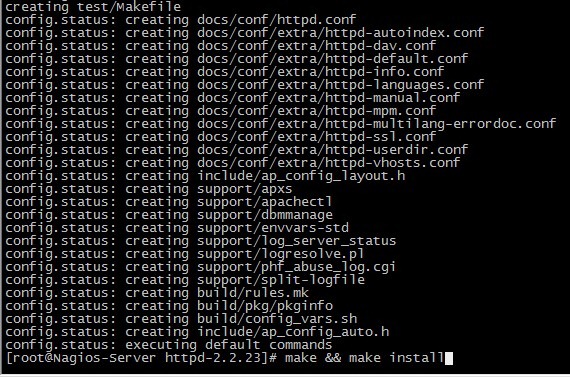
# wget [http://labs.mop.com/apache-mirror//httpd/httpd-2.2.23.tar.gz](http://labs.mop.com/apache-mirror/httpd/httpd-2.2.23.tar.gz" \o "http://labs.mop.com/apache-mirror//httpd/httpd-2.2.23.tar.gz)

# tar zxvf httpd-2.2.23.tar.gz

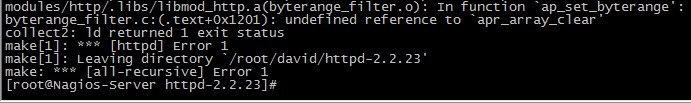
# cd httpd-2.2.23

# ./configure --prefix=/usr/local/apache2

# make && make install



若出现错误：



则在编译时入加 --with-included-apr 即可解决。

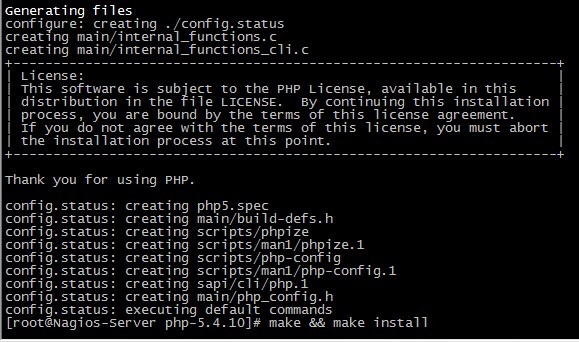
b. 安装Php

# wget <http://cn2.php.net/distributions/php-5.4.10.tar.gz>

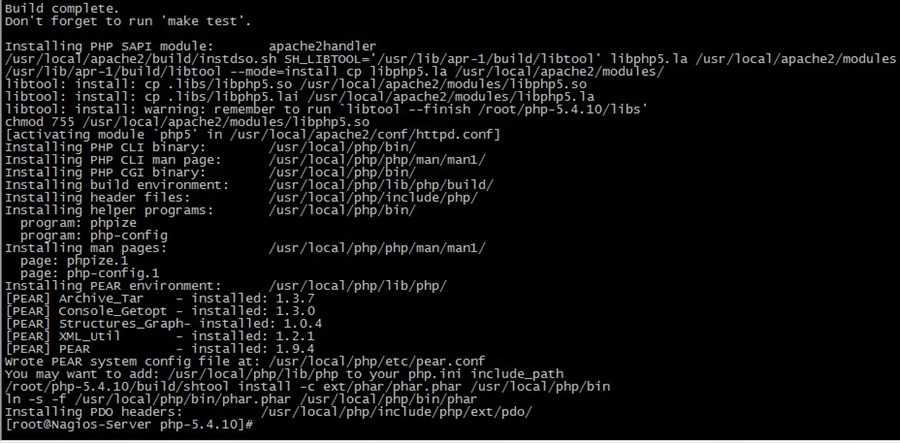
# tar zxvf php-5.4.10.tar.gz

# cd php-5.4.10

# ./configure --prefix=/usr/local/php --with-apxs2=/usr/local/apache2/bin/apxs

[](http://images.cnitblog.com/blog/370046/201301/30172050-6bda75716f614a6ab7f2acf8de422062.jpg)

# make && make install

[](http://images.cnitblog.com/blog/370046/201301/30172054-97af3f5b712041f78d9e44d94ac17bbd.jpg)

c. 配置apache   
找到apache 的配置文件/usr/local/apache2/conf/httpd.conf   
找到：

User daemon

Group daemon

修改为

User nagios

Group nagios

然后找到

<IfModule dir\_module>

　　DirectoryIndex index.html

</IfModule>

修改为

<IfModule dir\_module>

　　DirectoryIndex index.html index.php

</IfModule>

接着增加如下内容：

AddType application/x-httpd-php .php

为了安全起见，一般情况下要让nagios 的web 监控页面必须经过授权才能访问，这需要增加验证配置，即在httpd.conf 文件最后添加如下信息：

#setting for nagios

ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"

<Directory "/usr/local/nagios/sbin">

AuthType Basic

Options ExecCGI

AllowOverride None

Order allow,deny

Allow from all

AuthName "Nagios Access"

AuthUserFile /usr/local/nagios/etc/htpasswd //用于此目录访问身份验证的文件

Require valid-user

</Directory>

Alias /nagios "/usr/local/nagios/share"

<Directory "/usr/local/nagios/share">

AuthType Basic

Options None

AllowOverride None

Order allow,deny

Allow from all

AuthName "nagios Access"

AuthUserFile /usr/local/nagios/etc/htpasswd

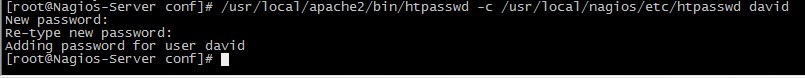
Require valid-user

</Directory>

d. 创建apache目录验证文件

在上面的配置中，指定了目录验证文件htpasswd，下面要创建这个文件：

# /usr/local/apache2/bin/htpasswd -c /usr/local/nagios/etc/htpasswd david

[](http://images.cnitblog.com/blog/370046/201301/30172056-36df7ebedd9d448a8b38a2ccd4633ebe.jpg)   
这样就在/usr/local/nagios/etc 目录下创建了一个htpasswd 验证文件，当通过[http://192.168.1.108/nagios/](http://172.16.1.124/nagios/) 访问时就需要输入用户名和密码了。

e. 查看认证文件的内容

# cat /usr/local/nagios/etc/htpasswd

[26](http://images.cnitblog.com/blog/370046/201301/30172058-a256928a00cf40769bc3a5815401732b.jpg)

f. 启动apache 服务

# /usr/local/apache2/bin/apachectl start

到这里nagios 的安装也就基本完成了，你可以通过web来访问了。



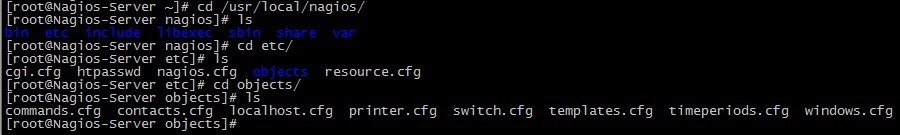


六、配置Nagios

Nagios 主要用于监控一台或者多台本地主机及远程的各种信息，包括本机资源及对外的服务等。默认的Nagios 配置没有任何监控内容，仅是一些模板文件。若要让Nagios 提供服务，就必须修改配置文件，增加要监控的主机和服务，下面将详细介绍。

6.1 默认配置文件介绍

Nagios 安装完毕后，默认的配置文件在/usr/local/nagios/etc目录下。



每个文件或目录含义如下表所示：

|  |  |
| --- | --- |
| 文件名或目录名 | 用途 |
| cgi.cfg | 控制CGI访问的配置文件 |
| nagios.cfg | Nagios 主配置文件 |
| resource.cfg | 变量定义文件，又称为资源文件，在些文件中定义变量，以便由其他配置文件引用，如$USER1$ |
| objects | objects 是一个目录，在此目录下有很多配置文件模板，用于定义Nagios 对象 |
| objects/commands.cfg | 命令定义配置文件，其中定义的命令可以被其他配置文件引用 |
| objects/contacts.cfg | 定义联系人和联系人组的配置文件 |
| objects/localhost.cfg | 定义监控本地主机的配置文件 |
| objects/printer.cfg | 定义监控打印机的一个配置文件模板，默认没有启用此文件 |
| objects/switch.cfg | 定义监控路由器的一个配置文件模板，默认没有启用此文件 |
| objects/templates.cfg | 定义主机和服务的一个模板配置文件，可以在其他配置文件中引用 |
| objects/timeperiods.cfg | 定义Nagios 监控时间段的配置文件 |
| objects/windows.cfg | 监控Windows 主机的一个配置文件模板，默认没有启用此文件 |

6.2 配置文件之间的关系

在nagios的配置过程中涉及到的几个定义有：主机、主机组，服务、服务组，联系人、联系人组，监控时间，监控命令等，从这些定义可以看出，nagios各个配置文件之间是互为关联，彼此引用的。

成功配置出一台nagios监控系统，必须要弄清楚每个配置文件之间依赖与被依赖的关系，最重要的有四点：

第一：定义监控哪些主机、主机组、服务和服务组；   
第二：定义这个监控要用什么命令实现；   
第三：定义监控的时间段；   
第四：定义主机或服务出现问题时要通知的联系人和联系人组。

6.3 配置Nagios

 为了能更清楚的说明问题，同时也为了维护方便，建议将nagios各个定义对象创建独立的配置文件：

 创建hosts.cfg文件来定义主机和主机组

 创建services.cfg文件来定义服务

 用默认的contacts.cfg文件来定义联系人和联系人组

 用默认的commands.cfg文件来定义命令

 用默认的timeperiods.cfg来定义监控时间段

 用默认的templates.cfg文件作为资源引用文件

a. templates.cfg文件

nagios主要用于监控主机资源以及服务，在nagios配置中称为对象，为了不必重复定义一些监控对象，Nagios引入了一个模板配置文件，将一些共性的属性定义成模板，以便于多次引用。这就是templates.cfg的作用。

下面详细介绍下templates.cfg文件中每个参数的含义：

[复制代码](javascript:void(0);)

define contact{

name generic-contact ; 联系人名称

service\_notification\_period 24x7 ; 当服务出现异常时，发送通知的时间段，这个时间段"24x7"在timeperiods.cfg文件中定义

host\_notification\_period 24x7 ; 当主机出现异常时，发送通知的时间段，这个时间段"24x7"在timeperiods.cfg文件中定义

service\_notification\_options w,u,c,r ; 这个定义的是“通知可以被发出的情况”。w即warn，表示警告状态，u即unknown，表示不明状态;  
 ; c即criticle，表示紧急状态，r即recover，表示恢复状态;  
 ; 也就是在服务出现警告状态、未知状态、紧急状态和重新恢复状态时都发送通知给使用者。

host\_notification\_options d,u,r ; 定义主机在什么状态下需要发送通知给使用者，d即down，表示宕机状态;  
 ; u即unreachable，表示不可到达状态，r即recovery，表示重新恢复状态。

service\_notification\_commands notify-service-by-email ; 服务故障时，发送通知的方式，可以是邮件和短信，这里发送的方式是邮件;  
 ; 其中“notify-service-by-email”在commands.cfg文件中定义。

host\_notification\_commands notify-host-by-email ; 主机故障时，发送通知的方式，可以是邮件和短信，这里发送的方式是邮件;  
 ; 其中“notify-host-by-email”在commands.cfg文件中定义。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL CONTACT, JUST A TEMPLATE!

}

define host{

name generic-host ; 主机名称，这里的主机名，并不是直接对应到真正机器的主机名;  
 ; 乃是对应到在主机配置文件里所设定的主机名。

notifications\_enabled 1 ; Host notifications are enabled

event\_handler\_enabled 1 ; Host event handler is enabled

flap\_detection\_enabled 1 ; Flap detection is enabled

failure\_prediction\_enabled 1 ; Failure prediction is enabled

process\_perf\_data 1 ; 其值可以为0或1，其作用为是否启用Nagios的数据输出功能;  
 ; 如果将此项赋值为1，那么Nagios就会将收集的数据写入某个文件中，以备提取。

retain\_status\_information 1 ; Retain status information across program restarts

retain\_nonstatus\_information 1 ; Retain non-status information across program restarts

notification\_period 24x7 ; 指定“发送通知”的时间段，也就是可以在什么时候发送通知给使用者。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL HOST, JUST A TEMPLATE!

