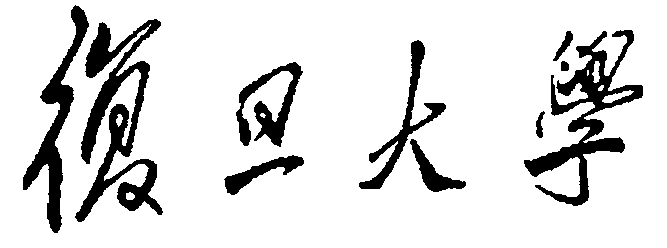
|  |  |
| --- | --- |
|  | 学校代码： 10246 |
|  | 学号： |



|  |
| --- |
| 硕 士 学 位 论 文 |

（专 业 学 位）

**弹性运维系统的设计与实现**

|  |  |
| --- | --- |
| 院 系： | 软件学院 |
| 专 业： | 软件工程 |
| 姓 名： |  |
| 指 导 教 师： |  |
| 完 成 日 期： | 2015年9月18日 |

目 录

[目 录 I](#_Toc432077697)

[摘要 III](#_Toc432077698)

[ABSTRACT IV](#_Toc432077699)

[第一章 绪 论 1](#_Toc432077700)

[1.1 研究的背景与意义 1](#_Toc432077701)

[1.2 本文的主要内容 2](#_Toc432077702)

[1.3 本文的章节安排 2](#_Toc432077703)

[第二章 技术背景 4](#_Toc432077704)

[2.1 统一配置管理系统 CFEngine 4](#_Toc432077706)

[2.2 监控平台Nagios和性能图形系统 6](#_Toc432077707)

[2.3 负载均衡和中央储存及备份 6](#_Toc432077708)

[2.4 Kickstart系统及软件包管理系统 7](#_Toc432077709)

[2.5 DNS、LDAP认证、时间和邮件系统 8](#_Toc432077710)

[第三章 需求分析 12](#_Toc432077711)

[3.1 背景概述 12](#_Toc432077713)

[3.2 系统需求 12](#_Toc432077714)

[第四章 系统架构 14](#_Toc432077715)

[4.1 空闲服务器池 14](#_Toc432077719)

[4.2 将服务器分组 14](#_Toc432077720)

[4.3 远程管理库 15](#_Toc432077721)

[4.4 DMZ设计 16](#_Toc432077725)

[1.2.1 传统DMZ及问题 16](#_Toc432077726)

[1.2.2 修正传统DMZ 18](#_Toc432077727)

[1.2.3 修正策略的实施和效果分析 19](#_Toc432077728)

[4.5 VLAN设计 20](#_Toc432077729)

[4.6 软件架构 22](#_Toc432077730)

[1.4.1 监控系统Nagios 23](#_Toc432077733)

[1.4.2 统一配置管理系统CFengine 26](#_Toc432077734)

[1.4.3 Kickstart/Anaconda系统 27](#_Toc432077735)

[4.7 软件开发模型的选用 28](#_Toc432077736)

[第五章 关键技术分析 29](#_Toc432077737)

[5.1 CFEngine的类设计 29](#_Toc432077739)

[2.1.1 CFEngine内置类及指令 29](#_Toc432077742)

[2.1.2 扩展CFEngine 类 31](#_Toc432077743)

[5.2 CFEngine中实现DMZ 33](#_Toc432077744)

[5.3 CFEngine中实施帐号策略 35](#_Toc432077745)

[2.3.1 统一帐号管理 35](#_Toc432077748)

[2.3.2 传统登录及问题 35](#_Toc432077749)

[2.3.3 改进的登录方法 36](#_Toc432077750)

[2.3.4 Role帐号的管理问题 36](#_Toc432077751)

[2.3.5 帐号及登录映射信息的存放 37](#_Toc432077752)

[5.4 CFEngine中的SELinux策略管理 39](#_Toc432077753)

[2.4.1 SELinux中bool型配置 40](#_Toc432077755)

[2.4.2 SELinux中文件content管理 42](#_Toc432077756)

[第六章 效果分析 43](#_Toc432077757)

[6.1 数据的存储中心化 44](#_Toc432077759)

[6.2 数据流量中心化 44](#_Toc432077760)

[第七章 总结与展望 47](#_Toc432077761)

[参考文献 48](#_Toc432077762)

[致谢 49](#_Toc432077763)

摘要

随着软件产业的发展，尤其是集群技术的大规模运用，系统规模日趋庞大，系统结构日趋复杂，本文深度集成多个开源系统并实现了一套弹性运维平台，以解决多机房情况下，大规模计算环境中面临的如下这些问题：

首先，集群环境下的服务器节点的统一配置管理。当只有一个或者数个服务器的时候，部署人员可以手动的进行一台一台进行部署。这样繁杂的工作对人来说容易出错进而影响业务系统的稳定性，集群环境下，这样的问题更加突出，不同地方、不同机房、不同服务器上面部署的不同应用，这样进一步加大对多数据中心中集群环境下的服务器运维管理难度。

其次，系统规模可以弹性的扩展或者收缩。在web应用为代表的互联网时代，容易出现面对业务量的不可预期增长，一旦过了这些特殊的时间节点业务量又恢复较低的平均水准，这为系统架构提出了更高的要求，处理不慎可能会导致系统整体崩溃，痛失本可创造更多价值的机会。

再者，准确的监控各项服务并且及时将报警信息送达工程师。服务器以及之上的各项服务支撑着机构各项业务的展开，这些业务大多是7x24的服务，而与之对应的，如何跟踪服务器和服务的状态？如何将出错的报警及时准确的送达到相关的部门和人员，是本文解决的另一个问题。

最后，强化安全方面的策略，包括：实现DMZ（Demilitarized Zone）、实现基于role的访问控制以及集群访问内支持SELinux策略。本文设计的安全策略，不仅仅可以解决来自网络、系统多层次的安全威胁，亦可以满足越来越多的上市公司的法规要求，尤其是纳斯达克上市公司必须遵循的《萨班斯&奥克斯利法案》法案（简称SOX法案）第404节中关于主要控制IT控制点，如信息安全、业务的流程、数据的备份及灾难恢复方面的规定。

本文试图从上面四个功能入手，设计并实施一个高可用性的弹性计算环境，该环境中，随着业务量的变化，支撑其业务量的各项应用规模可以敏捷、可靠、安全的水平扩展或者收缩。

**关键词** 数据中心, 弹性运维, DMZ, CFEngine, SELinux

ABSTRACT

With the development of software industry, especially the wide use of cluster technology, the system structure becomes more and more complicated, this paper integrated multiple open source systems to implement an elastic compute platform which can help us to resolve the following problems:

First, the unified configuration management of server nodes in the cluster environment. When there only are a few number of servers, these servers and applications management task can be done manually one by one, but it’s too onerous work for people prone to occur errors, then the scene becomes cluster environment, this problem become more prominent, because the situation becomes more and more complicated in multiple locations, multiple datacenters ad multiple servers.

Second, to boost the ability of flexible expansion and shrink, in the age of internet with the representative of web application, it's very frequent to experience un-expected growing volume, but after these special time window the traffic volume low down to an average level, it puts forward higher requirements for the system architecture, careless handling may lead to system collapse, lost the opportunity to create more value. How to keep the ability of flexible extension or shrink on the server systems side become huge problem that the IT industry have to face.

Moreover, the monitoring and alert against the health of the servers and all kinds of running services which the various business services depend on, the business services are probably 7x24 provides services, and the matching, how to determine whether the 7x24 servers and the services are running well as expected? If an abnormal servers or services occur, how can be detected in the first time, and be able to inform related departments and engineer.

Finally, to enhance the security policy, including to implement the Demilitarized Zone DMZ (Demilitarized Zone), the role based access control policy, and enhance the SELinux policy in the cluster platform. The security system designed in this paper not only help us to handle the security threating came from network or applications level but also can be applicable for a lots of listed companies, especially the Nasdaq listed company which have to abide the 404 section of the Sarbanes and Oxley act (SOX act) which focuses on the main control IT control points.

This article attempts to design and implementation a high availability of elastic computing environment to resolve above four issues.

**keywords** datacenter, elastic compute, DMZ, CFEngine, SELinux

# 绪 论

## 研究的背景与意义

随着IT行业的发展，尤其是互联网行业中服务器规模规模的日渐扩大，服务器出现集群化使用的发展，而与之配套的运维管理模式却呈现不一样的局势。

最初期的时候，系统管理员依靠SSH登录服务器，手工执行命令，进行运维工作，比如新服务器的配置，已有程序的升级，修改刷新各种应用配置等等，这种模式会导致缺乏一致性以及效率低下的问题。

随着服务器数量的增加，这种运维模式很快就达到手工操作的极限了，于是撰写一些自动化的shell脚本以帮助管理员快速的部署成为这个阶段的标志。但是很快这又会带来新的问题，大量的脚本，杂乱无章，脚本间难以互相调用，功能重复，最终留下系统性故障的隐患，进一步的，因为服务器的状态经过不同时期、无数个脚本的改变，完全无法明确现在所处的状态，必然大大降低系统的可靠性和稳定性。

随着虚拟化为代表的云技术发展，以往的单台服务器可以很容易的虚拟出数个甚至数十个虚拟服务器，这样需要管理的服务器数量成倍增加，上面这些运维的难题变得更加突出。

对虚拟机的管理涌现出了OpenStack为代表的管理平台，而虚拟机作为一个逻辑独立的服务器OpenStack却鲜有能力对它进行复杂的维护管理。另外一个后起之秀Docker作为轻量型虚拟化技术的代表，难以支撑很多已有的企业应用，因为这些的企业应用已经面临着CPU、IO或者内存等等计算资源紧张的局面，相比而言这些虚拟化技术并不能完全适用于这样的企业应用场景。

老的运维模式必须更新，而业界的虚拟化技术又不能支撑我们目前已有的应用，为了解决这些问题，本文设计并实现了一套具备弹性的运维管理系统，利用该系统实现了下面这些功能：

首先，集群环境下的服务器节点的统一配置管理。当只有一个或者数个服务器的时候，部署人员可以手动的进行一台一台进行部署：修改系统某项内核参数，修改文件以及目录权限等等工作，而不同应用的服务器又需要不同的这些配置工作，比如数据库服务器的这些配置和web服务器的就不一样。这样繁杂的工作对人来说容易出错进而影响业务系统的稳定性，集群环境下，这样的问题更加突出， 因为面临的服务器数量不是以十几、几十台台，而是以千、万台记，不同应用的服务种类可能高达几百、数千，进一步的随着业务的增长数据中心可能不止在一个城市，可能位于全球多个地方，这样进一步加大对多数据中心中集群环境下的服务器运维管理难度。

其次，在web应用为代表的互联网时代，容易出现面对业务量的不可预期增长，比如国内的双十一，国外圣诞节前后的购物季，比如突然爆发的某个热门话题引发巨量请求，等等都属于这种情况，一旦过了这些特殊的时间节点业务量又恢复较低的平均水准，这为系统架构提出了更高的要求，处理不慎可能会导致系统整体崩溃，痛失本可创造更多价值的机会。如何在服务器系统层面更加容易的水平扩展或者收缩成为企业IT计算环境面临的问题。

再者，服务器和运行于之上各项的服务的健康状况的监控及报警，服务器以及之上的各项服务支撑着机构各项业务的展开，这些业务极有可能是7x24提供服务，而与之对应的，如何确认服务器和服务的状态是否如我们设计和期望的那样7x24的在线持续服务？如果有服务器和服务出现异常，能够第一时间监测到，并且能够通知到相关的部门和人员。

最后，安全是网络时代面临的另外一个严峻的课题，在底层方面，安全主要来自两个层面，一个是网络层面的安全：如何在高效率的前提下对系统实现适度的隔离，进而实现非军事化区DMZ（Demilitarized Zone），同时可以很容易得让某台或者某一组服务器进、出DMZ。另外一个是操作系统层面的安全，堡垒最容易从内部攻破，在组织内部，需要实施严格访问控制，需要实现最小权限策略，即默认情况下，不会有人能够访问任何一台服务器，只能允许特定的人员访问特定的一组服务器和在这组服务器上运行的各项服务，这里的访问包含两个方面即只读的访问和读写的访问，对读写类的访问需要严格审核并且进行记录。这也是作为纳斯达克上市公司必须遵循的《萨班斯&奥克斯利法案》法案（简称SOX法案）第404节中关于主要控制IT控制点，如信息安全、业务的流程、数据的备份及灾难恢复方面的规定。

## 本文的主要内容

本文以配置管理系统CFEngine为核心，以监控平台Nagios、负载均衡器、中央储存为外围，Kickstart、备份系统、性能监控系统、包管理系统、DNS、LDAP、邮件及时间服务等为辅助设计并实现一套弹性运维系统，在这个过程中，主要完成了这些工作：

1. 综合考虑底层的网络规划、访问控制策略、辅助的Nagios等外围系统、设计CFEngine策略模型，使之相互集成。因为需要和Nagios进行数据交互，而Nagios内嵌了大量的Perl库，所以这部分主要集中以Perl语言进行开发。
2. 将面向服务器的手动运维模式转换到以CFEngine为核心的统一配置管理运维模式。将已有的大量的shell、python、perl运维脚本转换成CFEngine代码。因为运维系统的设计和实施与传统的软件项目有很大的区别，传统软件项目有明确的开发、上线这样的生命周期，而运维系统从机构一成立开始就有了，就处于“在线”状态，无法从零开始，必须在兼顾已有的系统的前提下完成上面这些工作。这部分工作集中在将已有的shell、python、perl程序用CFEngine代码来重构。
3. 进一步将CFEngine代码进行重构，以满足自动化扩展收缩系统规模的能力。在对CFEngine进行重构的时候，为了支持系统规模扩大时代码的扩展性，主要使用Python语言进行开发。
4. 重构的同时解决前文提到的监控、安全等任务。

## 本文的章节安排

1. 绪论  
   主要介绍本文的背景和目的，以大规模服务器运维为背景，提出传统运维方式面临的问题。
2. 技术背景  
   介绍本文相关的技术和开源系统，重点介绍本文的核心CFEngine。
3. 需求分析  
   将面临的运维问题综合需求，厘清该系统需要实现的关键任务点。并提出与之对应的解决方案。
4. 系统架构  
   进一步的针对这些需求和方案统筹、细化成该平台的细节设计。
5. 关键技术分析  
   对设计中的关键点CFEngine进行重点分析，并将之用于解决前文提到的几个重点问题。
6. 效果分析  
   对该运维平台的执行效果进行分析。
7. 总结与展望  
   结合行业发展，针对本系统依然存在的不足之处提出完善的方向。

# 技术背景

该系统主要由这样几个子系统组成：核心的统一配置管理系统 CFEngine，它负责整个平台内所有服务器无论是物理服务器还是虚拟服务器的配置管理，只要逻辑上是一个服务器，它的配置都有CFEngine来管理；通用监控平台Nagios，负责监控每个服务器以及服务器上的应用的状态并报警；负载均衡器、中央存储系统、备份系统、辅助的kickstart/Anaconda系统、性能图形系统Munin、软件包跟踪管理系统、以及一些基本的公共系统比如主从结构的DNS系统、主从结构的LDAP认证系统、时间服务系统、邮件系统、中央日志系统等。



## 统一配置管理系统 CFEngine

传统意义上，要上线某个应用，其步骤可能按照是某文档记录的上线这个应用需要做的操作，由操作员按部就班的进行操作。

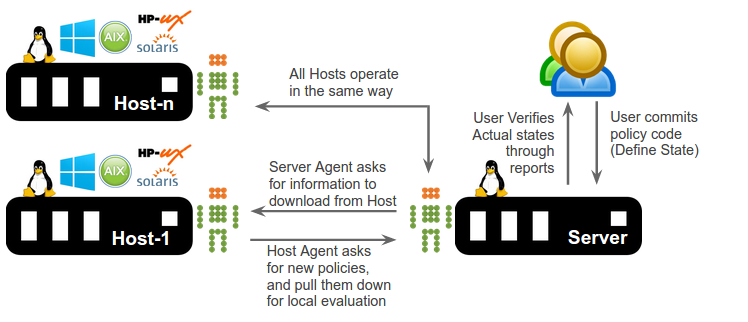
而在弹性环境中，首选是不能再由操作员照着文档按部就班的做，而是需要由系统按需的自动化完成这项工作。

为了让系统的规模可以根据业务量，弹性地扩张扩展收缩，尤其是在扩展也就是新服务上线的时候，需要将上线文档里面的配置任务代码化，比如对服务器如何进行配置、对文件权限做什么改动等等内容代码化，进而把这个配置任务做成自动化，才能完成弹性扩展。这个配置任务代码化、自动化的过程在系统运维中也叫配置管理。

Cfengine（configuration engine配置引擎）是一套使用C开发的跨平台的开源配置管理系统，它即可以运行于嵌入式设备，也可运行于普通服务器或者小型机乃至大型机。通过它可以让这些系统按照既定的策略，使系统从任意的状态向理想状态演变。使系统的状态最终达到运维人员所期望的、所定义的状态。进而可以帮助我们使系统达到相当高度的稳定性、可靠性。

* 系统的稳定性可以看作是自动修正不正确变动的能力。通过预定义的策略规则集来达到这个目的，比如希望指定某个目录有特殊的权限设置，比如web服务器的session目录，比如特定目录必须为指向某个目标的符号链接，等等。如果某个目录权限有变化、特定目录的符号链接被无意或者有意删除，cfengine都会自动侦测到该变化并且按着预定义的策略规则重新设置这些目录和符号链接。除了可以管理目录、管理符号链接，cfengine还可以管理服务器上面任意的对象，比如网络接口、内核参数、服务进程管理、压缩文件等等。
* 系统的可靠性则可视为向理想状态的趋近程度，试想任何一个服务器，从开始有了空白的操作系统到真正可投入生产环境使用的状态，会经历很多中间状态，比如会经历网络接口的设置成功、用户系统初始化完成、软件包的安装和配置成功、应用的部署、重要配置信息的拷贝和修改等等状态直到趋近于最终的理想状态。整个过程系统的状态由不可用（不可靠）的状态慢慢趋近于可用的状态，最终达到高可靠性。传统的配置管理方法中，为了让系统趋近于很高的可靠性，多数情况下是手工进行这些工作，然后手工的进行验证，对于一些重复性高得工作，可能接住脚本完成。但是依然对手动的依赖程度很高。而有了cfengine，所有的这些工作都交给cfengine来完成，运维人员需要做的是在更高层次上设计这些策略，而策略的执行和实施则由cfengine来保证。

如图2‑1所示，Cfengine由下面这些组件构成

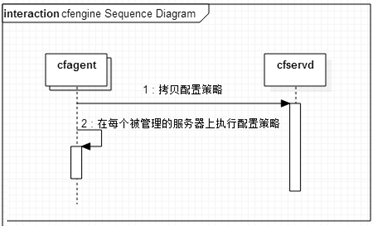


图一九一一年一月二日2‑1 CFEngine架构图

Cfengine主要分为两部分为服务进程和代理进程两部分：

* 服务进程
  + Cfservd，提供网络服务，负责策略文件和数据的分发服务，运行在策略服务器上，是整个cfengine系统的核心。一般运行在Linux服务器上，同时该cfsevd所在的服务器也是通过cfengine来管理的，也会运行下文提到的cfagent代理进程。
  + Cf-monitord, 负责收集系统的状态数据
  + Cf-execd，cfagent的本机调度服务
* 代理进程

Cfagent， 自动配置代理，运行在需要执行策略的服务器也就是被管理的服务器上，该服务器可以是linux服务器，可以是windows服务器，也可以是Solaris等等UNIX甚至是嵌入式设备，它是cfengine系统中非常关键的一个角色。其执行顺序如图2‑2所示它会首先从cfservd服务器上把配置策略拉到被管理的服务器上，然后根据预定义的策略对被管理的服务器进行配置。



图一九一一年一月二日2‑2 cfagent执行顺序图

* Cf-key， 生成CFEngine使用的密钥对，来进行远程认证。
* Cf-run agent,用来在一个或者多个远程服务器上，远程执行cfagent的辅助程序。

## 监控平台Nagios和性能图形系统

Nagios是一套业界广泛使用的源代码开放的监控系统，其核心部分由Perl开发，并且提供良好的脚本API，可以非常容易的与其他语言（比如Python、PHP、甚至Bash）协作，来监控包括应用、服务、操作系统级别的磁盘、CPU、内存以及网络相关的设备，当任何监控项出现问题时可以邮件、短信全天候地向对应技术支持人员发送报警信息，在我们的应用场景中，利用Asterisk项目，通过IP PBX更是实现了电话报警的功能。在监控项目出现问题时，让技术支持人员可以及时的得到通知，以第一时间进行响应。

性能图形系统能够密切监控各项服务的性能参数，比如系统的CPU使用率、系统的Load值、内存使用率、网卡的流量数据、磁盘的IO情况、Web服务的HTTP请求、Web服务的进程/线程数、Web服务的吞吐量、数据库的吞吐量、数据库的query数量以及SlowQuery数量，这些历史数据能够为问题调查、性能调优提供宝贵的第一手数据参考，为此部署了Munin作为图形系统，它是一套开源的图形系统，支持通用的SNMP协议以收集网络设备的性能数据，也支持自定义的扩展，以实现即插即用的插件可以方便的扩展。

## 负载均衡和中央储存及备份

负载均衡器作为高可用系统的核心组件，有两个主要的功能，其一是实现高可用，它可以自动离线意外失效的服务器，有了它的帮助，可以随时下线一台甚至多台失效的服务器而不会对整个在线系统有任何影响；其二是提供负载均衡，可以自动的分配请求给适合的后端服务器，以减轻每个服务器的压力，如果已有的服务器群组依然不能够支持当前的请求，可以很方便的增加更多服务器到该服务器群组中，以提高这个服务器群组的处理能力。负载均衡器可以为整个弹性运维系统提供一个稳定的可用环境，该核心组件的实现有开源的LVS项目，也有诸多的商业产品，比如F5®的BIG IP，Citrix®的Netscaler产品，Juniper®的Redline以及Cisco的LoadDirector/CSM可供选择。

负载均衡器虽然在整个系统结构中负载均衡器的地位非常重要，但是却不是不可替换的，在满足高可用性和负载均衡两个功能的前提下，有过从LoadDirector到Redline再到Netscaler再到F5的迁移经历。

在集群服务器环境下，保持每个服务器节点内容一致，每个节点内容的同步（实时的或者非实时的）将是影响服务器集群服务质量的一个重要指标，比如某个产品更新了新的产品图片，该新产品图片只同步到其中一个或者少数几个服务器节点而未同步至整个服务器集群，将会给用户带来非常糟糕的体验，服务质量会大打折扣。

有实时性要求的场景，譬如上传头像希望马上可以看得到，可以使用商用的中央储存系统比如NAS/SAN，相比而言，NAS比SAN更加便宜而且应用起来更加方便。如果非实时性的内容同步，则可以考虑软件同步来实现，比如rsync、inotify+rsync。