}

define host{

name linux-server ; 主机名称

use generic-host ; use表示引用，也就是将主机generic-host的所有属性引用到linux-server中来;  
 ; 在nagios配置中，很多情况下会用到引用。

check\_period 24x7 ; 这里的check\_period告诉nagios检查主机的时间段

check\_interval 5 ; nagios对主机的检查时间间隔，这里是5分钟。

retry\_interval 1 ; 重试检查时间间隔，单位是分钟。

max\_check\_attempts 10 ; nagios对主机的最大检查次数，也就是nagios在检查发现某主机异常时，并不马上判断为异常状况;  
 ; 而是多试几次，因为有可能只是一时网络太拥挤，或是一些其他原因，让主机受到了一点影响;  
 ; 这里的10就是最多试10次的意思。

check\_command check-host-alive ; 指定检查主机状态的命令，其中“check-host-alive”在commands.cfg文件中定义。

notification\_period 24x7 ; 主机故障时，发送通知的时间范围，其中“workhours”在timeperiods.cfg中进行了定义;  
 ; 下面会陆续讲到。

notification\_interval 10 ; 在主机出现异常后，故障一直没有解决，nagios再次对使用者发出通知的时间。单位是分钟;  
 ; 如果你觉得，所有的事件只需要一次通知就够了，可以把这里的选项设为0

notification\_options d,u,r ; 定义主机在什么状态下可以发送通知给使用者，d即down，表示宕机状态;  
 ; u即unreachable，表示不可到达状态;  
 ; r即recovery，表示重新恢复状态。

contact\_groups ts ; 指定联系人组，这个“admins”在contacts.cfg文件中定义。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL HOST, JUST A TEMPLATE!

}

define host{

name windows-server ; The name of this host template

use generic-host ; Inherit default values from the generic-host template

check\_period 24x7 ; By default, Windows servers are monitored round the clock

check\_interval 5 ; Actively check the server every 5 minutes

retry\_interval 1 ; Schedule host check retries at 1 minute intervals

max\_check\_attempts 10 ; Check each server 10 times (max)

check\_command check-host-alive ; Default command to check if servers are "alive"

notification\_period 24x7 ; Send notification out at any time - day or night

notification\_interval 10 ; Resend notifications every 30 minutes

notification\_options d,r ; Only send notifications for specific host states

contact\_groups ts ; Notifications get sent to the admins by default

hostgroups windows-servers ; Host groups that Windows servers should be a member of

register 0 ; DONT REGISTER THIS - ITS JUST A TEMPLATE

}

define service{

name generic-service ; 定义一个服务名称

active\_checks\_enabled 1 ; Active service checks are enabled

passive\_checks\_enabled 1 ; Passive service checks are enabled/accepted

parallelize\_check 1 ; Active service checks should be parallelized;  
 ; (disabling this can lead to major performance problems)

obsess\_over\_service 1 ; We should obsess over this service (if necessary)

check\_freshness 0 ; Default is to NOT check service 'freshness'

notifications\_enabled 1 ; Service notifications are enabled

event\_handler\_enabled 1 ; Service event handler is enabled

flap\_detection\_enabled 1 ; Flap detection is enabled

failure\_prediction\_enabled 1 ; Failure prediction is enabled

process\_perf\_data 1 ; Process performance data

retain\_status\_information 1 ; Retain status information across program restarts

retain\_nonstatus\_information 1 ; Retain non-status information across program restarts

is\_volatile 0 ; The service is not volatile

check\_period 24x7 ; 这里的check\_period告诉nagios检查服务的时间段。

max\_check\_attempts 3 ; nagios对服务的最大检查次数。

normal\_check\_interval 5 ; 此选项是用来设置服务检查时间间隔，也就是说，nagios这一次检查和下一次检查之间所隔的时间;  
 ; 这里是5分钟。

retry\_check\_interval 2 ; 重试检查时间间隔，单位是分钟。

contact\_groups ts ; 指定联系人组

notification\_options w,u,c,r ; 这个定义的是“通知可以被发出的情况”。w即warn，表示警告状态;  
 ; u即unknown，表示不明状态;  
 ; c即criticle，表示紧急状态，r即recover，表示恢复状态;  
 ; 也就是在服务出现警告状态、未知状态、紧急状态和重新恢复后都发送通知给使用者。

notification\_interval 10 ; Re-notify about service problems every hour

notification\_period 24x7 ; 指定“发送通知”的时间段，也就是可以在什么时候发送通知给使用者。

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!

}

define service{

name local-service ; The name of this service template

use generic-service ; Inherit default values from the generic-service definition

max\_check\_attempts 4 ; Re-check the service up to 4 times in order to determine its final (hard) state

normal\_check\_interval 5 ; Check the service every 5 minutes under normal conditions

retry\_check\_interval 1 ; Re-check the service every minute until a hard state can be determined

register 0 ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!

}

b. resource.cfg文件

resource.cfg是nagios的变量定义文件，文件内容只有一行：

$USER1$=/usr/local/nagios/libexec

其中，变量$USER1$指定了安装nagios插件的路径，如果把插件安装在了其它路径，只需在这里进行修改即可。需要注意的是，变量必须先定义，然后才能在其它配置文件中进行引用。

c. commands.cfg文件

此文件默认是存在的，无需修改即可使用，当然如果有新的命令需要加入时，在此文件进行添加即可。

#notify-host-by-email命令的定义   
define command{

command\_name notify-host-by-email #命令名称，即定义了一个主机异常时发送邮件的命令。

command\_line /usr/bin/printf "%b" "\*\*\*\*\* Nagios \*\*\*\*\*\n\nNotification Type: $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\nState: $HOSTSTATE$\nAddress: $HOSTADDRESS$\nInfo: $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" | /bin/mail -s "\*\* $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is $HOSTSTATE$ \*\*" $CONTACTEMAIL$ #命令具体的执行方式。

}

#notify-service-by-email命令的定义   
define command{

command\_name notify-service-by-email #命令名称，即定义了一个服务异常时发送邮件的命令

command\_line /usr/bin/printf "%b" "\*\*\*\*\* Nagios \*\*\*\*\*\n\nNotification Type: $NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress: $HOSTADDRESS$\nState: $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\nAdditional Info:\n\n$SERVICEOUTPUT$\n" | /bin/mail -s "\*\* $NOTIFICATIONTYPE$ Service Alert: $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ \*\*" $CONTACTEMAIL$

}  
#check-host-alive命令的定义

define command{

command\_name check-host-alive #命令名称，用来检测主机状态。

command\_line $USER1$/check\_ping -H $HOSTADDRESS$ -w 3000.0,80% -c 5000.0,100% -p 5   
 # 这里的变量$USER1$在resource.cfg文件中进行定义，即$USER1$=/usr/local/nagios/libexec;  
 # 那么check\_ping的完整路径为/usr/local/nagios/libexec/check\_ping;  
 # “-w 3000.0,80%”中“-w”说明后面的一对值对应的是“WARNING”状态，“80%”是其临界值。  
 # “-c 5000.0,100%”中“-c”说明后面的一对值对应的是“CRITICAL”，“100%”是其临界值。  
 # “-p 1”说明每次探测发送一个包。  
 }

define command{

command\_name check\_local\_disk

command\_line $USER1$/check\_disk -w $ARG1$ -c $ARG2$ -p $ARG3$ #$ARG1$是指在调用这个命令的时候，命令后面的第一个参数。

}

define command{

command\_name check\_local\_load

command\_line $USER1$/check\_load -w $ARG1$ -c $ARG2$

}

define command{

command\_name check\_local\_procs

command\_line $USER1$/check\_procs -w $ARG1$ -c $ARG2$ -s $ARG3$

}

define command{

command\_name check\_local\_users

command\_line $USER1$/check\_users -w $ARG1$ -c $ARG2$

}

define command{

command\_name check\_local\_swap

command\_line $USER1$/check\_swap -w $ARG1$ -c $ARG2$

}

define command{

command\_name check\_ftp

command\_line $USER1$/check\_ftp -H $HOSTADDRESS$ $ARG1$

}

define command{

command\_name check\_http

command\_line $USER1$/check\_http -I $HOSTADDRESS$ $ARG1$

}

define command{

command\_name check\_ssh

command\_line $USER1$/check\_ssh $ARG1$ $HOSTADDRESS$

}

define command{

command\_name check\_ping

command\_line $USER1$/check\_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -p 5

}

define command{

command\_name check\_nt

command\_line $USER1$/check\_nt -H $HOSTADDRESS$ -p 12489 -v $ARG1$ $ARG2$

}

d. hosts.cfg文件

此文件默认不存在，需要手动创建，hosts.cfg主要用来指定被监控的主机地址以及相关属性信息，根据实验目标配置如下：

define host{

use linux-server #引用主机linux-server的属性信息，linux-server主机在templates.cfg文件中进行了定义。

host\_name Nagios-Linux #主机名

alias Nagios-Linux #主机别名

address 192.168.1.111 #被监控的主机地址，这个地址可以是ip，也可以是域名。

}

#定义一个主机组

define hostgroup{

hostgroup\_name bsmart-servers #主机组名称，可以随意指定。

alias bsmart servers #主机组别名

members Nagios-Linux #主机组成员，其中“Nagios-Linux”就是上面定义的主机。   
 }

注意：在/usr/local/nagios/etc/objects 下默认有localhost.cfg 和windows.cfg 这两个配置文件，localhost.cfg 文件是定义监控主机本身的，windows.cfg 文件是定义windows 主机的，其中包括了对host 和相关services 的定义。所以在本次实验中，将直接在localhost.cfg 中定义监控主机（Nagios-Server），在windows.cfg中定义windows 主机（Nagios-Windows）。根据自己的需要修改其中的相关配置，详细如下：

localhost.cfg

define host{

use linux-server ; Name of host template to use

; This host definition will inherit all variables that are defined

; in (or inherited by) the linux-server host template definition.