另外一方面使用NAS作为中央储存系统来简化内容同步的问题时，还可以带来额外一个好处那就是备份系统的实施非常简单。

所以本系统中使用的是NAS这样的存储系统，平台内所有的生产数据全都存放在NAS中而不是服务器的本地磁盘里，这样会带来两个好处：其一是数据安全，一般而言服务器磁盘的寿命约三万小时，大约是3.5年，这大大低于数据的生命周期，所以必须将生产数据存放在安全的设备上；

其二是备份容易实施，数据作为企业和机构的核心数字资产，其安全性直接影响到企业和机构的生存，所以备份系统的设计和实施变成为保障公式和机构生存的重要堡垒。如果基于本地磁盘进行备份，在备份作业启动时大量的数据IO会明显降低服务器响应速度进而拖慢服务，而且基于本地磁盘的备份作业管理难度（作业运行时间、运行状态等等）大大增加，所以在系统中备份都是基于中央储存NAS来进行，并且同时支持线备份和离线备份以及异地容灾备份。由于使用NAS中央储存作为数据储存中心，备份系统则只需要使用支持网络数据管理协议（Network Data Management Protocol，简称NDMP协议）的备份方案即可快速实现NAS到磁带的备份和从磁带的数据恢复，此处直接使用的是商业的产品Symantec®的netbackup，再搭配NAS的SnapMirror技术即可以实现机房之间的异地容灾。

## Kickstart系统及软件包管理系统

服务器硬件从供应商交付之后，即开始服务器上线，硬件的上架安装、电源的分配、网络硬件的安装、网络interface/VLAN的分配。到此时依然只是一个只包含硬件甚至连操作系统都没有的裸机，而Kickstart/Anoconda则是完成从硬件裸机到具备基本可用的服务器这一步工作的，Kickstart/Anoconda是RHEL开发的一个用来自动化完成系统安装的系统，它通过自动应答文件ks.cfg来确定一些基本的系统信息并进行安装、配置，比如服务器操作系统版本、机器名名、IP地址、LDAP/PAM认证配置、监控系统注册、图形系统注册以及cfagent环境初始化等。

为了配合Kickstart/Anaconda系统工作，还需要DHCP系统、tftp系统以及每个数据中心都需要的软件包管理系统，进一步的在跨机房的环境中，DHCP协议还需要网络层开启DHCP relay功能以转发DHCP请求到单一的DHCP服务器中去。而在每个数据中心设置软件包管理系统是为了提高Kickstart/Anaconda的速度，尤其是Kickstart/Anaconda时获取PXE镜像的速度，根据测试发现，由于Kickstart所用的tftp协议是UDP的，一个30M左右大小的pxelinux.0文件，从美国的洛杉矶数据中心发送到欧洲的斯德哥尔数据中心摩耗时两个多小时之多。如果不在每个数据中心设立软件包管理系统，如此漫长的Kickstart过程是不能接受的。

任何现代的操作系统都会有软件包管理系统来管理软件包，及补丁系统来追踪最新的系统补丁，以及时更新系统软件包，修复安全漏洞，作为同时使用Red Hat Enterprise Linux®和CentOS两种发行版的企业来说，基于yum软件仓库技术，不仅仅可以及时与Red Hat Network®保持版本更新，更可以建立CentOS的本地rsync镜像，来同时获取Red Hat公司的商业支持和CentOS开源社区的开源力量支持。

同时包管理系统中还包括对应的系统安装文件，以供Kickstart/Anaconda阶段使用，因为每个数据中心都需要设置一个软件包仓库，所以此处直接使用中央存储的snapmirror即实现多数据中心的软件包数据镜像问题，只需要在中心仓库上获取来自Red Hat Network®和CentOS社区的更新，中央储存的snapmirror功能会自动将这些更新镜像到其余所有的数据中心去。

## DNS、LDAP认证、时间和邮件系统

作为基础架构的DNS服务和LDAP认证系统，虽然很重要，但是业界已有非常成熟的解决方案，即使用开源的bind作为DNS服务，开源的openLDAP实施LDAP服务,这些实现都有大量的研究[[[1]](#endnote-1)][[[2]](#endnote-2)][[[3]](#endnote-3)]，可很好地实现跨机房的多点主从结构，以实现在同一个数据中心内部都使用同数据中心的这些基础服务。另外，出于安全考虑对bind的执行使用chroot技术以提高DNS服务器的安全性。

所谓chroot技术是指change root即将bind的执行环境的文件根目录切换到另外一个隔离的目录中去，如图2‑3所示普通的linux文件系统目录结构如图2‑3所示，那么在任意一个执行中的进程来说，访问/etc/passwd目录都是可以访问得到帐号数据，而chroot技术可以将另外一个目录比如/var/jail “伪装”成这个进程的”/”，这样这个进程调用open(“/etc/passwd”, O\_RDONLY)时打开的实际上却是/var/jail/etc/passwd文件，这样就算因为某些原因这个服务进程被破环而导致的文件系统破坏范围会被隔离在/var/jail这个目录底下，而不会影响到真正的文件系统。



图一九一一年一月二日2‑3 Linux文件系统目录结构

另外，因为普通的bind将zone数据存放在格式比较自由的文本文件中，这对系统自动化增删改DNS记录尤其不便，所以在系统中增加了Bind DLZ支持，将bind的DNS zone file数据存储到数据库中，这样不仅仅解决了程序化增删改DNS记录的问题，而且把DNS的主从复制问题转换为数据库的主从复制这一具备通用解决方案的问题。

另外使用ntp.org的开源ntp[[[4]](#endnote-4)]软件来实现时间服务，以确保这个计算平台内所有机器的时钟保持一致，进一步的也与平台外部时钟保持一直，因为系统中有很多计算设计到时序先后的问题，如果是只有一台机器参与计算，那么无论怎么分配请求，其处理顺序一定是先处理的请求其时间戳一定早于后处理请求的时间戳，但是在集群环境下，如果集群内部机器间时钟不一致的话，有可能出现后处理请求的时间戳却早于先处理的请求这样一种混乱局面，所以我们的原则是优先保障计算平台内所有机器的时钟同步，然后进一步的，与north-america.pool.ntp.org的时钟保持同步，north-america.pool.ntp.org是pool.ntp.org北美成员，为北美提供公共ntp服务的地址，这样一来就完成了平台内、外时钟的一致同步。

邮件系统，在集群环境中的邮件系统不同于普通办公环境使用的邮件系统，也不同于单机环境下的邮件转发系统，首先是邮件类型繁多，数量庞大，为此在每个数据中心都专门架设邮件系统集群来处理邮件，而对于需要大量发送邮件的应用比如发送用户订阅信息的订阅系统，则另设专门的邮件系统来处理，这都是为了解决邮件发送方面的负载，另一方面，由于互联网垃圾邮件和恶意软件泛滥的缘故，邮件系统能否成功的将邮件发送出去，还取决于互联网上对本邮件系统信誉的认可度，信誉认可度很高，那么投递成功率就必然很高，反之，如果被列入一些知名的黑名单譬如spamhaus.org, spamcop.net, abuseat.org的话，那么互联网上绝大多数邮件系统都会拒收该邮件系统发出的任何邮件，因为这些非营利机构都有提供Realtime Blackhole List(RBL)功能，一旦列入黑名单，即刻在全球范围内生效。

譬如，在9月18号，图2‑4中所示的几个IP就被列入国际反垃圾邮件组织Spamhaus的黑名单，这些IP就再也无法向全球发送邮件直到改善并申请完成解封。



图一九一一年一月二日2‑4 Spamhaus黑名单

更有甚者图2‑5这个中国山东电信的大段IP都被全球邮件服务器屏蔽[[[5]](#endnote-5)]，



图一九一一年一月二日2‑5 更大范围的黑名单

更加糟糕的是这个网段还被列入了危害等级更高的Don't Route Or Peer(DROP)列表，这个DROP列表一般是因为不仅仅垃圾邮件泛滥而且有恶意软件施虐区，这个结果往往是被劫持或者被专业的垃圾发送者及网络犯罪者所使用，被列入了DROP列表之后意味着，全球网络中，国际反垃圾邮件组织建议不要与之进行通讯联络。另外，因为Email协议本身的缺陷，假冒身份发送邮件成为可能，鉴于此，为了防止别人冒用本机构的名义发送假邮件，也实现了Sender Policy Framework（简称SPF）来保护我们自己不被别人冒充，所谓SPF是一套基于DNS的邮件发送者安全策略，该策略由开放组织OpenSPF于2003年发布，2004年提交到RFC4408，并于2005年6月纳入IETF，简言之，利用DNS中的SPF来指定哪些邮件服务器才是本机构合法的邮件服务器可以使用本机构的域名发送邮件，知名邮件安全组织Email Security Grader的评级报告中，评分高达93的网易[[[6]](#endnote-6)]，其SPF记录如表2-1所示：

表一九一一年一月二日2-1 网易SPF设置

|  |
| --- |
| "v=spf1 ip4:220.181.12.0/22 ip4:220.181.31.0/24 ip4:123.125.50.0/24 ip4:220.181.72.0/24 ip4:123.58.178.0/24 ip4:123.58.177.0/24 ip4:113.108.225.0/24 ip4:218.107.63.0/24 ip4:123.58.189.128/25 -all"  "v=spf1 ip4:176.34.21.58 ip4:176.34.53.178 ip4:121.195.178.48/28 ip4:223.252.213.0/24 -all"  "v=spf1 ip4:223.252.206.0/24 ip4:43.230.90.0/27 -all"  "v=spf1 ip4:123.126.65.0/24 ip4:220.181.97.0/24 ip4:180.150.142.123 ip4:180.150.142.124 ip4:180.150.154.88 ip4:180.150.154.92 ip4:180.150.154.93 ip4:103.251.128.69 -all" |

这些策略是由专门的SPF语法构成[[[7]](#endnote-7)]，它向整个因特网宣布只有在此列表中的IP才是163.com合法的邮件服务器，才能以163.com的名义向外发送邮件，这样通过SPF有效的阻止了对163.com这个邮件地址的假冒使用。

# 需求分析



## 背景概述

服务器运维的工作，一般情况下，是业务发展需要服务器了，便根据需要向服务器商下采购订单，然后等待到货直至服务器到达机房，开始服务器上架（配置电源和网络），安装操作系统，部署应用，测试（测试人员测试新的服务器节点，确认各项应用工作正常之后），实施监控，最后正式上线开始提供服务。

这样一系列的过程中，向服务器商下订单这个环节最费时，算上企业内部财务流程，一台普通的机架式服务器从开始定计划准备采购到收货，常常耗费超过半个月时间。这在弹性环境下按需立马上线新服务器是不可接受的，这是需要考虑的第一个非技术层面的问题。

同时，如何快速准确地根据在线系统负载或者在线系统失效时，动态地、弹性地部署新服务器，部署过程中，完成上面所有这些步骤是该系统需要解决的问题。

另一方面，最复杂、最容易出错的环节是部署应用，不同的应用需要不同的系统环境、不同的软件版本；甚至相同的软件版本时，不同的应用又需要不同的参数。

进一步的，在将应用部署上线之后又面临着需要统一协调监控，以防发生错误时能够及时、准确的向响应的技术人员发送报警。

## 系统需求

面临前面提到的这些已有的问题，细化成弹性运维系统需要实现的功能细节：

1. 将服务器购买到物理上架时间缩短到小时级别，以满足弹性系统可以随时部署新服务器的硬件需求。
2. 可以让服务器裸机接收来自系统的控制命令，以开关机并自动开始操作系统的安装和基础环境的初始化。
3. 让装好操作系统具备基本环境的服务器节点自动开始进入对应服务组相关的应用部署环节，应用的代码复制或者内容同步，相关的目录/文件权限等等全部自动完成，并且开始提供服务。
4. 在监控系统中注册刚刚上线的新节点，并且自动设置好监控该服务器节点上所运行的服务，比如数据库服务器节点需要设置数据库方面的监控，而web服务器节点则需要设置web服务器方面的URL监控，Solr服务器节点则需要设置搜索相关的监控。
5. 要求所有已上线服务器接受在性能图形系统的性能监控，并且自动发现相关的服务，比如web服务器组的服务器需要收集web相关的性能数据，而数据库服务器组的服务器则需要收集数据库相关的性能数据。
6. 需要实现一些基础的公共服务，比如DNS、LDAP认证、邮件分发服务、时间服务等等，这些服务为基础服务，如果出现宕机，会影响这个数据中心，所以在单个数据中心中这些服务必须有冗余，至少部署双服务节点。
7. 满足较高级别的安全要求，由于该系统为纳斯达克上市公司所属，所以必须遵循《萨班斯&奥克斯利法案》法案的规定，在安全方面要满足特殊要求
   1. 网络层面必须支持DMZ。
   2. 每个服务器节点必须的系统安全级别必须高于美国国防部颁布的《可信计算机系统评估系统Trusted Computer System Evaluation Criteria》（简称TcsEC）中的C2级别。
   3. 所有服务器节点上的服务进程必须运行在特殊的role帐号之上，这些role帐号的访问也受到严格的记录和审查。
   4. 针对容易出现安全威胁的服务比如DNS有专门的安全加固策略。
   5. 对于重要的基础平台数据的同步都需要使用加密的安全连接或者使用密钥方式进行认证。
8. 进一步的，针对上面这些需求，要在全球多个数据中心中实现。
9. 另外因为在全面部署这套系统之前，已有一套老的运维系统在线运行并且提供服务，虽然这套老的系统很落后，但是不能在很短的时间段内完成这个系统的替换，换句话说，在相当长的时间段内，需要新旧两套运维系统同时在线运行。

# 系统架构



对服务器进行断电、上电、甚至安装操作系统等操作可以看作是系统运维涉的开端，这些操作必须要求是可以远程实施的，而且很多时候服务器容易挂起，导致操作系统没有任何响应，所以这些断电上电操作不能依赖操作系统进行，而必须在独立于操作系统之外进行，这需要有特殊的远程管理硬件支持以及一个统一的远程管理库作为运维系统的接口来调用。

另一方面，一个大的运维平台中，必然会涉及到许许多多的网络，而网络方面又处于整个系统自上而下这个体系中最底层的基础，所以在开始设计软件层面的架构之前，先对网络环境进行了一些规划，这些规划都是为了满足运维系统上层需求尤其是DMZ而做出的。

运维系统中的软件部分，根据设计，则主要由CFEngine配置管理系统和包括Nagios、Kickstart等在内的一些辅助功能子系统来构成。



## 空闲服务器池

用软件项目管理的眼观来看待弹性计算系统中节点部署的问题，则向服务器商商下订单购买服务器上架及部署这三步，无疑是关键路径上耗时最长的步骤。

经过综合考虑，决定使用成本换时间的策略来解决这个问题，即构建空闲服务器池，该空闲服务器池是已经完成上架步骤，分配好电源和网络，开机即可安装操作系统的裸机。该服务器池中服务器的数量在不同的阶段可能不一样，业务预增期可能十台甚至数十台，业务预降期可能保持在五台到十台的水平。不能让空闲服务器过多，原因是这些空闲服务器本身需要成本，而且它们依然要上架，需要消耗机柜资源，从成本考虑尽量保持在合理的范围内。

同时定时按周或者按月检查空闲服务器池的分配情况，如果不足则立即采购补充。这样便可大大压缩购买环节所耗时间。

这样一来就将以往的准备服务器硬件周期由数十天缩短到0，随需随取，随取随用。极大的满足弹性运维系统中对服务器硬件的需求。

## 将服务器分组

由于弹性运维系统中管理的服务器数量庞大，而且应用种类繁多，传统上，我们习惯性的将服务器分组，为同一种类型的服务分配N台服务器，从失效备用考虑，此处的N一般大于等于2。但将一些负载非常低的服务器兼作它用的情况也常常发生。

为了简化运维管理，我们沿用并强化了这种传统做法，服务器按应用分组，并且最起码使用N+X（N≥1,X一般为1）策略来部署每组服务器节点数量，其中的N为正常提供服务的服务器数量，而X则为在线的已完成所有配置和应用部署的冗余服务器，它随时可以投入生产。这样每组服务器最少使用两个成员（N=1，X=1的情况下），任何一个生产环境的服务器节点失效后，它对应的冗余服务器X可以马上上线投入使用，不会对在线服务有任何影响；如果关键应用或者负载偏高的应用甚至使用X=2甚至X=3的方式来部署。

通过这种方式可以快速为某个服务器组添加或者删除成员，便于我们后期的弹性策略的实现。

分组之后，服务器成员最少设为2（N=1，X=1的情况下），这时候，会面临数据同步的问题，比如在成员服务器A上面修改的文件必须在在适当的周期内也反应到成员服务器B上面，为了简化数据同步的问题，我们利用中央储存，将同组服务器的应用相关数据都部署在同一个中央储存的相同卷上来实现数据同步，这样A的修改可以及时的出现在成员服务器B上面。

但是我们放弃了将低负载服务器兼作它用的做法，因为这么做会极大得模糊组的界限，增加管理的难度。取而代之的是将一些常见的低负载应用集中部署在一组机器上面，以避免服务器硬件资源的浪费。

## DMZ设计

一般情况下的网络，只需要考虑子网划分、VLAN设置以及路由这些问题。但本系统需要支持DMZ，这成为设计和实施的重点,所以先设计DMZ，然后规划VLAN。

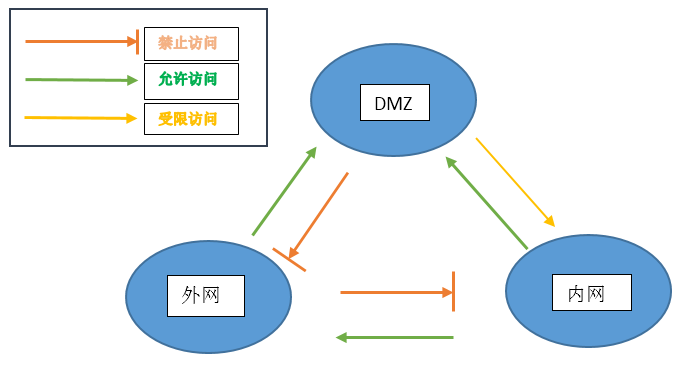


### 传统DMZ及问题

传统意义上 DMZ网络中，有三个类型的角色：外网、DMZ和内网，这三者两两组合构成六种访问关系：即外网与DMZ的相互访问、外网与内网的相互访问以及DMZ与内网的相互访问。传统DMZ情况下，与之对应的访问策略是：

1. 内网可以访问外网
2. 内网可以访问DMZ
3. 外网不能访问内网
4. 外网可以访问DMZ
5. DMZ访问内网有限制
6. DMZ不能访问外网

即图4‑1所示

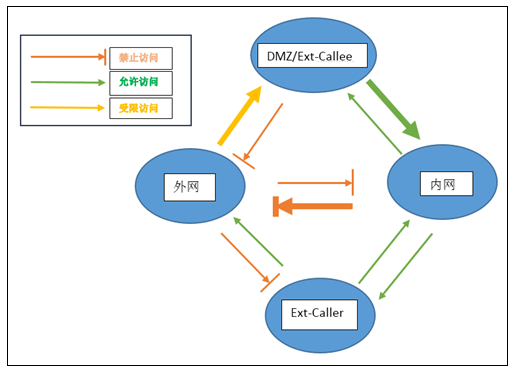


图一九一一年一月四日4‑1传统DMZ访问策略

在这种策略中，有这样一些问题：

1. 首先是外网可以自由访问DMZ，虽然DMZ是属于容许受破坏的环境，但是允许自由的对其进行网络访问对于企业来说依然是不可接受的。
2. 外网只能访问DMZ中的服务器，而DMZ对内网的访问有是受限访问。这在互联网行业是有矛盾的，作为重要的企业数据，大多属于乐于公开型的数据，换句话说，需要将很大部分内网的数据，尽可能多的提供给外部访问。当大部分内外数据都放在DMZ中的时候，DMZ已经不再是DMZ了，这其实大大模糊了DMZ的界限。
3. 内网可以自由的访问外部网络，这极有可能导致出现“提权”攻击即最开始只是小权限漏洞通过对外部资源的利用导致二次提权从而取得最高级别访问的攻击，从而导致发生“堡垒从内部攻破”的困境。
4. 服务器分配入DMZ和退出DMZ过程繁琐，因为这种情况下的DMZ其实是直接通过VLAN来实施的，换句话说如果需要进入DMZ，必须重新分配IP。如果要退出DMZ，也重新分配IP。对于大多数系统来说重新分配IP，比如会导致DNS变化、服务的使用方需要更改配置等等连锁的反应，重新牵一发而动全身的情况。

### 修正传统DMZ

为了上面这些问题，我们对传统的DMZ做出修正，修正情况如图4‑2所示，

图一九一一年一月四日4‑2修正之后的DMZ访问策略

1. 首先，将传统DMZ允许内网访问外网的策略修正为禁止外网和内网的网络连接，即无论外网发起还是内网发起的网络连接，都被禁止,这在极大的程度上提高系统的安全防范能力，因为谁都无法攻破一个根本不能连接的服务器。
2. 增加一个允许自由访问外网的Ext-Caller角色，它可以自由访问外网，但是外网是不能够访问它的，增加这个角色只要是解决禁止内外网连接后，造成无法调用第三方合作伙伴服务的情况，比如调用weather.com.cn的天气服务获取天气信息，或者调用<http://finance.yahoo.com/>的API获取股价走势信息等等。
3. 允许内网和DMZ自由双向通讯，这么做其实是将DMZ和内网数据共享的界线打破，这对于互联网行业的实际情况是相符的，但我们也认为这仅仅适用于互联网这样的外部共享型环境。
4. 外网到DMZ的访问由自由访问变为受限访问，只允许访问特定服务的端口，比如SMTP只允许110，ftp只允许控制链接的21端口和数据连接的20端口，这样DMZ中的服务器相当于一个被动等待外部请求链接的角色，我们将之命名为Ext-Callee。

简单说来，默认时服务器就是内网服务器，与外网完全隔离。只有下面两种情况发生时角色会发生转变：

* 1. 当需要对外提供服务时，只开放固定服务的端口，比如如果是web服务器只开发80端口给外部。这种角色称之为Ext-Callee。
  2. 需要访问外部服务时允许单向的外部访问。这种情况称之为Ext-Caller。

有的服务器节点可能同时具备Ext-Caller和Ext-Callee双重角色，这个时候，这种节点可以自由访问外网，但是外网对这些节点的访问是受限的。

### 修正策略的实施和效果分析

这样上面的策略按照下列方法来实施：

* 默认时所有服务器节点都位于内网中，只配备内网IP地址，而不配备外部公网IP地址。
* 只添加IP v4三个保留专用地址段（即A类地址的10.0.0.0/8,B类地址的 172.16.0.0/12 和C类地址的192.168.0.0/16）的路由，而不添加其他路由，特别的不添加默认路由，这样默认时系统是无法访问外网的。