host\_name Nagios-Server

alias Nagios-Server

address 127.0.0.1

}

define hostgroup{

hostgroup\_name linux-servers ; The name of the hostgroup

alias Linux Servers ; Long name of the group

members Nagios-Server ; Comma separated list of hosts that belong to this group

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description PING

check\_command check\_ping!100.0,20%!500.0,60%

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Root Partition

check\_command check\_local\_disk!20%!10%!/

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Current Users

check\_command check\_local\_users!20!50

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Total Processes

check\_command check\_local\_procs!250!400!RSZDT

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Current Load

check\_command check\_local\_load!5.0,4.0,3.0!10.0,6.0,4.0

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description Swap Usage

check\_command check\_local\_swap!20!10

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description SSH

check\_command check\_ssh

notifications\_enabled 0

}

define service{

use local-service ; Name of service template to use

host\_name Nagios-Server

service\_description HTTP

check\_command check\_http

notifications\_enabled 0

}

windows.cfg

define host{

use windows-server ; Inherit default values from a template

host\_name Nagios-Windows ; The name we're giving to this host

alias My Windows Server ; A longer name associated with the host

address 192.168.1.113 ; IP address of the host

}

define hostgroup{

hostgroup\_name windows-servers ; The name of the hostgroup

alias Windows Servers ; Long name of the group

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description NSClient++ Version

check\_command check\_nt!CLIENTVERSION

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description Uptime

check\_command check\_nt!UPTIME

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description CPU Load

check\_command check\_nt!CPULOAD!-l 5,80,90

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description Memory Usage

check\_command check\_nt!MEMUSE!-w 80 -c 90

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description C:\ Drive Space

check\_command check\_nt!USEDDISKSPACE!-l c -w 80 -c 90

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description W3SVC

check\_command check\_nt!SERVICESTATE!-d SHOWALL -l W3SVC

}

define service{

use generic-service

host\_name Nagios-Windows

service\_description Explorer

check\_command check\_nt!PROCSTATE!-d SHOWALL -l Explorer.exe

}

e. services.cfg文件

此文件默认也不存在，需要手动创建，services.cfg文件主要用于定义监控的服务和主机资源，例如监控http服务、ftp服务、主机磁盘空间、主机系统负载等等。Nagios-Server 和Nagios-Windows 相关服务已在相应的配置文件中定义，所以这里只需要定义Nagios-Linux 相关服务即可，这里只定义一个检测是否存活的服务来验证配置文件的正确性，其他服务的定义将在后面讲到。

define service{

use local-service #引用local-service服务的属性值，local-service在templates.cfg文件中进行了定义。

host\_name Nagios-Linux #指定要监控哪个主机上的服务，“Nagios-Server”在hosts.cfg文件中进行了定义。

service\_description check-host-alive #对监控服务内容的描述，以供维护人员参考。

check\_command check-host-alive #指定检查的命令。  
 }

f. contacts.cfg文件

contacts.cfg是一个定义联系人和联系人组的配置文件，当监控的主机或者服务出现故障，nagios会通过指定的通知方式（邮件或者短信）将信息发给这里指定的联系人或者使用者。

define contact{

contact\_name David #联系人的名称,这个地方不要有空格

use                             generic-contact #引用generic-contact的属性信息，其中“generic-contact”在templates.cfg文件中进行定义

alias                           Nagios Admin

email david.tang@bsmart.cn

}

define contactgroup{

contactgroup\_name ts #联系人组的名称,同样不能空格

alias Technical Support #联系人组描述

members David #联系人组成员，其中“david”就是上面定义的联系人，如果有多个联系人则以逗号相隔

}

g. timeperiods.cfg文件

此文件只要用于定义监控的时间段，下面是一个配置好的实例：

#下面是定义一个名为24x7的时间段，即监控所有时间段

define timeperiod{

timeperiod\_name 24x7 #时间段的名称,这个地方不要有空格

alias 24 Hours A Day, 7 Days A Week

sunday 00:00-24:00

monday 00:00-24:00

tuesday 00:00-24:00

wednesday 00:00-24:00

thursday 00:00-24:00

friday 00:00-24:00

saturday 00:00-24:00

}

#下面是定义一个名为workhours的时间段，即工作时间段。

define timeperiod{

timeperiod\_name workhours

alias Normal Work Hours

monday 09:00-17:00

tuesday 09:00-17:00

wednesday 09:00-17:00

thursday 09:00-17:00

friday 09:00-17:00

}

h. cgi.cfg文件

此文件用来控制相关cgi脚本，如果想在nagios的web监控界面执行cgi脚本，例如重启nagios进程、关闭nagios通知、停止nagios主机检测等，这时就需要配置cgi.cfg文件了。  
由于nagios的web监控界面验证用户为david，所以只需在cgi.cfg文件中添加此用户的执行权限就可以了，需要修改的配置信息如下：

default\_user\_name=david

authorized\_for\_system\_information=nagiosadmin,david

authorized\_for\_configuration\_information=nagiosadmin,david

authorized\_for\_system\_commands=david

authorized\_for\_all\_services=nagiosadmin,david

authorized\_for\_all\_hosts=nagiosadmin,david

authorized\_for\_all\_service\_commands=nagiosadmin,david

authorized\_for\_all\_host\_commands=nagiosadmin,david

i. nagios.cfg文件

nagios.cfg默认的路径为/usr/local/nagios/etc/nagios.cfg，是nagios的核心配置文件，所有的对象配置文件都必须在这个文件中进行定义才能发挥其作用，这里只需将对象配置文件在Nagios.cfg文件中进行引用即可。

log\_file=/usr/local/nagios/var/nagios.log # 定义nagios日志文件的路径

cfg\_file=/usr/local/nagios/etc/objects/commands.cfg # “cfg\_file”变量用来引用对象配置文件，如果有更多的对象配置文件，在这里依次添加即可。  
cfg\_file=/usr/local/nagios/etc/objects/contacts.cfg  
cfg\_file=/usr/local/nagios/etc/objects/hosts.cfg  
cfg\_file=/usr/local/nagios/etc/objects/services.cfg  
cfg\_file=/usr/local/nagios/etc/objects/timeperiods.cfg

cfg\_file=/usr/local/nagios/etc/objects/templates.cfg  
cfg\_file=/usr/local/nagios/etc/objects/localhost.cfg # 本机配置文件  
cfg\_file=/usr/local/nagios/etc/objects/windows.cfg # windows 主机配置文件

object\_cache\_file=/usr/local/nagios/var/objects.cache # 该变量用于指定一个“所有对象配置文件”的副本文件，或者叫对象缓冲文件

precached\_object\_file=/usr/local/nagios/var/objects.precache  
resource\_file=/usr/local/nagios/etc/resource.cfg # 该变量用于指定nagios资源文件的路径，可以在nagios.cfg中定义多个资源文件。

status\_file=/usr/local/nagios/var/status.dat # 该变量用于定义一个状态文件，此文件用于保存nagios的当前状态、注释和宕机信息等。

status\_update\_interval=10 # 该变量用于定义状态文件（即status.dat）的更新时间间隔，单位是秒，最小更新间隔是1秒。

nagios\_user=nagios # 该变量指定了Nagios进程使用哪个用户运行。  
nagios\_group=nagios # 该变量用于指定Nagios使用哪个用户组运行。

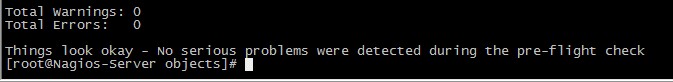
check\_external\_commands=1 # 该变量用于设置是否允许nagios在web监控界面运行cgi命令;  
 # 也就是是否允许nagios在web界面下执行重启nagios、停止主机/服务检查等操作;  
 # “1”为运行，“0”为不允许。

command\_check\_interval=10s # 该变量用于设置nagios对外部命令检测的时间间隔，如果指定了一个数字加一个"s"(如10s);  
 # 那么外部检测命令的间隔是这个数值以秒为单位的时间间隔;  
 # 如果没有用"s"，那么外部检测命令的间隔是以这个数值的“时间单位”的时间间隔。  
interval\_length=60 # 该变量指定了nagios的时间单位，默认值是60秒，也就是1分钟;  
 # 即在nagios配置中所有的时间单位都是分钟。

6.4 验证Nagios 配置文件的正确性

Nagios 在验证配置文件方面做的非常到位，只需通过一个命令即可完成：

# /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg



Nagios提供的这个验证功能非常有用，在错误信息中通常会打印出错误的配置文件以及文件中的哪一行，这使得nagios的配置变得非常容易，报警信息通常是可以忽略的，因为一般那些只是建议性的。

看到上面这些信息就说明没问题了，然后启动Nagios 服务。

七、Nagios的启动与停止

7.1 启动Nagios

a. 通过初始化脚本启动nagios

# /etc/init.d/nagios start  
or  
# service nagios start

b. 手工方式启动nagios

通过nagios命令的“-d”参数来启动nagios守护进程：

# /usr/local/nagios/bin/nagios -d /usr/local/nagios/etc/nagios.cfg

7.2 重启Nagios

当修改了配置文件让其生效时，需要重启/重载Nagios服务。

a. 通过初始化脚本来重启nagios

# /etc/init.d/nagios reload  
or  
# /etc/init.d/nagios restart  
or  
# service nagios restart

b. 通过web监控页重启nagios

可以通过web监控页的 "Process Info" -> "Restart the Nagios process"来重启nagios



c. 手工方式平滑重启

# kill -HUP <nagios\_pid>

7.3 停止Nagios

a. 通过初始化脚本关闭nagios服务

# /etc/init.d/nagios stop  
or  
# service nagios stop

b. 通过web监控页停止nagios

可以通过web监控页的 "Process Info" -> "Shutdown the Nagios process"来停止nagios



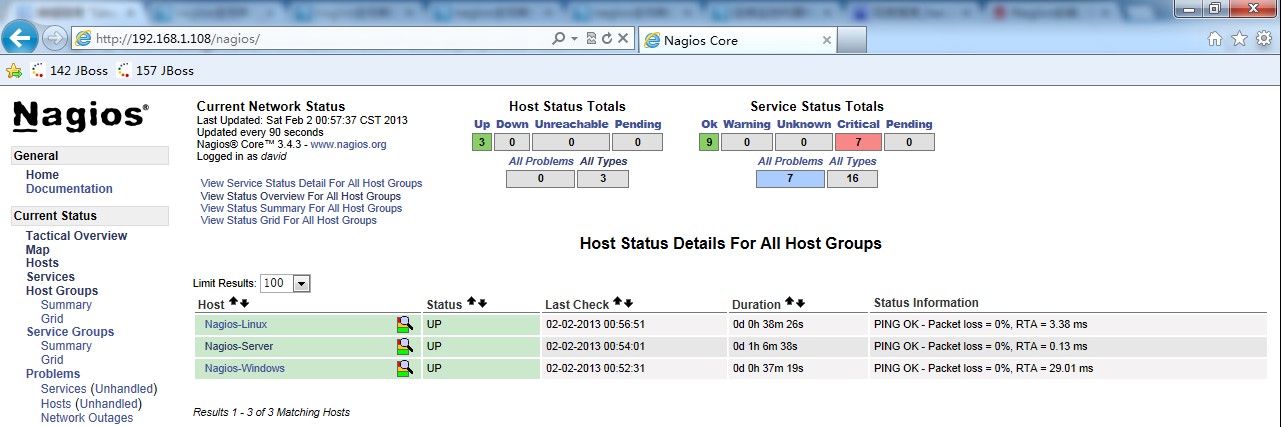
c. 手工方式停止Nagios

# kill <nagios\_pid>

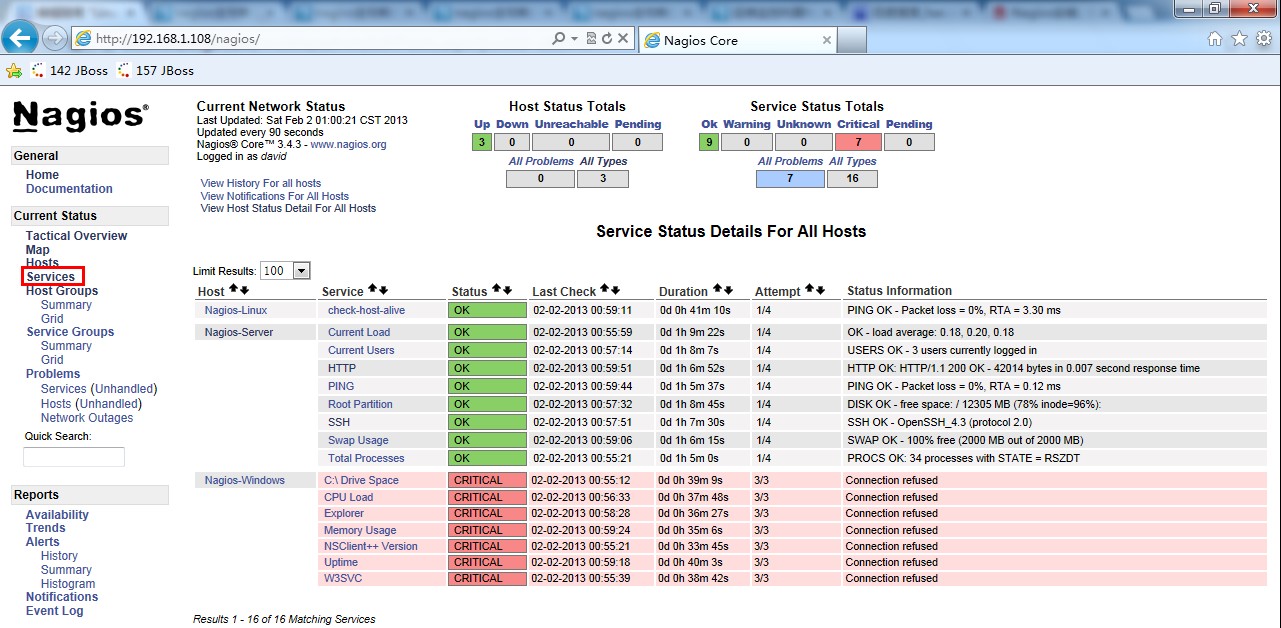
八、查看初步配置情况

8.1 启动完成之后，登录Nagios Web监控页[http://192.168.1.108/nagios/](http://172.16.1.124/nagios/) 查看相关信息。

8.2 点击左面的Current Status -> Hosts 可以看到所定义的三台主机已经全部UP了。



8.3 点击Current Status -> Services 查看服务监控情况。

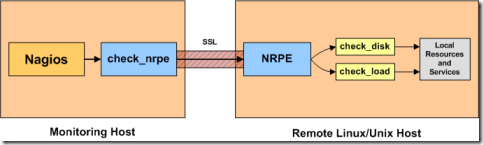


看到Nagios-Linux和Nagios-Server的服务状态已经OK了，但是Nagios-Windows的服务状态为CRITICAL，Status Information 提示Connection refused。因为Nagios-Windows上还未安装插件，内部服务还无法查看，所以出现这种情况。将在下面具体讲解。

九、利用NRPE监控远程Linux上的“本地信息”

上面已经对远程Linux 主机是否存活做了监控，而判断远程机器是否存活，我们可以使用ping 工具对其监测。还有一些远程主机服务，例如ftp、ssh、http，都是对外开放的服务，即使不用Nagios，我们也可以试的出来，随便找一台机器看能不能访问这些服务就行了。但是对于像磁盘容量，cpu负载这样的“本地信息”，Nagios只能监测自己所在的主机，而对其他的机器则显得有点无能为力。毕竟没得到被控主机的适当权限是不可能得到这些信息的。为了解决这个问题，nagios有这样一个附加组件--“NRPE”，用它就可以完成对Linux 类型主机"本地信息”的监控。

9.1 NRPE 工作原理

[](http://images.cnitblog.com/blog/370046/201301/30172018-ee257c664e12472590da4a3b503be33e.png)

NRPE 总共由两部分组成：

check\_nrpe 插件，位于监控主机上

NRPE daemon，运行在远程的Linux主机上(通常就是被监控机)

按照上图，整个的监控过程如下：

当Nagios 需要监控某个远程Linux 主机的服务或者资源情况时：

Nagios 会运行check\_nrpe 这个插件，告诉它要检查什么；

check\_nrpe 插件会连接到远程的NRPE daemon，所用的方式是SSL；

NRPE daemon 会运行相应的Nagios 插件来执行检查；

NRPE daemon 将检查的结果返回给check\_nrpe 插件，插件将其递交给nagios做处理。

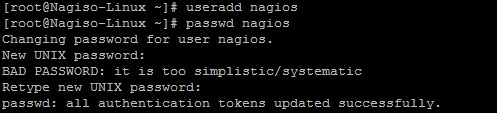
注意：NRPE daemon 需要Nagios 插件安装在远程的Linux主机上，否则，daemon不能做任何的监控。

9.2 在被监控机（Nagios-Linux）上

a. 增加用户&设定密码

# useradd nagios

# passwd nagios



b. 安装Nagios 插件

# tar zxvf nagios-plugins-1.4.16.tar.gz

# cd nagios-plugins-1.4.16

# ./configure --prefix=/usr/local/nagios

# make && make install

这一步完成后会在/usr/local/nagios/下生成三个目录include、libexec和share。

http://images.cnitblog.com/blog/370046/201302/02152814-8ad7b743e94d47e4a273d26f17fc039b.jpg

修改目录权限

# chown nagios.nagios /usr/local/nagios

# chown -R nagios.nagios /usr/local/nagios/libexec

http://images.cnitblog.com/blog/370046/201302/02152955-bba448f8c550421199e95f5abcd2a367.jpg

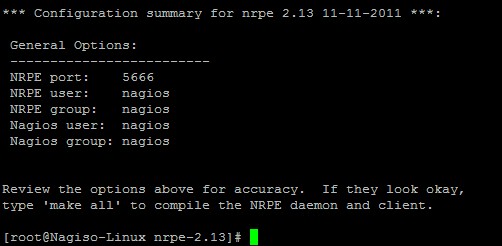
c. 安装NRPE

# wget http://prdownloads.sourceforge.net/sourceforge/nagios/nrpe-2.13.tar.gz

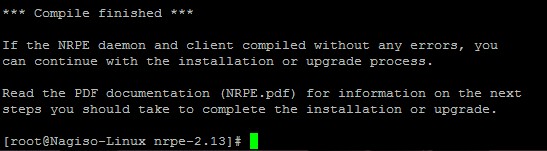
# tar zxvf nrpe-2.13.tar.gz

# cd nrpe-2.13

# ./configure



# make all



接下来安装NPRE插件，daemon和示例配置文件。

c.1 安装check\_nrpe 这个插件

# make install-plugin

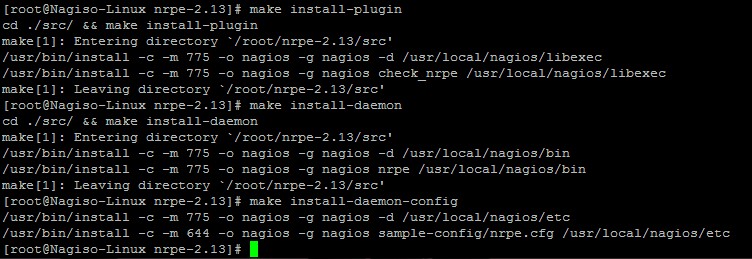
监控机需要安装check\_nrpe 这个插件，被监控机并不需要，我们在这里安装它只是为了测试目的。

c.2 安装deamon

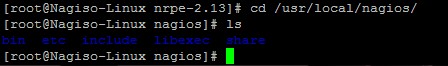
# make install-daemon

c.3 安装配置文件

# make install-daemon-config



现在再查看nagios 目录就会发现有5个目录了



按照安装文档的说明，是将NRPE deamon作为xinetd下的一个服务运行的。在这样的情况下xinetd就必须要先安装好，不过一般系统已经默认安装了。

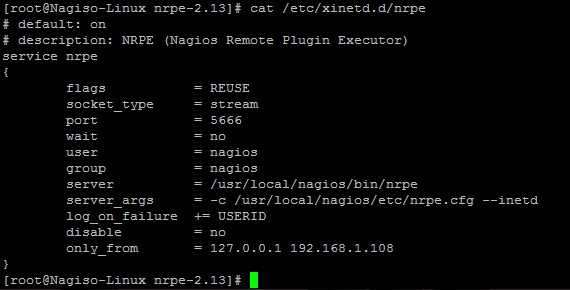
d. 安装xinted 脚本

# make install-xinetd

http://images.cnitblog.com/blog/370046/201302/02154307-bee4a49f8d87420a87899cd9c840c6d9.jpg

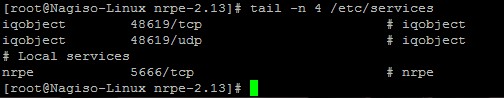
可以看到创建了这个文件/etc/xinetd.d/nrpe。

编辑这个脚本：



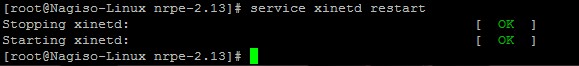
在only\_from 后增加监控主机的IP地址。

编辑/etc/services 文件，增加NRPE服务

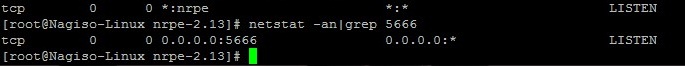


重启xinted 服务

# service xinetd restart



查看NRPE 是否已经启动



可以看到5666端口已经在监听了。

e. 测试NRPE是否则正常工作

使用上面在被监控机上安装的check\_nrpe 这个插件测试NRPE 是否工作正常。

# /usr/local/nagios/libexec/check\_nrpe -H localhost

会返回当前NRPE的版本

http://images.cnitblog.com/blog/370046/201302/02160042-6ae7c0101c264ff8b8076192ace36a23.jpg

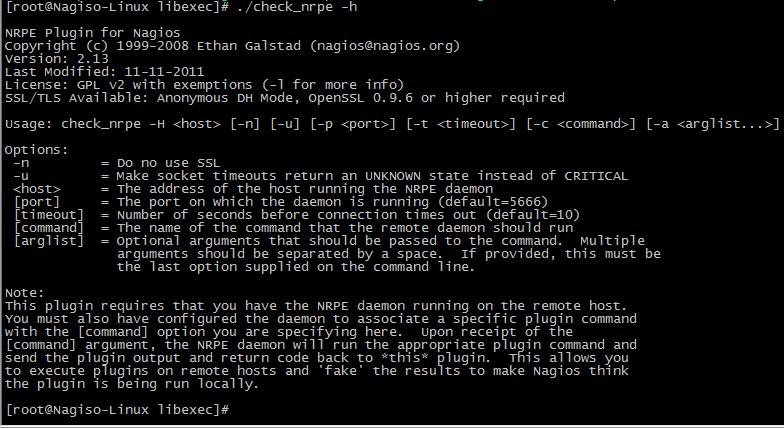
也就是在本地用check\_nrpe连接nrpe daemon是正常的。

注：为了后面工作的顺利进行，注意本地防火墙要打开5666能让外部的监控机访问。

f. check\_nrpe 命令用法

查看check\_nrpe 命令用法

# /usr/local/nagios/libexec/check\_nrpe –h



可以看到用法是：

check\_nrpe –H 被监控的主机 -c 要执行的监控命令

注意：-c 后面接的监控命令必须是nrpe.cfg 文件中定义的。也就是NRPE daemon只运行nrpe.cfg中所定义的命令。

g. 查看NRPE的监控命令

# cd /usr/local/nagios/etc

# cat nrpe.cfg |grep -v "^#"|grep -v "^$"

[root@Nagiso-Linux etc]# cat nrpe.cfg |grep -v "^#"|grep -v "^$"

log\_facility=daemon

pid\_file=/var/run/nrpe.pid

server\_port=5666

nrpe\_user=nagios

nrpe\_group=nagios

allowed\_hosts=127.0.0.1

dont\_blame\_nrpe=0

debug=0

command\_timeout=60

connection\_timeout=300

command[check\_users]=/usr/local/nagios/libexec/check\_users -w 5 -c 10

command[check\_load]=/usr/local/nagios/libexec/check\_load -w 15,10,5 -c 30,25,20

command[check\_sda1]=/usr/local/nagios/libexec/check\_disk -w 20% -c 10% -p /dev/sda1

command[check\_zombie\_procs]=/usr/local/nagios/libexec/check\_procs -w 5 -c 10 -s Z

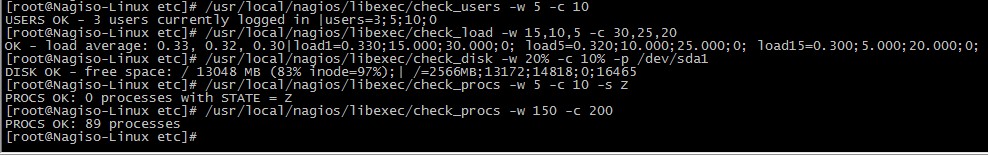
command[check\_total\_procs]=/usr/local/nagios/libexec/check\_procs -w 150 -c 200