这样一来，可以实现内网和外网的完全隔离，内外网互相无法直接通讯。

这个时候，我们必须知道当前服务器所使用的网关地址，因为服务器成千上百规模的系统中，为了避免网络广播风暴，我们根据业务性质来划分VLAN以隔离广播，而不同VLAN的服务器之间要通讯的话，必须首先将数据包通过本VLAN的网关IP进行转发，而不同VLAN的网关IP是不一样的，有多少VLAN就需要多少网关，成千上百个VLAN则有成千上百个网关。我们不能简单的认为每个VLAN只能有10个或者1000个网络接口，10个对于很多类型的业务太少，而1000个则过于浪费VLAN资源，导致IP地址严重浪费。如何按需的设置全球VLAN地址库并且将VLAN的网关信息交换至运维系统，成为VLAN规划需要解决的问题，这个问题留在4.5节解决，简化起见，本节先假设已经获取到本VLAN的网关地址。

而对于DMZ/Ext-Callee和Ext-Caller的网络连接策略通过下面两条来实施：

* DMZ/Ext-Callee角色需要外网连接接入时，通过网络地址转换（Network Address Translation，简称NAT）来实现。
* 当Ext-Caller需要访问外网时，为该Ext-Caller节点添加默认路由至本LVAN的默认网关地址。

对Ext-Callee的实现相对比较容易，网络层面做NAT是很容易的，而且互联网企业而言大多数曾经是开发80端口（http服务）和443（https服务）的NAT，尤其是几乎所有的http/https服务都是通过负载均衡器实现的NAT，即公网IP绑定在负载均衡器上而有负载均衡器通过80或者443端口对内部服务器进行NAT，情况就变得更加容易。

而实现Ext-Caller的时候，面临一个问题是应该在网络设备实施还是在服务器系统层面实施。虽然在网络防火墙中也可以通过白名单的方式允许服务器访问外网，但是这需要运维系统修改边界防火墙，因为边界防火墙一般处于整个机房内外网边界处，这种全局的修改如果出现bug会导致整个机房的内外访问都出现故障的严重后果，经过权衡，最终我们选择使用在在服务器网络层面修改默认路由的方式实现，这样即便出现bug，受影响的服务器只是当前服务器而不是整个机房这么大范围。这样由统一配置系统CFEngine即可在系统层面控制让一个内网服务器节点到Ext-Callee/Caller的角色转换，具体CFEngine设置细节代码参考5.2节。

改进之后的DMZ策略具备这样几个优势：

1. 内网安全性得到增强，因为除了极少数的Ext-Caller外可以单向的自由访问外网外，内外网彻底隔离，彻底杜绝“堡垒从内部攻破”的情况 。
2. DMZ和内网的通讯更加自由，更加适应对外发布的网站数据就是企业最重要的数据资产的互联网企业。避免了原来访问模型中DMZ无法自由访问内网导致的诸多问题。
3. 服务器进出DMZ非常方便快捷，不需要对服务器的IP做任何改动，更加不会引起服务调用方的连锁反应。对于Ext-Caller只需要简单的删除或者设置默认路由即可，而对应Ext-Callee只需要设置NAT即可。

## VLAN设计

为了避免广播风暴，我们也如业界同行一样，对网络按照业务类型划分VLAN，以提高网络通讯效率，于是在全球多机房的内网环境中，划分出来的VLAN数量有成百上千个，面对如此数量庞大的VLAN，为了实现DMZ，要将所有这些VLAN信息提交给CFEngine，尤其是非常重要的本机的子网VLAN和其正确的网关信息成为需要解决的另外一个网络问题。

作为企业内部IP地址管理，成千上万的设备需要使用IP地址，会同时使用IP V4中三个保留的专用地址段：A类地址的10.0.0.0/8,B类地址的 172.16.0.0/12 和C类地址的192.168.0.0/16。并对这三个地址段进行进一步的子网VLAN规划（此处为了简化网络层和系统层的接口协议，不特别区分VLAN和子网的差别，将VLAN和子网一一对应），比如A类地址段中的10.40.1.0/24, 10.40.2.0/24, 10.40.3.0/24, 10.40.4.0/24这四个VLAN划分给中国上海机房，而172.16.0.0/12再次划分多个子网VLAN，分配个数个欧洲机房，与之类似的其余10.0.0.0/8等等则可能划分给北美数个机房。

在上海机房的这四个VLAN是在核心交换机上进行配置，而接入交换机则使用VLAN中继协议（VLAN Trunking Protocol，简称VTP协议）从VTP Server处更新本域内所有的VLAN信息。并且有如下约定：

表一九一一年一月四日4-1 VLAN约定规则

|  |
| --- |
| VLAN中，第一个可以地址为该VLAN的网关，并且网络层使用热备份路由协议（Hot Standby Router Protocol,简称HSRP）以保证网关可用 |

表一九一一年一月四日4-2 VLAN规则实例

|  |
| --- |
| ; 10.40.2.0/24 -- vlan202  ;  0 PTR net.vlan202.sw-core1.sh1.foo.net.  1 PTR hsrp.vlan202.sw-core1.sh1.foo.net.  2 PTR vlan202.sw-core1.sh1.foo.net.  3 PTR vlan202.sw-core2.sh1.foo.net. |

*表4-2*中，以10.40.2.0/24这个网段为例，10.40.2.0为该子网VLAN的网络地址，10.40.2.255为其广播地址，按照表4-1的约定，而该子网VLAN中，第一个可用的地址10.40.2.1即为该子网VLAN的网关地址。

根据这个规则，使用python，开发了一个基于SNMP的库，这个库从各个数据中心的核心交换机上面去获取整个平台的VLAN信息，并计算每个VLAN的网关地址，再将这个信息导出给CFEngine，以供CFEngine进一步配置每个VLAN中的各个主机。

同时每个数据中心的核心交换机上启用简单网络管理协议（simple network management protocol, 简称SNMP），这样在配置管理系统中利用SNMP来向每个数据中心的核心交换机获取所有的子网VLAN信息，并按照的约定即可计算出该子网VLAN正确的默认网关，并添加默认路由，进而完成将该节点从内部节点向Ext-Callee/Caller的转变工作。

这样我们既解决了VLAN的规划问题，又解决了网络层与CFEngine共享VLAN及VLAN网关信息的问题。

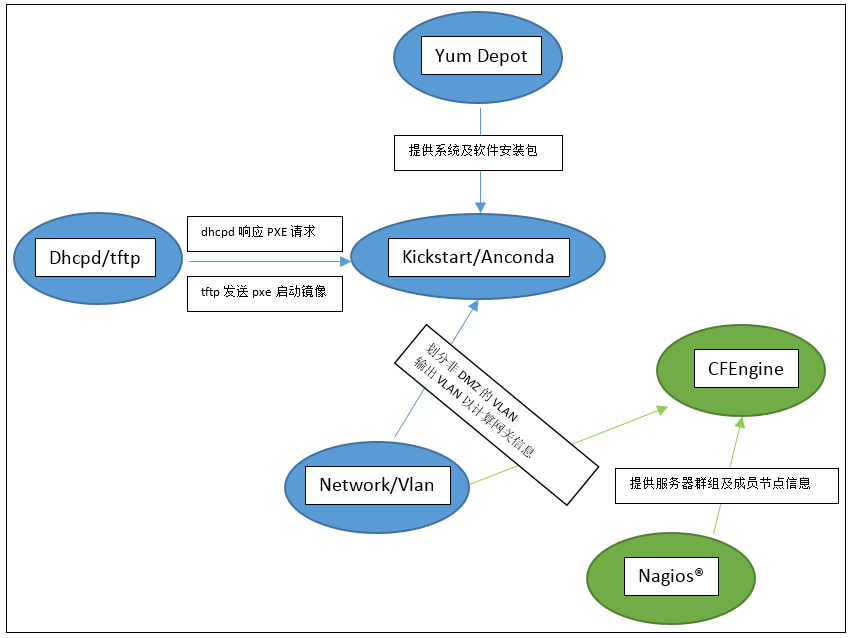
## 软件架构

在基于服务器组进行管理进而实现一个弹性计算环境的前提下，如何让各个子系统围绕着服务器组来设计和实现，成为软件架构部分最关键的部分。

我们从服务器的生命周期来理解整个各个子系统所起到的作用，进而帮助我们理解这个运维系统软件架构的设计全貌。

服务器从裸机上电开始，便开始接受运维系统的管理，此时的服务器只是一个通了电源却连操作系统都没有的裸机，给裸机开始装机模式开始安装操作系统是运维系统的第一件任务，让裸机4.1节中的远程管理库可以进入PXE环境的装机模式，PXE历史上被用作无盘工作站，但是我们的系统中是为了让裸机接受PXE启动镜像并进入Kickstart系统以开启无人应答的装机模式，在Kickstart的装机模式中，会根据我们的要求，设置好服务器名、服务器组以及IP、内网默认路由、DNS等等基础信息，让服务器处于一个基本可用的状态，然后通过调用CFEngine由CFEngine将之配置成一个真正可以上线提供服务的服务器。

整个运维系统的结构如图4‑3所示



图一九一一年一月四日4‑3运维系统概图

在CFEngine配置的服务器的时候，从全局来看，所有服务器节点是按照服务器群组来进行配置的，即功能相同的服务器节点以N+X的形式合并入相同的一个服务器群组中，它们拥有相同的用户访问权限、部署着相同的应用与服务，大多数情况下也挂载着相同的来自中央储存的存储卷（例外情况是有写锁冲突的时候，比如两个MySQL节点尤其是主从结构中的双主机机无法共享相同的存储卷），自然也接受着来自监控系统Nagios相同阀值的监控，也处理着来自负载均衡器的相同请求。同时相同的服务器群组一般分配在相同得子网VLAN中，但这不是必须条件，也可以分配在其他子网VLAN中，因为这不影响他们相互通讯。

从上面的流程中可以看出，服务器名和服务器组是要在开始装机前就确定好的，这样的映射关系从安装操作系统开始一直贯穿着服务器的整个生命周期直到某一天它某一天被下架。

所以服务器组映射关系的管理成为本系统软件设计方面需要解决的第一个问题。

由于服务器节点和服务器群组的映射关系是系统中一个非常重要的基础数据，监控系统Nagios需要用到、配置管理系统cfengine需要用到、设置服务器登录访问权限也需要用到，而在所有得这几个子系统中只有Nagios是原生支持hostgroup/host这种映射关系的，所以我们的设计中，用Nagios的映射关系实现前文提到的分组策略，以将这个分组映射关系存放在监控系统Nagios中，进而以Nagios为中心，将之导出到其他系统中去。

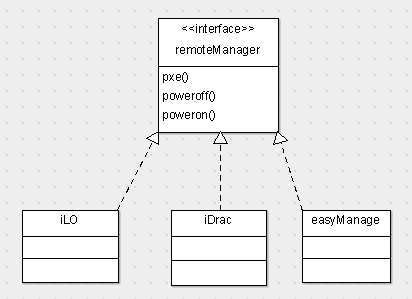


### 远程管理库

在服务器硬件上架完成之后，需要可以远程管理服务器，这样可以进行远程上电、断电、重装系统等等操作，而无需运维人员费时费力得亲自到机房现场让服务器从光盘或者网络启动以进行操作系统安装。