[root@Nagiso-Linux etc]#

红色部分是命令名，也就是check\_nrpe 的-c 参数可以接的内容，等号 “=” 后面是实际执行的插件程序（这与commands.cfg 中定义命令的形式十分相似，只不过是写在了一行）。也就是说check\_users 就是等号后面/usr/local/nagios/libexec/check\_users -w 5 -c 10 的简称。

我们可以很容易知道上面这5行定义的命令分别是检测登陆用户数，cpu负载，sda1的容量，僵尸进程，总进程数。各条命令具体的含义见插件用法（执行“插件程序名 –h”）。

由于-c 后面只能接nrpe.cfg 中定义的命令，也就是说现在我们只能用上面定义的这五条命令。我们可以在本机实验一下。



9.3 在监控主机（Nagios-Server）上

之前已经将Nagios运行起来了，现在要做的事情是：

安装check\_nrpe 插件；

在commands.cfg 中创建check\_nrpe 的命令定义，因为只有在commands.cfg 中定义过的命令才能在services.cfg 中使用；

创建对被监控主机的监控项目；

9.3.1 安装check\_nrpe 插件

# tar zxvf nrpe-2.13.tar.gz

# cd nrpe-2.13

# ./configure

# make all

# make install-plugin

只运行这一步就行了，因为只需要check\_nrpe插件。

在Nagios-Linux 上我们已经装好了nrpe，现在我们测试一下监控机使用check\_nrpe 与被监控机运行的nrpe daemon之间的通信。

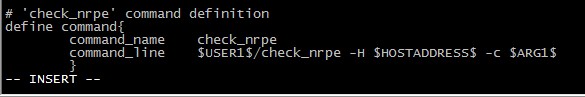
http://images.cnitblog.com/blog/370046/201302/02165718-6aa53e921dc2432a9738a5638aaa17e3.jpg

看到已经正确返回了NRPE的版本信息，说明一切正常。

9.3.2 在commands.cfg中增加对check\_nrpe的定义

# vi /usr/local/nagios/etc/objects/commands.cfg

在最后面增加如下内容：



意义如下：

# 'check\_nrpe' command definition

define command{

command\_name check\_nrpe # 定义命令名称为check\_nrpe,在services.cfg中要使用这个名称.

command\_line $USER1$/check\_nrpe -H $HOSTADDRESS$ -c $ARG1$ #这是定义实际运行的插件程序.  
 # 这个命令行的书写要完全按照check\_nrpe这个命令的用法,不知道用法的就用check\_nrpe –h查看.

}

-c 后面带的$ARG1$ 参数是传给nrpe daemon 执行的检测命令，之前说过了它必须是nrpe.cfg 中所定义的那5条命令中的其中一条。在services.cfg 中使用check\_nrpe 的时候要用 “!” 带上这个参数。

9.3.3 定义对Nagios-Linux 主机的监控

下面就可以在services.cfg 中定义对Nagios-Linux 主机的监控了。

define service{

use local-service

host\_name Nagios-Linux

service\_description Current Load

check\_command check\_nrpe!check\_load

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Check Disk sda1

check\_command check\_nrpe!check\_sda1

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Total Processes

check\_command check\_nrpe!check\_total\_procs

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Current Users

check\_command check\_nrpe!check\_users

}

define service{

use local-service

host\_name Nagios-Linux

service\_description Check Zombie Procs

check\_command check\_nrpe!check\_zombie\_procs

}

还有一个任务是要监控Nagios-Linux 的swap 使用情况。但是在nrpe.cfg 中默认没有定义这个监控功能的命令。怎么办？手动在nrpe.cfg 中添加，也就是自定义NRPE命令。

现在我们要监控swap 分区，如果空闲空间小于20%则为警告状态 -> warning；如果小于10%则为严重状态 -> critical。我们可以查得需要使用check\_swap插件，完整的命令行应该是下面这样。

# /usr/local/nagios/libexec/check\_swap -w 20% -c 10%  
  
在被监控机（Nagios-Linux）上增加check\_swap 命令的定义

# vi /usr/local/nagios/etc/nrpe.cfg

增加下面这一行

command[check\_swap]=/usr/local/nagios/libexec/check\_swap -w 20% -c 10%

我们知道check\_swap 现在就可以作为check\_nrpe 的-c 的参数使用了

修改了配置文件，当然要重启。

如果你是以独立的daemon运行的nrpe，那么需要手动重启；如果你是在xinetd 下面运行的，则不需要。

由于本实验中nrpe 是xinetd 下运行的，所以不需要重启服务。  
  
在监控机（Nagios-Server）上增加这个check\_swap 监控项目

define service{

use local-service

host\_name Nagios-Linux

service\_description Check Swap

check\_command check\_nrpe!check\_swap

}

同理，Nagios-Linux 上我还开启了http 服务，需要监控一下，按照上面的做法，在被监控机（Nagios-Linux）上增加check\_http 命令的定义

# vi /usr/local/nagios/etc/nrpe.cfg

增加下面这一行

command[check\_http]=/usr/local/nagios/libexec/check\_http -I 127.0.0.1

在监控机（Nagios-Server）上增加check\_http 监控项目

define service{

use local-service

host\_name Nagios-Linux

service\_description HTTP

check\_command check\_nrpe!check\_http

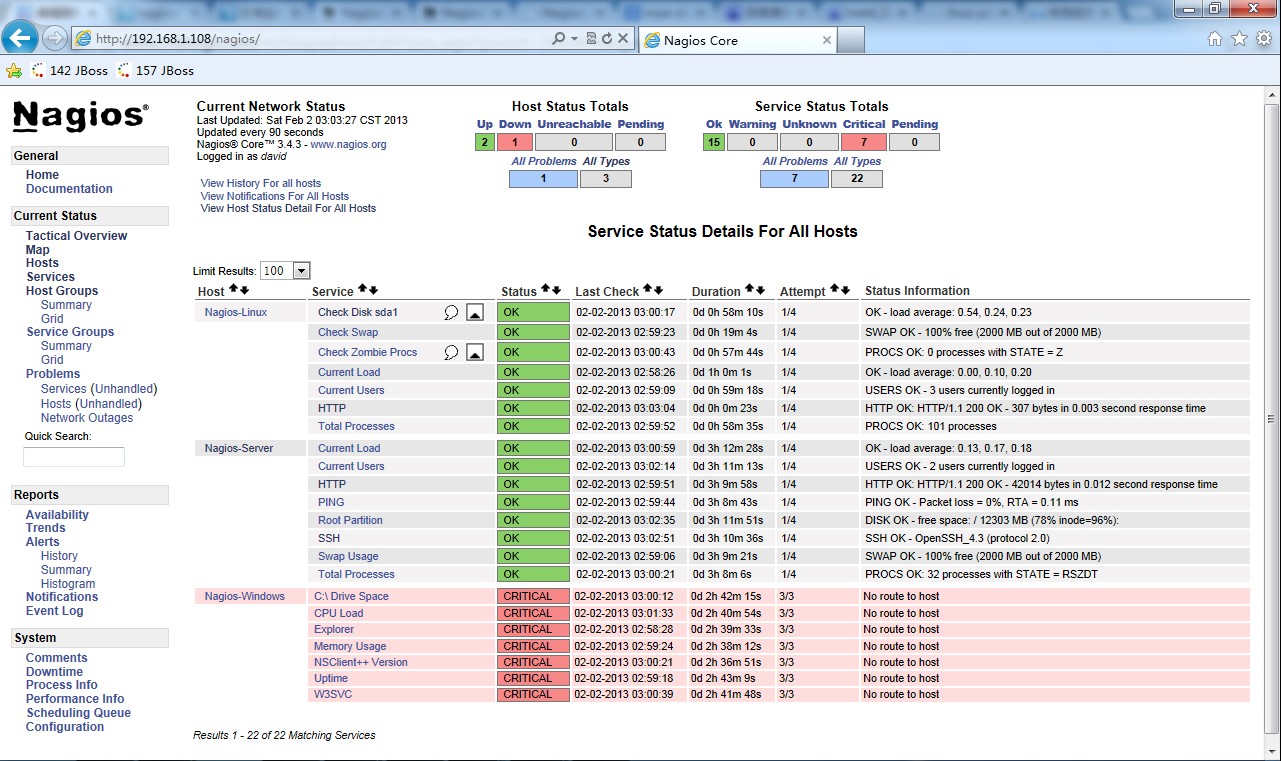
}

所有的配置文件已经修改好了，现在重启Nagios。

# service nagios restart

9.3.4 查看配置情况

登录Nagios Web监控页[http://192.168.1.108/nagios/](http://172.16.1.124/nagios/) 查看相关信息。

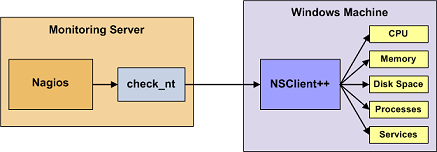


可以看到，对于Nagios-Server 和Nagios-Linux 上的相关服务的监控已经成功了，还有Nagios-Windows 上的服务还没有定义，下面讲到。

十、利用NSClient++监控远程Windows上的“本地信息”

在Nagios的libexec下有check\_nt这个插件，它就是用来检查windows机器的服务的。其功能类似于check\_nrpe。不过还需要搭配另外一个软件NSClient++，它则类似于NRPE。

NSClient++的原理如下图



可以看到NSClient与nrpe最大的区别就是：

被监控机上安装有nrpe，并且还有插件，最终的监控是由这些插件来进行的。当监控主机将监控请求发给nrpe后，nrpe调用插件来完成监控。

NSClient++则不同，被监控机上只安装NSClient，没有任何的插件。当监控主机将监控请求发给NSClient++后，NSClient直接完成监控，所有的监控是由NSClient完成的。

这也说明了NSClient++的一个很大的问题：不灵活、没有可扩展性。它只能完成自己本身包含的监控操作，不能由一些插件来扩展。好在NSClient++已经做的不错了，基本上可以完全满足我们的监控需求。

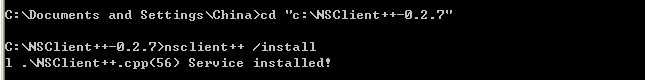
10.1 安装NSClient++

从http://www.nsclient.org/nscp/downloads 下载NSClient++-0.2.7.zip

解压到C盘根目录。

打开cmd 切换到c:\NSClient++-0.2.7

执行nsclient++ /install 进行安装



执行nsclient++ SysTray （注意大小写），这一步是安装系统托盘，时间稍微有点长。

http://images.cnitblog.com/blog/370046/201302/02222739-b0076ad25fbe40d99ff7094f43418657.jpg

在运行里面输入services.msc 打开“服务”



看到下图就说明NSClient服务已经安装上了

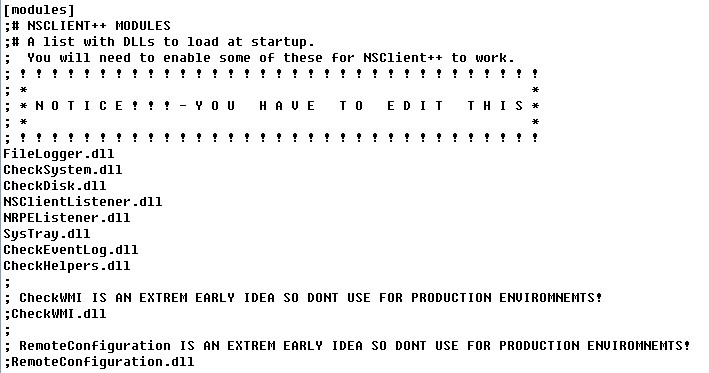
http://images.cnitblog.com/blog/370046/201302/02222842-718e9a7851a348809b4d00244e369544.jpg

双击打开，点“登录”标签，在“允许服务与桌面交互”前打勾。



编辑c:\NSClient++-0.2.7下的NSC.ini文件。

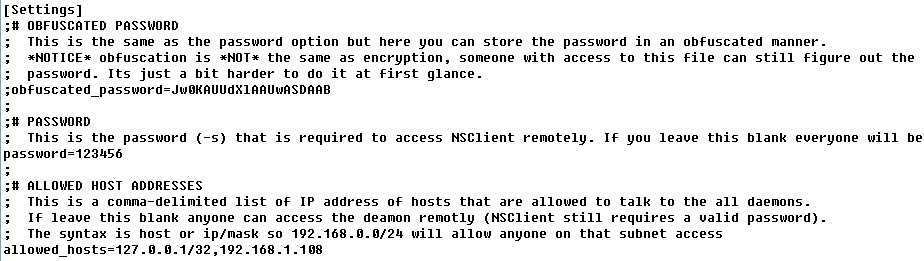
将 [modules]部分的所有模块前面的注释都去掉，除了CheckWMI.dll 和 RemoteConfiguration.dll 这两个。



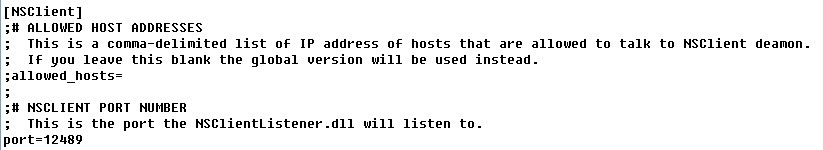
在[Settings]部分设置'password'选项来设置密码，作用是在nagios连接过来时要求提供密码。这一步是可选的，我这里设置为'123456'。

将[Settings]部分'allowed\_hosts'选项的注释去掉，并且加上运行nagios的监控主机的IP。各IP之间以逗号相隔。这个地方是支持子网的，如果写成192.168.1.0/24则表示该子网内的所有机器都可以访问。如果这个地方是空白则表示所有的主机都可以连接上来。

注意是[Settings]部分的，因为[NSClient]部分也有这个选项。



必须保证[NSClient]的'port'选项并没有被注释，并且它的值是'12489'，这是NSClient的默认监听端口。



在cmd 中执行nsclient++ /start启动服务，注意所在目录是c:\NSClient++-0.2.7

http://images.cnitblog.com/blog/370046/201302/02223713-a55666da26bb47338db68d06d04db0ff.jpg

这时在桌面右下角的系统托盘处会出现一个黄色的M字样的图标

http://images.cnitblog.com/blog/370046/201302/02223741-eea5809e0c0a4a75920308e0530a679e.jpg

查看服务

http://images.cnitblog.com/blog/370046/201302/02223800-aa5a5cf0f907417e804e1bf42e3ad95c.jpg

已经正常启动了。

注意服务默认设的是“自动”，也就是说是开机自动启动的。

在cmd 里面执行netstat –an 可以看到已经开始监听tcp的12489端口了。

http://images.cnitblog.com/blog/370046/201302/02223852-f70c360e03204553a28086040839174d.jpg

这样外部就可以访问了吗？

错！

防火墙也要打开tcp的12489端口，否则nagios 检查此服务的时候会报错。

这样被监控机的配置就搞定了，它就等待nagios 发出某个监控请求，然后它执行请求将监控的结果发回到nagios监控主机上。

之前已经在监控主机（Nagios-Server）上对Windows 主机的监控做了配置，但是commands.cfg 中默认没有设置密码项，所以要修改一下，增加"-s 123456"，如下：

# 'check\_nt' command definition

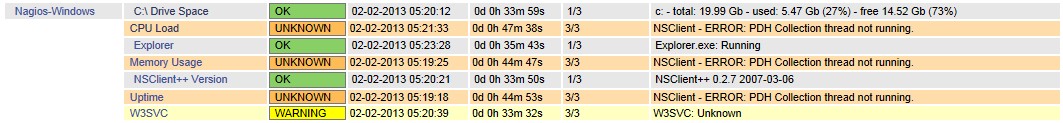
define command{

command\_name check\_nt

command\_line $USER1$/check\_nt -H $HOSTADDRESS$ -p 12489 -s 123456 -v $ARG1$ $ARG2$

}

现在打开Nagios Web监控页便可查看到相关信息了。



可以看到有错误：NSClient - ERROR: PDH Collection thread not running.

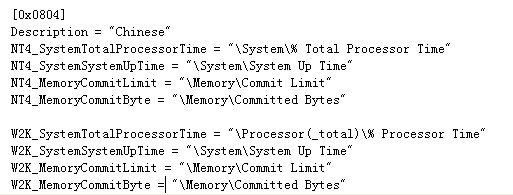
Google 一下，是由于操作系统语言的问题，好像NSClient 默认支持的语言并不多，具体可以百度一下。

查看NSClient的日志C:\NSClient++-0.2.7\nsclient.log，信息如下：

2013-02-02 22:05:30: error:.\PDHCollector.cpp:98: You need to manually configure performance counters!

需要手动配置performance counters。

打开C:\NSClient++-0.2.7\counters.defs文件，复制文件里面"English US"那部分内容，粘贴到counters.defs 文件的最后，修改Description = "Chinese"。



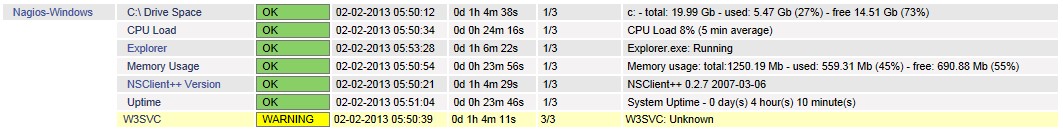
修改完之后，在mmc中重启NSClient 服务。

然后查看日志，内容如下：

http://images.cnitblog.com/blog/370046/201302/03004023-10d2dacf5c6a4d57bbb1c723972a67bd.jpg

在正常执行了。

打开Nagios Web监控页查看。

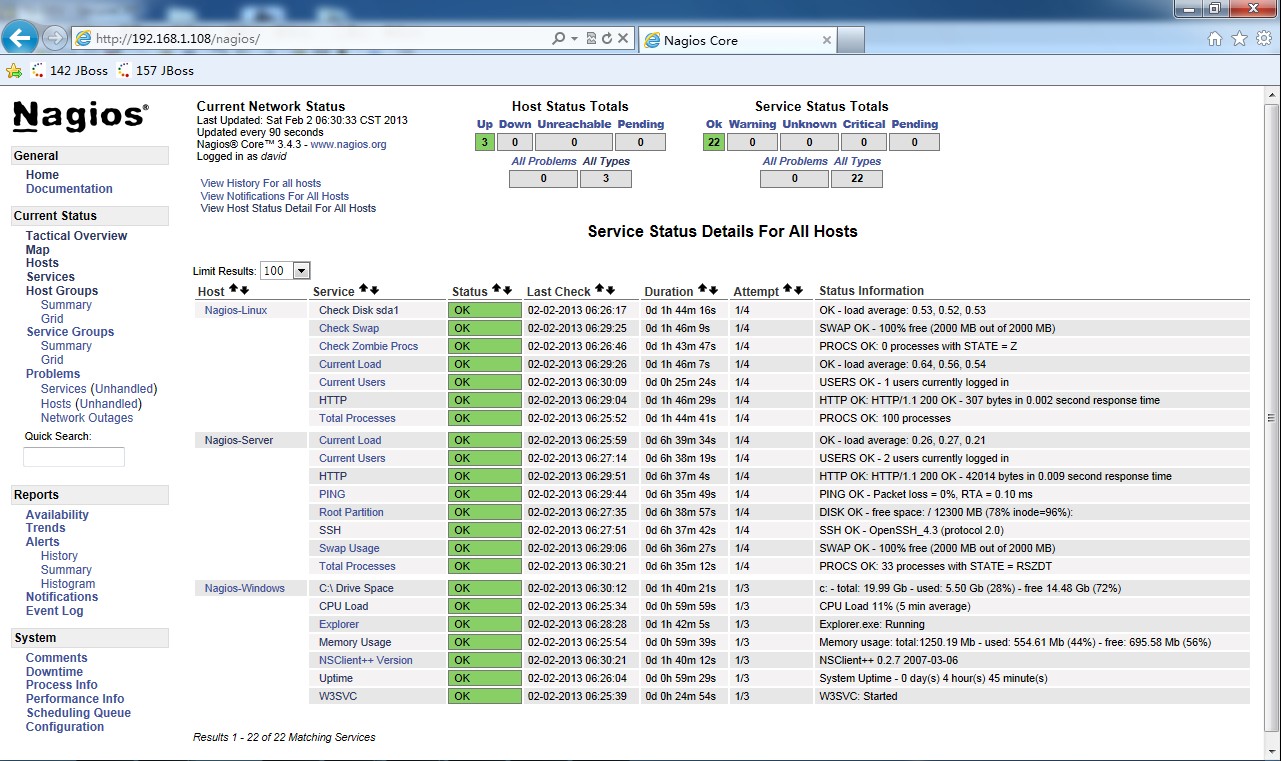


执行成功，但是W3SVC服务为Unknown 状态。查资料，需要开户Windows 的IIS服务。

打开“控制面板”进行安装。



安装完毕后，再到Nagios Web监控页查看，全部监控正常。



十一、Nagios邮件报警的配置

11.1 安装sendmail 组件

首先要确保sendmail 相关组件的完整安装，我们可以使用如下的命令来完成sendmail 的安装：

# yum install -y sendmail\*

然后重新启动sendmail服务：

# service sendmail restart

然后发送测试邮件，验证sendmail的可用性：

# echo "Hello World" | mail david.tang@bsmart.cn

11.2 邮件报警的配置

在上面我们已经简单配置过了/usr/local/nagios/etc/objects/contacts.cfg 文件，Nagios 会将报警邮件发送到配置文件里的E-mail 地址。

11.3 Nagios 通知

PROBLEM



RECOVERY



Linux下Nagios安装配置完毕。

**参考资料**

* Nagios官方网站：[http://www.nagios.org/](http://www.nagios.org/" \t "_blank)
* yahoon的小屋 《nagios全攻略》：http://yahoon.blog.51cto.com/
* 技术成就梦想 《运维监控利器Nagios》：http://ixdba.blog.51cto.com/

## ganglia

# 自动化管理

## Cmdb

## Puppet

参考资料

<https://puppetlabs.com/>

<http://docs.puppetlabs.com/>

<http://dongxicheng.org/cluster-managemant/puppet/>

<http://puppet.wikidot.com/>

Puppet: 强大的中心化配置管理系统

不管是管理1台，或者是上万台机器，puppet都能轻松搞定。让你的所有配置工作整齐划一。所谓配置管理系统，就是管理机器里面诸如文件，用户，进程，软件包这些资源。传统的方法要管理1千台相同的机器，可以选择cfengine，或者是for循环，其实for循环是用的最多的方法。

用puppet，可以运行一个服务器端，然后每个客户端通过ssl证书连接服务器，得到本机器的配置列表，然后更加列表的来完成配置工作。

所以如果硬件配置好了，你在一天之内配置好上万台机器是很轻松的事情。前提是大部分机器配置类似。google用puppet管理了6000多台mac桌面机器。

这个是puppet的官方主页：<http://reductivelabs.com/projects/puppet>

个人用户也很有用，可以把你常用的配置记录成puppet格式的配置文件，下次重装系统，只需要运行这个配置就能配置到你指定的状态。比如我自己的机器，就是所有的软件包和各种各样的配置全都搞好了。