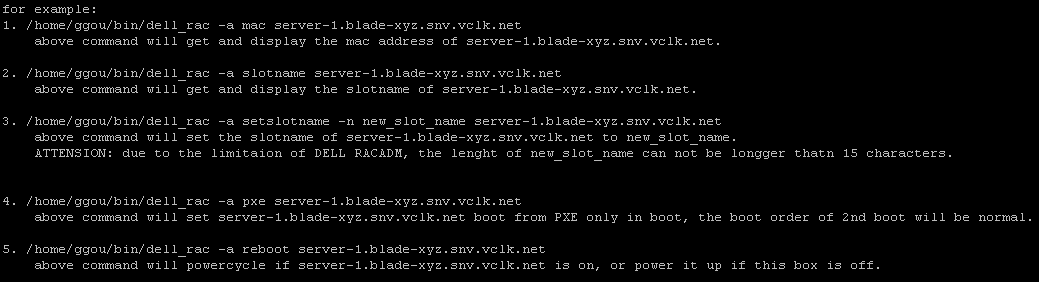
在业界，一些知名的服务器厂商提供特殊的硬件以支持这种远程管理，比如dell的dell Integrated Dell Remote Access Controller（简称iDRAC）硬件，HP的Integrated Lights-Out（简称iLO）或者联想的EasyManage。

经过调研，这些硬件提供的管理途径有多种：有基于web浏览器进行的GUI方式，也有命令行CLI方式，遗憾的是没有支持API方式的，而运维系统而言是需要API级别的支持，为此，利用Python语言，以模拟CLI的方式开发了一个API库，利用这个库，可以在命令行对服务器硬件进行管理，也可以接收来自运维系统的指令，以实现开关机等等服务器管理命令。

  
图一九一一年一月四日4‑4 RemoteManager接口

因为这个库需要同时和多个类型的硬件（iDrac、iLO、EasyManage等）进行交互，而又要保持向上层服务的统一性，而无需让库的调用方牵扯到过多的细节。为此借鉴“针对接口而非面向实现编程”的思想，所有需要调用远程管理库的调用方，简单的调用RemoteManager接口来进行服务器硬件的开关机等等管理，如所示，而无需关心这是个管理的服务器是dell还是HP的。

图4‑5展示了在命令行中利用这个库来对某个Dell机器进行断电、上电、重装等操作。



图一九一一年一月四日4‑5服务器命令行操作



### 监控系统Nagios

Nagios的配置文件中hostgroup/host都称之为对象，不同之处是hostgroup对象可以包含host对象，适用于于hostgroup级别的监控规则（触发报警的阀值，接收报警的联系人等等）会全部运用于隶属于该hostgroup的所有host成员。

表一九一一年一月四日4-3设置hostgroup

|  |
| --- |
| define hostgroup {  hostgroup\_name sh1.prd.foo.pl.adminweb.linux  alias sh1.prd.foo.pl.adminweb.linux  } |

表一九一一年一月四日4-4设置hostgroup成员

|  |
| --- |
| define host {  hostgroups sh1.prd.foo.pl.adminweb.linux  use host-prd-foo  host\_name admin101.sh1.foo.com  address admin101.sh1.foo.com  alias admin101.sh1.foo.com  } |

表4-3展示了如何设置hostgroup，表4-4这展示了如何将admin101这个host节点加入到sh1.prd.foo.pl.adminweb.linux 这一hostgroup中去。

在此处，hostgroup的命名约定一个通用的规范，即用点号“.”将各个部分隔开，从服务器节点的逻辑归属属性来看，从头至尾包含六部分内容：数据中心名称、服务级别（prd、dev、qa三种）、子公司名称、产品新名称、服务名称以及操作系统名称。以sh1.prd.foo.pl.adminweb.linux这个hostgroup名称为例：

1. sh1是机房名称
2. prd是生产环境
3. foo是子公司名称，之所以添加子公司名称，是因为作为基础架构，这套系统必须同时为多个子公司提供IT基础架构支撑。
4. pl是产品线名称
5. adminweb是服务名称
6. linux是服务器的操作系统名称

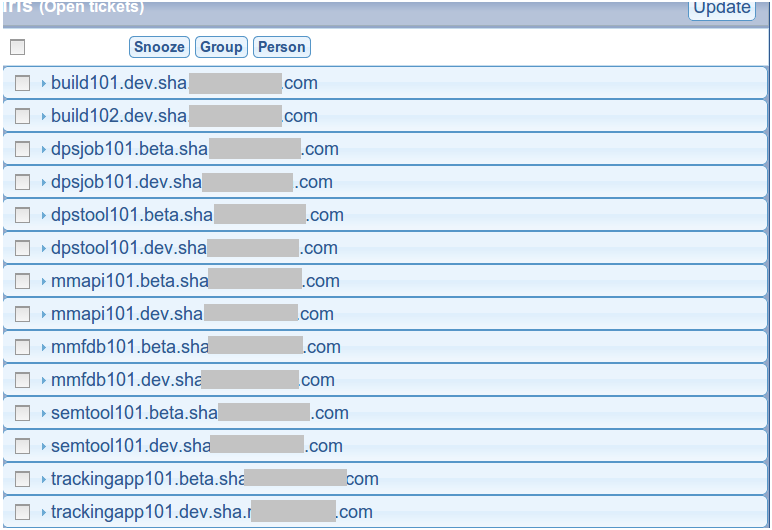
表一九一一年一月四日4-5基于hostgroup的监控

|  |
| --- |
| define service{  service\_description 5.http.status  check\_command http\_check  hostgroup\_name sh1.prd.foo.pl.adminweb.linux  use system-level-service  contact\_groups admin\_developer  }  define service{  service\_description 5.http.processes  check\_command named\_process!400 500 httpd  hostgroup\_name sh1.prd.foo.pl.adminweb.linux  use system-level-service  contact\_groups admin\_developer  } |

而表4-5则展示了在hostgroup这个服务器群组上增加了两个服务的监控，分别是http\_check和apache的进程数量（400和500分别是其warning和critical级别的阀值）监控，如果这两个监控项出现错误，无论是warnning级别还是critical级别，都会把通知发送到contact\_groups，此处contact\_groups是admin\_developer所代表的一个或者几个技术人员。

因为默认的Nagios开源实现功能不够丰富，尤其是报警方式的支持非常薄弱，它仅仅支持邮件报警，所以对它进行了深度二次开发，增加了下例功能：

* Nagios默认的报警只支持邮件和短信，这在技术人员8小时之外的非工作时间內，难以及时将报警送达，尤其是深夜熟睡的时候，所以利用Asterisk系统，通过IP PBX协议扩展出了电话报警功能，一有报警，便邮件加短信通知，如果因为熟睡等的原因不能在3-5分钟内响应，则出发打电话方式报警，以电话的方式通知支持人员，以确保报警可以第一时间送达正确的相关人员。
* 增加支持手机进行访问，比如通过手机就可以第一时间对问题服务进行SDT（schedule down time）、转发给本部门其他同事或者其他部门或者其他部门的人，如图4‑6所示。



图一九一一年一月四日4‑6手机访问

* 对Nagios添加了一些额外的辅助功能，比如在bugzilla中创建ticket以继续追踪问题的修复、技术人员响应速度评估等等辅助功能。

另外，因为hostgrou/host映射信息是存放在nagios系统中，而这个信息需要共享给CFEngine、sudo文件等其他系统，所以特别的加入了check\_MK[[[8]](#endnote-8)]支持，如

表4-3，表4-4所示Nagios中Hostgroup/Host映射关系的设置是格式比较自由的，但是对于其它子系统来说，这种自由格式对hostgroup/host数据的共享却是不利的，因为其它子系统无法理解进而处理这种格式。而check\_MK提供了一个可以实时的以类SQL语句（实时状态查询语言Livestatus Query Language，检查LQL）查询hostgroup/host数据、hostgroup/host/service/service status等等监控系统数据的途径，查询结果以csv格式返回，然后就非常容易的可以转换成其它子系统中的数据格式，只需要简单的TCP连接就可以直接向check\_MK发送LQL语句。

表一九一一年一月四日4-6 LQL查询

|  |
| --- |
| echo -e "GET services\nColumns: host\_groups host\_name description contacts\n\n" | nc monitor102 6557 |

如表4-6所示，该语句演示了如何简单的利用shell命令通过check\_MK监听的6557端口获取实时的hostgroup、host、监控的服务项以及报警联系人四项内容。

### 统一配置管理系统CFengine

CFengine作为整个运维系统的核心，其余子系统都是为它间接或者直接服务的，整个运维系统利用CFengine来做配置，既包含系统级别的基础配置，这些基础配置项对所有节点来说都是比较类似的；也包含应用级别的，这个级别的配置则是根据服务器群组来设置的，各个服务器群组之间一般是不一样的。

1. 配置系统基础环境，这部分几乎所有服务器都是一样的，有如下任务：
   1. 设置正确的三条内部路由，即10.0.0.0/8,172.16.0.0/12,192.168.0.0/16三条内部路由，以确保所有内部节点可以相互连通。
   2. 检查本节点是否需要Ext-caller的网络连接权限以设置正确的默认路由
   3. 启用SELinux，并设置正确的配置策略：设置正确的文件或者目录上下文。
   4. 设置统一LDAP认证环境以及挂载公共的HOME目录。
   5. 更新正确的/etc/sudoers，利用它来管理普通工程师帐号到应用role帐号的访问。
   6. 设置正确的DNS服务器，不同数据中心服务器节点使用的DNS是不一样的，但是同一个数据中心内的所有内部服务器节点都是有相同的DNS服务器，比如对应上海SH1数据中心，都是使用dns1.sh1.foo.com和dhs2.sh1.foo.com，而洛杉矶数据中心内的服务器节点则使用dns1.la.foo.com和dns2.la.foo.com。
   7. 设置正确的email转发，和DNS类似，每个数据中心有设置有各自的邮件转发器，以确保系统的各类邮件能够正确的投递。
   8. 设置时间服务，以确保所有数据中心中的所有服务器时间一致。
2. 配置应用的环境，这部分绝大多数服务器都是不一样的，也是最大的不同：
   1. 安装软件包，比如web服务器的apache和php，或者j2ee应用的jdk
   2. 挂载应用所需的存储卷，包括代码所在的存储卷和所使用数据的数据卷，
   3. 调整目录结构以及权限甚至SELinux上下文，比如修改session目录权限，调整
   4. 创建必须的符号链接，比如创建www.sample.com.conf到apache的vhosts目录下以为apache增加虚拟主机。
   5. 启动或者重启对应的应用和服务，比如启动MySQL服务
   6. 检查是否向负载均衡器注册本节点以成为正式的在线服务或者成为Ext-callee角色。
   7. 根据应用设置正确的内核参数。

### Kickstart/Anaconda系统

将服务的所属服务器组在报警系统确定好了之后，即可开始对操作系统的安装了。

在此处有一点是特殊之处是这些机器在安装操作系统之前就已经被加入到了监控系统了，这和先安装后监控的做法不一样，因为在安装任何一台服务器都是有目的的，换句话说已经确定了这个服务器会被安装成什么样的应用，是web服务器还是数据库服务器？而不会毫无目的的安装一台不知道安装目的的服务器。这就是为什么是先监控再安装。