在 CentOS 6.2 上安装 Puppet 配置管理工具

Puppet 是一个客户端/服务器（C/S）架构的配置管理工具，在中央服务器上安装 puppet-server 服务器（puppet master），在需要被管理的目标服务器上安装 puppet 客户端软件（puppet client）。当客户端连接上服务器后，定义在服务器上的配置文件会被编译，然后在客户端上运行。客户端每隔半小时主动会和服务器通信一次，确认配置信息的更新情况，如果有新的配置信息（或者配置有变化），配置文件将会被重新编译并分发到客户端执行。当然，也可以在服务器上主动触发更新指令来强制各客户端进行配置更新。

以下安装采用两台服务器，一台是 master.vpsee.com 用来安装 puppet-server 服务；一台是 client.vpsee.com 用来安装 puppet 客户端。

Puppet 要求所有机器有完整的域名（FQDN），如果没有 DNS 服务器提供域名的话，可以在两台机器上设置主机名（注意要先设置主机名再安装 Puppet，因为安装 Puppet 时会把主机名写入证书，客户端和服务端通信需要这个证书）：

# vi /etc/hosts

192.168.2.10 master master.vpsee.com

192.168.2.11 client client.vpsee.com

Puppet 要求所有机器上的时钟保持同步，所以需要安装和启用 ntp 服务（如果采用 CentOS-6.2-x86\_64-minimal.iso 最小化安装，需要额外安装这个软件包）。

# yum install ntp

# chkconfig ntpd on

# ntpdate pool.ntp.org

29 Feb 15:22:47 ntpdate[15867]: step time server 196.25.1.1 offset 98.750417 sec

# service ntpd start

Starting ntpd: [ OK ]

安装 puppet 服务

Puppet 需要 Ruby 的支持，如果要查看命令行帮助的话需要额外 ruby-rdoc 这个软件包：

# yum install ruby ruby-libs ruby-rdoc

Puppet 不在 CentOS 的基本源中，需要加入 PuppetLabs 提供的官方源：

# yum -y install wget

# wget http://yum.puppetlabs.com/el/6/products/x86\_64/puppetlabs-release-6-1.noarch.rpm

# yum install puppetlabs-release-6-1.noarch.rpm

# yum update

在 master 上安装和启用 puppet 服务：

# yum install puppet-server

# chkconfig puppet on

# service puppetmaster start

Starting puppetmaster: [ OK ]

关闭 iptables：

# /etc/init.d/iptables stop

iptables: Flushing firewall rules: [ OK ]

iptables: Setting chains to policy ACCEPT: filter [ OK ]

iptables: Unloading modules: [ OK ]

安装 puppet 客户端

在 client 上安装 puppet 客户端：

# yum install puppet

Puppet 客户端使用 HTTPS 和服务端（master）通信，为了和服务器端通信必须有合法的 SSL 认证，第一次运行 puppet 客户端的时候会生成一个 SSL 证书并指定发给 Puppet 服务端。

# puppet agent --no-daemonize --onetime --verbose --debug --server=master.vpsee.com

Puppet 服务端接受到客户端的证书后必须签字（sign）才能允许客户端接入，sign 后用 puppet cert list –all 查看会发现 client.vpsee.com 前面多了一个 + 后，表示 “加入” 成功：

# puppet cert list --all

client.vpsee.com (65:3C:20:82:AE:F6:23:A8:0A:0B:09:EF:05:64:1D:BB)

+ master.vpsee.com (AF:A0:32:78:D4:EB:D3:EE:02:1C:62:1C:83:3C:46:EC) (alt names: DNS:master, DNS:master.vpsee.com)

# puppet cert --sign client.vpsee.com

notice: Signed certificate request for client.vpsee.com

notice: Removing file Puppet::SSL::CertificateRequest client.vpsee.com at '/var/lib/puppet/ssl/ca/requests/client.vpsee.com.pem'

# puppet cert list --all

+ client.vpsee.com (65:3C:20:82:AE:F6:23:A8:0A:0B:09:EF:05:64:1D:BB)

+ master.vpsee.com (AF:A0:32:78:D4:EB:D3:EE:02:1C:62:1C:83:3C:46:EC) (alt names: DNS:master, DNS:master.vpsee.com)

这样，客户端和服务端就配置好了，双方可以通信了。

Hello, world

现在可以在服务端写个小例子来测试一下。这个例子作用很简单，用来在客户端的 /tmp 目录下新建一个 helloworld.txt 文件，内容为 hello, world. 在服务端编写代码：

# vi /etc/puppet/manifests/site.pp

node default {

file {

"/tmp/helloworld.txt": content => "hello, world";

}

}

在客户端上执行 puppet，运行成功后会在 /tmp 看到新生成的 helloworld.txt：

$ puppet agent --test --server=master.vpsee.com

warning: peer certificate won't be verified in this SSL session

info: Caching certificate for client.vpsee.com

info: Caching certificate\_revocation\_list for ca

info: Caching catalog for client.vpsee.com

info: Applying configuration version '1330668451'

notice: /Stage[main]//Node[default]/File[/tmp/helloworld.txt]/ensure: defined content as '{md5}e4d7f1b4ed2e42d15898f4b27b019da4'

info: Creating state file /home/vpsee/.puppet/var/state/state.yaml

notice: Finished catalog run in 0.03 seconds

$ cat /tmp/helloworld.txt

hello, world

如何使用 Example42 的 Puppet 模块

现在我们差不多移植了100多台服务器到虚拟机，中间合并了很多功能类似的服务器（历史遗留问题）到同一虚拟机，并且整个配置过程完全用 Puppet 代码化，也就是说有一天我们实验室机房遭遇物理毁灭（刚看完 [Battleship](http://www.imdb.com/title/tt1440129/)，有点激动～），只要再给我们合适的硬件设备（异地有备份数据）我们应该能在6小时内恢复整个基础设施（OS, DNS, DHCP, Provision, Firewall, Squid, Web, Database, LDAP, Email, VPN, Kerberos, NFS, HPC, Cloud, Monitor, Backup, …）。自从 Puppet/Chef 之类的自动化配置工具流行以后，现在甚至有了一个新职位叫做 DevOps.

[安装好 Puppet 后](http://www.vpsee.com/2012/03/install-puppet-on-centos-6-2/)，就可以开始自己写配置代码了，当然也可以在网上找到别人的配置代码直接拿过来用，更妙的是一些通用的 Puppet 配置代码，比如配置一台 Nginx/PHP/MySQL、配置 NFS、配置 DHCP、配置 DNS、配置 OpenNebula 等等已经有人写成模块了，直接拷过来就可以用了。Example42 就是这样的开源 Puppet 模块大集合，包含了众多常用的服务器配置。

首先去 Example42 下载 Puppet 模块代码：

$ git clone http://github.com/example42/puppet-modules.git

看看 Example42 包含了哪些模块吧，几乎常用的都有，不用自己亲自去写配置：

$ cd puppet-modules/

$ ls

DOCS example42 mysql puppi synbak

Example42-tools exim nagios rails sysctl

LICENSE firewall network redis sysklogd

README.rdoc foo nfs repo syslog-ng

Rakefile foreman nginx resolver tftp

activemq git nrpe rootmail timezone

apache haproxy ntp rpmbuild tomcat

apt hardening openldap rsync trac

autofs hosts openntpd rsyslog users

backup iptables openssh samba vagrant

bind jboss openvpn sarg varnish

clamav jenkins oracle selinux virtualbox

cobbler lighttpd pam sendmail vmware

collectd link php snmpd vsftpd

common logrotate phpsyslogng spamassassin wordpress

concat lsb portmap splunk xinetd

controltier mailscanner postfix sqlgrey yum

cron mailx postgresql squid zip

dashboard mcollective powerdns squirrelmail

dhcpd monit psad ssh

dovecot monitor psick ssmtp

drupal munin puppet stdlib42

把上面的 Example42 代码移到 /etc/puppet 目录，然后修改 Puppet 服务器的配置，加上 puppet-modules 模块路径，别忘了重启 puppetmaster 服务：

# mv puppet-modules /etc/puppet

# vi /etc/puppet/puppet.conf

...

[master]

modulepath = /etc/puppet/puppet-modules

# /etc/init.d/puppetmaster

写个配置文件测试一下，自动配置 PHP + Apache + MySQL：

# vi /etc/puppet/manifests/node.pp

node 'web.vpsee.com' {

include apache

include php

include php::pear

include php::apc

php::module { mysql: }

php::module { curl: }

php::module { gd: }

php::module { idn: }

php::module { imagick: }

php::module { imap: }

php::module { mcrypt: }

php::module { ming: }

php::module { ps: }

php::module { pspell: }

php::module { recode: }

php::module { snmp: }

php::module { tidy: }

php::module { xmlrpc: }

php::module { xsl: }

php::module { ldap: }

include mysql

}

在要配置的机器上执行下面一条命令就自动装上 Apache/PHP/MySQL 了，再不用自己手动去安装配置了：

# puppet agent --test --server=master.vpsee.com

## chef

<http://www.opscode.com/chef/>

# SOA架构与实施

## HTTP

## SOA基本概念与术语

服务架构的类型如下：

服务架构（Service Architectures）

服务组合架构（Service Composition Architecture）

服务库存架构（Service Inventory Architecture）

面向服务的企业架构（Service-Oriented Enterprise Architecture）

在Web服务合约内部，服务能力叫做服务操作

服务模型：

任务服务

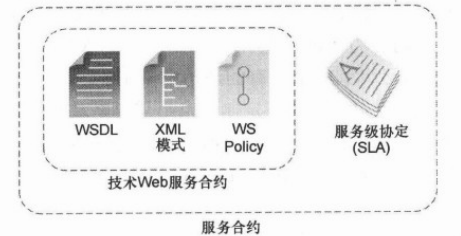
实体服务

工具服务

服务组合

服务库存

服务候选



组成Web服务常见文档

## REST

### JAX-RS规范

JAX-RS: Java API for RESTful Web Services是一个Java编程语言的应用程序接口,支持按照 表象化状态转变 (REST)架构风格创建Web服务Web服务. JAX-RS使用了Java SE 5引入的Java 标注来简化Web服务客户端和服务端的开发和部署。

JAX-RS提供了一些标注将一个资源类，一个POJOJava类，封装为Web资源。标注包括：

（1）、@Path，标注资源类或方法的相对路径

（2）、@GET，@PUT，@POST，@DELETE，标注方法是用的HTTP请求的类型

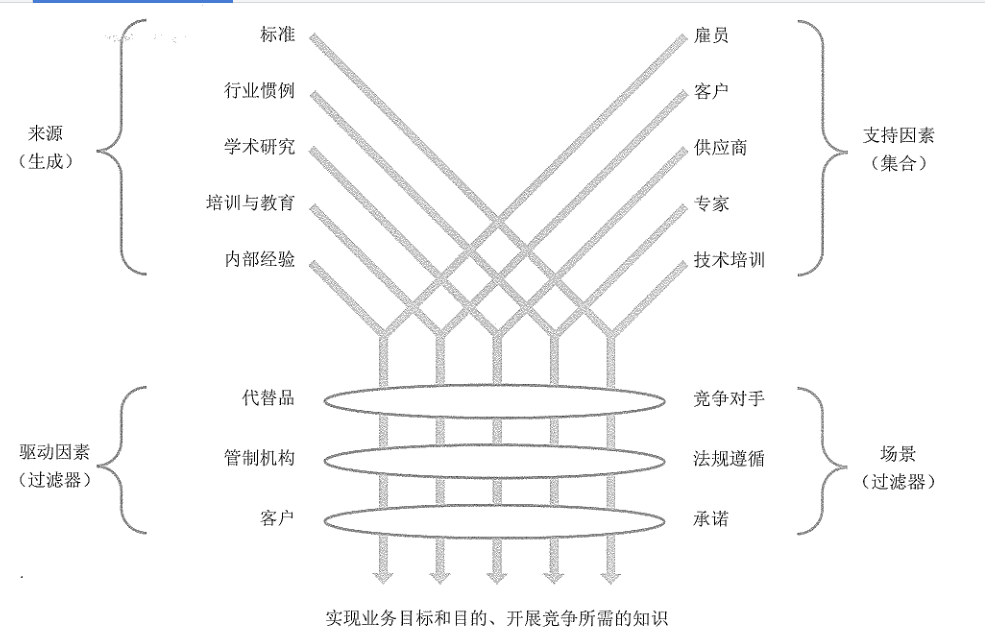
（3）、@Produces，标注返回的MIME媒体类型，（注解标注，这个注解可以包含一组字符串,默认值是\*/\*，它指定REST 服务的响应结果的MIME 类型，例如：application/xml、application/json、image/jpeg 等），你也可以同时返回多种类型，但具体生成结果时使用哪种格式取决于ContentType。CXF 默认返回的是JSON 字符串。

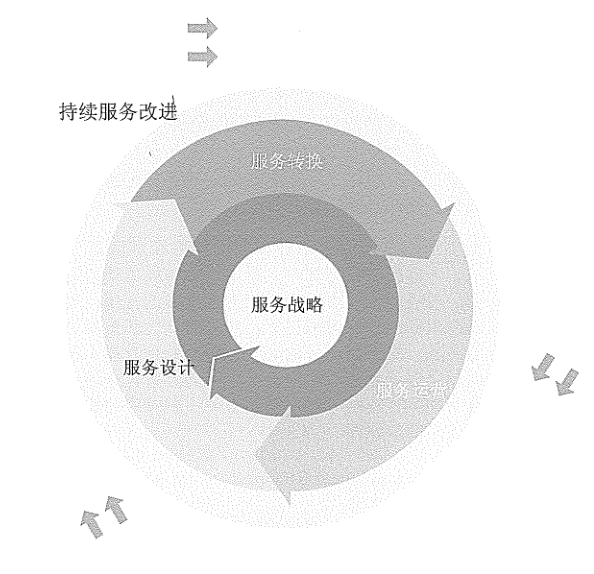
（4）、@PathParam，@QueryParam，@HeaderParam，@CookieParam，@MatrixParam，@FormParam,分别标注方法的参数来自于HTTP请求的不同位置，例如@PathParam来自于URL的路径，@QueryParam来自于URL的查询参数，@HeaderParam来自于HTTP请求的头信息，@CookieParam来自于HTTP请求的Cookie。

## 微服务

### IT基础架构库(Information Technology Infrastructure Library) ITIL

ITIL 服务管理实践构建于一个由各种概念、流程、职能与活动组成的坚实框架基础之上，可带来积极的商业价值回报。这些实践的一个共同特新是以改进成熟度为目标，进而确保服务在各个方面表现出色。





#### 持续服务改进

#### 服务设计

#### 服务运营

#### 服务战略

（1）服务管理的实践

（2）服务战略的原则

（3）服务战略

（4）服务经济学

（5）战略与组织

（6）战略战术与运营

（7）技术与战略

（8）挑战、关键成功因素和风险

#### 服务转换

### 基于JAVA

#### Spring mvc



#### Spring Data JPA入门

##### Spring Data 包含多个子项目

* **Commons - 提供共享的基础框架，适合各个子项目使用，支持跨数据库持久化**
* **JPA - 简化创建 JPA 数据访问层和跨存储的持久层功能**
* Hadoop - 基于 Spring 的 Hadoop 作业配置和一个 POJO 编程模型的 MapReduce 作业
* Key-Value  - 集成了 Redis 和 Riak ，提供多个常用场景下的简单封装
* Document - 集成文档数据库：CouchDB 和 MongoDB 并提供基本的配置映射和资料库支持
* Graph - 集成 Neo4j 提供强大的基于 POJO 的编程模型
* Graph Roo AddOn - Roo support for Neo4j
* JDBC Extensions - 支持 Oracle RAD、高级队列和高级数据类型
* Mapping - 基于 Grails 的提供对象映射框架，支持不同的数据库
* Examples - 示例程序、文档和图数据库
* Guidance - 高级文档

##### Spring Data JPA有什么

主要来看看Spring Data JPA提供的接口，也是Spring Data JPA的核心概念：

1：Repository：最顶层的接口，是一个空的接口，目的是为了统一所有Repository的类型，且能让组件扫描的时候自动识别。

2：CrudRepository ：是Repository的子接口，提供CRUD的功能

3：PagingAndSortingRepository：是CrudRepository的子接口，添加分页和排序的功能

4：JpaRepository：是PagingAndSortingRepository的子接口，增加了一些实用的功能，比如：批量操作等。

5：JpaSpecificationExecutor：用来做负责查询的接口

6：Specification：是Spring Data JPA提供的一个查询规范，要做复杂的查询，只需围绕这个规范来设置查询条件即可

### 基于GO构建

### 基于消息传递

集成解决方案面对的挑战

1. 网络是不靠的
2. 网络速度慢
3. 任何两个应用都是不同的
4. 改变在所难免

开发人员主要通过四种方法客服上述问题

1. 文件传输
2. 共享数据库
3. 远程过程调用
4. 消息传递

什么是消息传递

## UML

### 模型是什么

模型是对现实的简化

模型提供了系统的蓝图，模型可以包括详细计划，也可以包括从很高层次考虑系统的总体计划。

一个好的模型包括有广泛影响的主要元素，忽略那些与给定的抽象水平不相关的次要元素。

每个模型都是一个在语义上闭合的系统抽象。

模型可以是结构性的，强调系统的组织。

模型也可以是行为性的，强调系统的动态。

### 为什么建模

建模是为了能更好的理解正在研发的系统

建模需要达到四个目的：

模型有助于按照实际情况或按照所需要的样式对系统进行可视化

模型给出了指导构造系统的模板

模型对做出的决策进行文档化

因为不能完整地理解一个复杂系统，所以需要对它建模

人对复杂问题的理解能力是有限的，通过模型，缩小研究问题的范围。一次只着重研究他的一个方面。（分而治之）

### 建模原理

选择要创建什么模型，对如何动手解决问题和如何形成解决方案有着意义深远影响。

## 设计模式

**模式的主要三个特征:**

* **上下文是问题存在的相关条件**
* **问题是域中发杂而不确定的主题，它受益于当前上下文**
* **解决方案是解决问题的方法**

并不是某个问题的每个解决方案都可以被称为模式，只有经常出现的问题可复用的解决方案才能被视为模式。

### 创建模式

### 结构模式

解决如何将类或者对象结合在一起形成更大的结构。结构模式描述两种不同的东西，类和类的实例。

类的结构模式：使用继承和接口组合（静态，适配器模式）

对象的结构模式：对象的结构模式描述怎样把各种不同类型的对象组合在一起，实现新的功能的方法。（动态，代理人模式）

#### 适配器

参与角色

目标角色：Target，我期望得到的接口

源角色：Adaptee， 我需要适配的子系统接口、类，或某个核心模块的入口

适配角色：Adapter， 我用来适配当前系统的类

#### 策略模式

#### 职责链模式

享元模式

### 行为模式

状态模式，又称为状态对象模式

一个对象的行为取决于一个或多个动态变化的属性，这样的属性叫做状态，这样的对象叫做有状态（stateful）的对象。

把所研究的对象行为包装在不同的状态对象中

每一个状态对象都属于一个抽象状态类的一个子类

意图是让一个对象在其内部状态改变的时候，其行为也随之而改变。

角色包括

抽象状态角色（State）:定义一个接口，用来封装环境(Context)对象的一个特定的状态所对应的行为。

具体状态角色（AwsState）：每一个具体类都实现了环境的一个状态所对应的行为。

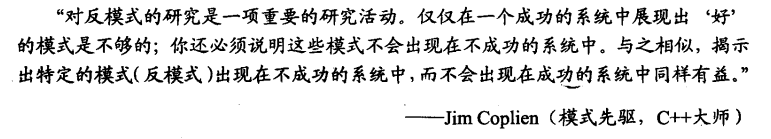
环境角色(Context)：定义客户端所感兴趣的接口，并且保留一个具体状态类型的实例。这个具体类的实例给出此环境对象的现有状态。

## 反模式

反模式是把一般情况映射到一类特定解决方案的有效方法。

1994年 设计模式进入面向对象软件开发社区。

1994年中期 Hillside Group 举办软件设计模式行业会议。



从三个视角讨论反模式

开发性反模式

架构性反模式

管理型反模式

原力：模式语言中提出的指导原则

功能管理，满足需求

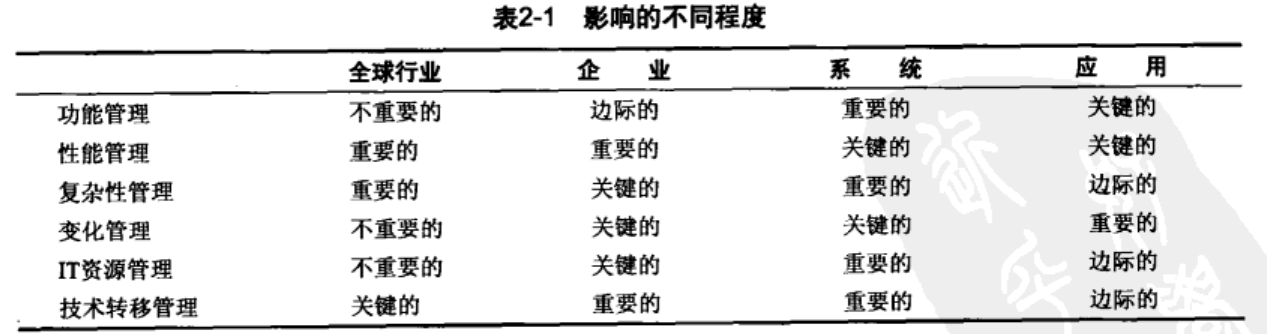
性能管理，满足对操作速度的需求

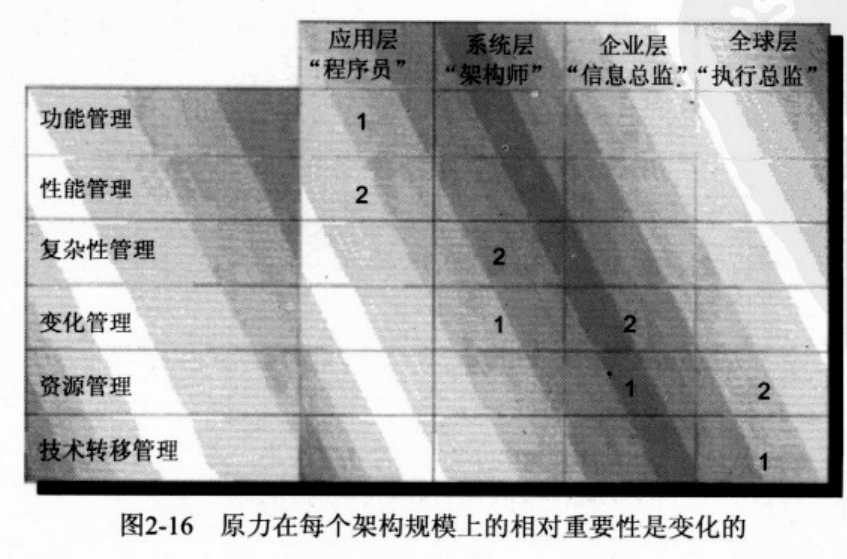
复杂性管理，定义需要的抽象

变化管理，控制软件的发展

IT资源管理，控制人员和IT制品的使用与实现

技术转移管理，控制技术的变化





## 持续交付

如何发布可靠的软件系统？

### 软件交付的问题

### 配置管理

### 持续集成

### 测试策略的实现

### 部署流水线解析

### 构建与部署脚本化

### 提交阶段

### 自动化验收测试

### 非功能需求测试

### 应用程序的部署与发布

# 解决方案

1、广告投放技术

2、广告效果监测

3、数据处理及动态优化

4、网页监测及网络访问行为分析

5、目标受众精细分类

6、智能匹配广告