当然在这之前还需要做好IP分配以及设置好内部机器名，以及对应的DNS正反向解析。

除了DNS，Kickstart/Anaconda对另外几个子系统也有依赖的，它们是

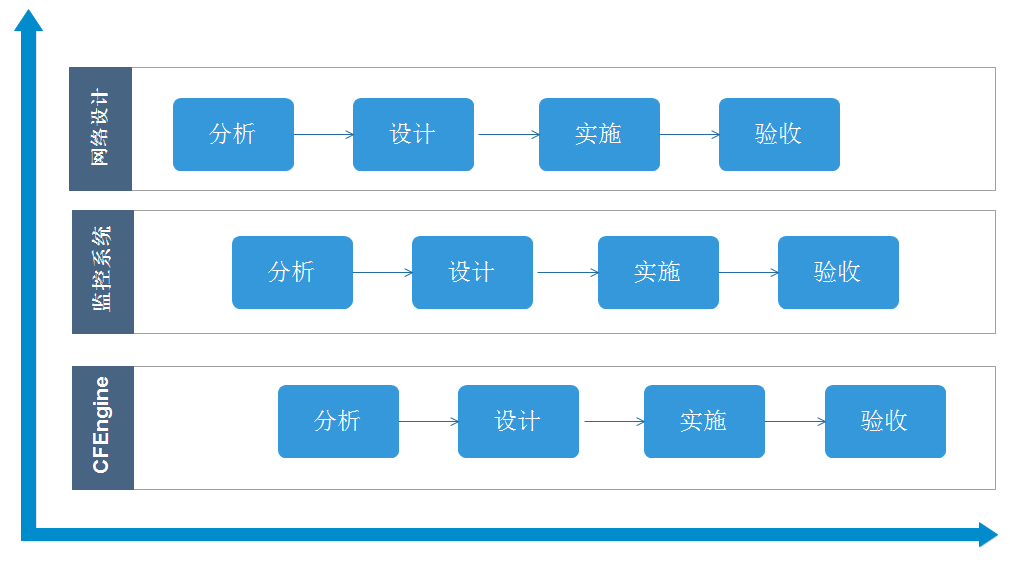
* DHCPD/Tftp以及在网络层需要支持DHCP relay，因为DHCP请求属于二层广播请求，无法跨交换机进行广播，为了避免在每一个网段中都设置一个DHCP服务器，需要在交换机上面将DHCP类型的广播请求转到某个单一的DHCP服务器上去。
* 每个数据中心都设置软件包管理系统yum仓库，为Kickstart/Anaconda提供安装源。同时这些软件仓库间使用rsync做同步。
* 动态的生成Kickstart/Anaconda所需的无人值守应答文件ks.cfg，之所以是动态生成，是因为不同数据中心安装操作系统的时候，一些参数是不同的，比如安装源每个数据中心不一样；需要使用的安装源需要为本数据中心的，否则如果所有数据中心都同一个远程安装源的话，比如上海数据中心的服务器节点使用美国数据中心的安装源会耗时很长，30M左右的pxelinux.0在只能使用UDP/tftp的PXE环境中，从美国洛杉矶数据中心发送到欧洲瑞典的斯德哥尔摩需要耗时两个半之多，另外一个问题是网络方面的设置尤其是实现DMZ所需的本子网VLAN的网关地址，需要通过SNMP的方式计算出来，而不能直接指定，如果没有网关地址，安装好的操作系统的路由表中将不具备任何路由：既没有10.0.0.0/8,172.16.0.0/12,192.168.0.0/16三条内部路由，更没有默认路由，如果这些路由都没有，会出现无法访问该服务器的情况。

当上面这些准备工作就绪之后，便可以利用服务器远程管理库，调用远程管理工具，比如dell的iDRAC，HP的iLO来开机并且从PXE启动以开始系统的安装。在安装完成的最后一步会再完成两个辅助任务：

1. 向图形系统进行注册，通知图形系统一个新的服务器节点上线，可以开始基础性能数据的抓取了，比如CPU使用率、系统的Load值、内存使用率、网卡的流量数据、磁盘的IO情况等等。
2. 向cfengine系统注册，发放cfagent的密钥，以让cfagent可以顺利的和cfservd进行通讯，复制cfengine配置策略文件，并在这个新的服务器节点上面开始第一次的CFEngine执行。

## 软件开发模型的选用

在3.2节中提到，由于已经有一套老的运维系统在线运行，虽然新的弹性系统是用来替换它的，但是众所周知运维系统涉及的方方面面都非常广泛，无法在短时间内完成对老系统的整体替换，为了顺利完成弹性系统的开发和新老系统过渡，我们在开发和部署过程中使用了增量模型，每一个增量标志着对老的运维系统的一部分替换，直至完成全部替换。



图一九一一年一月四日4‑7 运维系统中使用增量模型

在本设计中，由于网络部分处于整个系统最基础的部分，所以将它作为第一个增量来设计并完成部署，其余的依次是监控系统Nagios和统一配置系统CFEngine。

# 关键技术分析

CFengine作为整个运维系统的核心子系统，虽然它提供了强大的管理功能，但是并没有现成的策略，可以的让我们来管理这些大规模的系统，使之达到前文要求。如何制定合适全面而完整的CFEngine策略，使之将系统配置成前文所定的目标才是整个系统的关键所在。

要满足按组管理服务器的需求，则需要在CFEngine中设计出合理的类结构。



## CFEngine的类设计

CFEngine是一套独立的系统，其语法和传统的C++、Java等面向对象语言差别较大，但是其核心思想与这些以类为基础的语言比较类似，那就是将要管理的目标以类（class）来进行区别，然后针对这些区别出来的目标施加一个特定的指令（Action），以完成服务器的配置管理。



### CFEngine内置类及指令

默认时，CFEngine根据不同的运行环境，内部预置了一批内置类，如表5-1所示

表一九一一年一月五日5-1 CFEngine类表

|  |
| --- |
| [build101:~]$sudo /usr/sbin/cfagent -p -v | grep Defined  Defined Classes = ( 192\_168\_214 192\_168\_214\_133 64\_bit COMPANY\_mezi DATACENTER\_la1 Day28 EnterpriseLinux5 EtcHttpdConfIsSymlink HasDefaultGW Hr02 Hr02\_Q2 LEVEL\_prd Min25\_30 Min27 Monday NET\_192\_168\_214\_128 NeedsDefaultGateway OS\_linux PRODUCT\_build PanicOnOOM Q2 RebootOnPanic SERVICE\_web September UsesHttpd UsesUploads VCB\_LDAP Yr2015 any autofs\_local build101 build101\_la\_foo\_com build101\_la\_foo\_com\_foo1\_com cfengine\_2 cfengine\_2\_2 cfengine\_2\_2\_8 com com\_foo1\_com compiled\_on\_linux\_gnu httpd\_can\_network\_connect httpd\_can\_network\_connect\_db ipv4\_192 ipv4\_192\_168 ipv4\_192\_168\_214 ipv4\_192\_168\_214\_133 la\_foo\_com la\_foo\_com\_foo1\_com linux linux\_2\_6\_18\_348\_12\_1\_el5 linux\_x86\_64 linux\_x86\_64\_2\_6\_18\_348\_12\_1\_el5 linux\_x86\_64\_2\_6\_18\_348\_12\_1\_el5\_\_1\_SMP\_Mon\_Jul\_1\_17\_54\_12\_EDT\_2013 lsb\_compliant foo\_com foo\_com\_foo1\_com net\_iface\_eth0 net\_iface\_lo redhat redhat\_s redhat\_s\_5 redhat\_s\_5\_9 redhatenterpriseserver redhatenterpriseserver\_5 redhatenterpriseserver\_5\_9 redhatenterpriseserver\_tikanga foo1\_com x86\_64 ) [build101:~]$ |

这些预置的内部类，大多数是工具在不同的运行环境中是不一样的，比如在RHEL 5中有预置redhat redhat\_s、 redhat\_s\_5、 redhat\_s\_5\_8 redhatenterpriseserver、 redhatenterpriseserver\_5 这几个类。而在RHEL 6中则会有redhat 、redhat\_enterprise、 redhat\_enterprise\_6 这些类，对应的在CentOS 6.3中则没有redhat这样的预定义Classes，但是会预定义centos、centos\_6 centos\_6\_3、 centos\_final，根据这些内置的预定义Classes可以对这些系统进行初步的分类。

运维人员可以根据这些CFEngine设置的不同类的类来进行系统的配置，举例来说，可以设置类似于这样的策略：“centos这类服务器节点需要编辑/etc/http.conf.centos文件，redhat这类服务器节点需要编辑/etc/http.conf.redhat”， 这样就完成了把redhat的服务器节点和CentOS服务器按组进行区别配置。

配置是由编辑文件、创建符号链接等等不同的动作指令完成的，所谓动作指令就是让CFEngine在指定类所匹配的服务器节点上面需要执行的改动，如表5-2中，editfiles是CFEngine的动作指令，此处是editfiles这样的编辑文件动作，还有执行shell动作shellcommand、拷贝文件copy动作、创建链接links动作、以及一些CFEngine自身的控制动作等等，CFEngine正是通过这些动作完成对各项配置的改动。

表一九一一年一月五日5-2 动作指令

|  |
| --- |
| editfiles:  centos::  { /etc/http.conf.centos  #edit command  ｝  redhat::  { /etc/http.conf.redhat  #edit command  } |

上面提到的redhat、redhat\_s\_5、centos\_6\_3都是最简单的类，如果情况复杂些，想要同时识别RHEL或者CentOS的服务器节点的话，这可以使用CFEngine的组合类语法，所谓组合类是指由一个或者多个简单类通过逻辑与、逻辑或组合而成的类别标识语句：

* 逻辑与使用”.”表示，将两个简单类使用”.”连接起来即可，比如” ipv4\_192\_168\_214.redhat”便可以表示“某个既是redhat且在192.168.214子网VLAN的服务器节点”。
* 逻辑或则使用“|”连接一个或者多个简单类，比如”redhat|centos”就可以识别出RHEL的服务器节点或者CentOS服务器节点。

### 扩展CFEngine 类

很显然，仅仅依靠这些预置的内部Classes，CFEngine并无法完成我们目前的设计需求，必须对CFEngine的类进行扩展，才能满足按组管理所需。

如表5-3**Error! Reference source not found.**所示，如果IP地址是在10.32.17.0/24这个子网VLAN中，则定义NET\_10\_32\_17\_0这个自定义类。自定义的Classes也是可以用于组合类的。比如”redhat.NET\_10\_32\_17\_0”表示位于10.32.17.0/24这个子网VLAN中的RHEL服务器节点。

表一九一一年一月五日5-3根据网络扩展Class

|  |
| --- |
| NET\_10\_32\_17\_0 = ( IPRange(10.32.17.0/24) ) |

扩展类的时候，考虑到和Nagios系统数据交互的方便和概念的一致，参考了监控系统中的分组概念，并将之与CFEngine的类进行统一，即将监控系统中定义好的组，以CFEngine的格式，用”.”号分隔开的hostgroup形式映射到CFEngine中的自定义类，可以极大的帮助我们对服务器的管理。

表一九一一年一月五日5-4根据其他属性扩展类

|  |
| --- |
| adminweb101\_sh1\_foo\_com::  hostgroup = ( 'sh1.prd.foo.pl.adminweb.linux' )  hostgroup = ( A(DATACENTER,"sh1") A(LEVEL,"prd") A(COMPANY,"foo") A(PRODUCT,"pl") A(SERVICE,"adminweb") ) |

如表5-4所示，它将监控系统Nagios中sh1.prd.foo.pl.adminweb.linux 这样一个hostgroup映射为与CFEngine的一个扩展组合类，其中，数据中心部分的sh1就是一个自定义的简单类，那么可以将之定义为CFEngine的DATACENTER\_sh1这样一个简单类，与之类似的hostgroup名称sh1.prd.foo.pl.adminweb.linux 的其余部分这可以分别自定义这样一些CFEngine简单类:

1. 代表服务级别的简单类LEVEL\_prd
2. 代表子公司名称的COMPANY\_foo
3. 代表foo子公司下某个产品线的PRODUCT\_pl
4. 代表pl产品线中某个服务的SERVICE\_adminweb
5. 代表该服务器节点操作系统的OS\_linux

以上每个CFEngine类都可以有多个取值，但是都和监控系统中的hostgroup一一对应，比如监控系统中la1.qa.foo2.taobao.web.linux，则对应着在洛杉矶(la1)数据中心里中，隶属于另外一个名称是foo2的子公司的服务器节点，它上面运行着该子公司taobao这个产品线的用于测试(qa)的web服务。其对应的CFEngine类为DATACENTER\_la1.LEVEL\_qa.COMPANY\_foo2.PRODUCT\_taobao.SERVICE\_web.OS\_linux。

虽然CFEngine对组合类没有严格的顺序要求，比如DATACENTER\_la1.LEVEL\_qa和LEVEL\_qa. DATACENTER\_la1是一样的效果，但是做这种从监控系统Nagios中hostgroup到CFEngine的自定义类的时候，我们约定严格按照DATACENTER,SERVICE\_LEVEL,COMPANY,PRODUCT\_NAME,SERVICE\_NAME,OS\_NAME这样的顺序执行。

这样所有服务器都有一系列预定义和自定义的CFEngine class，只要综合的利用好这些class尤其是扩展出来的自定义class，可以非常方便的对服务器进行按组管理配置。

表一九一一年一月五日5-5根据类修改配置文件

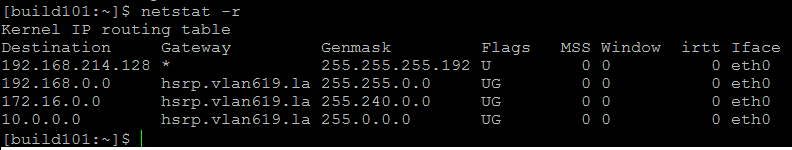
|  |
| --- |
| editfiles:  COMPANY\_foo.PRODUCT\_dw.(SERVICE\_job|SERVICE\_report)::  { /etc/selinux/targeted/contexts/files/file\_contexts.local  AutoCreate  AppendIfNoSuchLine "/usr/lib64/oracle/10.2.0.3/client/lib/(.\*)? system\_u:object\_r:textrel\_shlib\_t:s0"  AppendIfNoSuchLine "/usr/lib64/oracle/10.2.0.3/client/lib system\_u:object\_r:lib\_t:s0"  }  { /etc/php.ini  ResetSearch '1'  BeginGroupIfNoSuchLine 'upload\_max\_filesize = 10M'  ResetSearch '1'  HashCommentLinesMatching '[ ]\*upload\_max\_filesize[ ]\*=.\*'  ResetSearch '1'  LocateLineMatching '#?[ ]\*upload\_max\_filesize[ ]\*=.\*'  InsertLine 'upload\_max\_filesize = 10M'  EndGroup  CatchAbort  AppendIfNoSuchLine 'upload\_max\_filesize = 10M'  } |

比如，表5-5 的CFEngine 代码做了这样的配置：对于foo这个子公司的dw（data warehouse）产品的job和report两种服务的服务器，修改其SELinux配置和php.ini的参数。

## CFEngine中实现DMZ

如4.4节所述，在通过snmp取得子网VLAN信息之后，将之转换成CFEngine中的简单类，并且自动生成出CFEngine所需的各个子网VLAN网关等等数据，有了这些数据，通过简单的运算，便可以如下来实现修正之后的四条DMZ的网络访问策略

1. 禁止外网和内网的网络连接。提供在系统级别只设置三个内网网段的路由而不设定默认路由来实现，如图5‑1所示。这样内网服务器节点是无法访问外网的。而内网服务器节点本身不具备外网IP地址，故外网也无法访问内网的服务器节点。

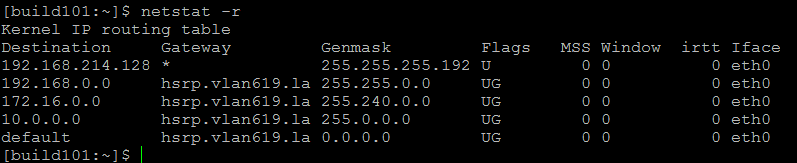


图一九一一年一月五日5‑1未设默认路由

1. 允许内网和DMZ自由双向通讯。一个可被外网访问的普通内网服务器节点就是DMZ中的一员，换句话说默认时，内网和DMZ中的服务器节点已经具备自由双向通讯的能力。
2. 外网到DMZ的访问受限。这是通过网络层的NAT来实现，大多数情况下是利用负载均衡器功能实现这一规则。这一层的NAT一般严格限制指定目标地址和端口的NAT，比如1.2.3.4:80到内网服务器192.168.0.4-10:80这七台web服务器的NAT。
3. 可自由单向访问外网的Ext-Caller。这个时候通过添加一条默认路由，其网关设置为本子网VLAN的第一个可以IP即可。比如要把图5‑1中的机器设置成可以访问外网的Ext-Caller，那么只需要增加一条这样的路由：将其所在子网VLAN的第一个IP设为默认网关即可，CFEngine实现代码见表5-6，其效果见图5‑2

表一九一一年一月五日5-6 CFEngine中设置默认路由

|  |
| --- |
| NeedsDefaultGateway.!Eth1::  { /etc/sysconfig/static-routes  AutoCreate  DefineClasses 'AddedDefaultGateway'  BeginGroupIfNoLineMatching 'any net 0\.0\.0\.0/0 gw $(defaultgw)'  InsertLine 'any net 0.0.0.0/0 gw $(defaultgw)'  ResetSearch '1'  LocateLineMatching 'any net 0.0.0.0/0 gw.\*'  DeleteLinesAfterThisMatching "any[ ]net[ ]0.0.0.0.\*"  EndGroup  } |



图一九一一年一月五日5‑2 已设默认路由

## CFEngine中实施帐号策略



### 统一帐号管理

默认时，各个服务器使用各自独立的/etc/passwd文件来进行帐号管理的，而在集群服务器环境下，必须对帐号进行统一管理。否则不同服务器上有不同的登录帐号，不仅仅有带来帐号管理的困难而且更加容易导致安全问题。在Windows服务器中，参见的是活动目录提供帐号管理，而Linux系统中则普遍使用LDAP作为帐号目录服务。

系统设计中，根据帐号用处的不同，将这些帐号分为两类帐号

1. 带有密码策略（一定密码强度、过期策略）的普通帐号，主要用户是需要登录系统执行开发的开发工程师、或者执行操作的操作员，或者执行系统管理的系统管理员。但是这些普通帐号上面禁止运行任何应用级别的服务。因为帐号的持有者可能有人员流动，离开公司后，普通帐号的修改和删除可能导致这些服务不可用，引发故障。
2. 永远不会过期的无密码role帐号。专门用来执行应用程序、运行服务进程等等。因为无密码、也不会过期，那么无需担心由此引发的应用/服务出现问题。

### 传统登录及问题

LDAP中的帐号和服务器节点之间默认是不做任何关联的，也就是说这些普通帐号以及role帐号是无法登录任何一个服务器节点。当然role帐号也不需要ssh登录服务器节点，因为它的访问是通过sudo的方式切换过去的。这样普通帐号登录服务器节点有下面两种方式

1. 明确的申请登录某个服务器群组，一般这适用于只读权限的帐号，比如审计人员，他们不需要实际操作任何应用。
2. 或者需要在某个服务器群组上面通过sudo访问某个role帐号，这适用于开发或者运维人员，他们不仅仅需要读应用的权限，更需要写的权限以及管理应用的权限，比如重启JAVA APP、重启数据库应用、重启Web服务等等。

Linux系统中身份认证服务是有PAM(Pluggable Authentication Modules)来进行的，根据不同的场景，PAM可以配合文本型用户数据库（/etc/passwd）或者其他类型的用户数据库进行认证。

在使用LDAP进行统一帐号管理的传统场景下，认证的LDAP客户端服务器通过PAM的LDAP模块与LDAP服务器建立LDAP连接，并且通过过滤帐号的host属性来判断该帐号是否具备登录该LDAP客户端节点的权限，如果如果host属性列表中没有该LDAP客户端节点，则说明该帐号没有权限，会被系统拒绝登录；反之，这同意该帐号登录该服务器节点。

在几乎所有的文献9,10都提到直接将LDAP客户端加入LDAP数据库中的用户Host属性，以此判断是否具备访问权限；而实际上，我们发现如果真这样做的话，会导致两个问题：

1. 很多帐号中会包含冗余信息，比如如果某个帐号需要登录一个web服务器组中的20台服务器节点时，就需要将这20个服务器节点都加到其host属性列表中，其实这20台服务器节点都隶属于同一个群组，完全没有利用起服务器群组这样的信息。
2. 新加或者删除服务器节点的时候，需要频繁的修改LDAP数据库，否则LDAP会慢慢累计下很多早已下线的服务器节点。

### 改进的登录方法

针对此情况，经过调研，我们在LDAP中，我们将host和hostgroup的概念切换一下，即可以解决上面两个问题，即PAM检查登陆时不是使用机器名去过滤，取而代之的是用本机所属的hostgroup去过滤，如表5-7，这样就把原来的按照host来验证切换到以hostgroup来验证

表一九一一年一月五日5-7重设PAM过滤条件

|  |
| --- |
| pam\_filter Host= sh1.prd.foo.pl.adminweb.linux |

对应的在LDAP中，将帐号的host属性设置为hostgroup即可，这样LDAP中只需要简单的存放少量的hostgroup信息即可，而无需记录所有的host信息，大大减小数据的容易，更重要的是由于服务器登录信息中不依赖LDAP中帐号host属性的设置，这样可以让服务器节点的上下线与LDAP服务尽可能的解耦。

### Role帐号的管理问题

通过上面的设置，普通帐号就可以ssh登录某个服务器节点了，但是无密码且永不过期的role帐号的使用却面临则无密码但是却需要使用的问题。

经过研究，我们发现可以使用sudo来解决这个问题。sudo是Linux服务器上一个身份切换的程序，它可以用另外一个帐号来执行命令的工具，普通场景是系统管理员用普通用户登录服务器然后用sudo切换到root执行进行服务器管理工作。

在我们的设计中利用sudo来帮助我们实现role帐号的管理。利用它来让普通帐号执行一个shell，这样就完成了由普通帐号到role帐号的切换。

并进一步的将sudo的管理文件/etc/sudoers中host 别名的概念用服务器组来“偷梁换柱”，这个别名的概念我们的设计中将之等同于服务器分组:

表一九一一年一月五日5-8 Host别名/分组

|  |
| --- |
| Host\_Alias FOO\_PL\_ADMINWEB\_PRD = admin101.sh1.foo.com, admin102.sh1.foo.com |

如表5-8所示，就可以把两个服务器节点admin101/102.sh1.foo.com加到sudo的hostgroup FOO\_PL\_ADMINWEB\_PRD中去。

然后进一步的指定哪个或者哪些帐号，在这个FOO\_PL\_ADMINWEB\_PRD host别名中可以何身份执行哪些命令。

在sudoers文件格式中，哪个或者哪些帐号可以用User\_Alias来设置，它的作用类似于给帐号分组，如表5-9：

表一九一一年一月五日5-9 用户别名/分组

|  |
| --- |
| User\_Alias FOO\_PL\_ADMINWEB\_DEV\_SUDO\_APACHE = user1, user2 |

哪些命令这个信息可以使用Command\_Alias来设置，它的作用类似于给命令分组，比如adminweb服务器群组上apache服务的管理命令，则如

*表5-10*分组

表一九一一年一月五日5-10命令别名/分组

|  |
| --- |
| Command\_Alias FOO\_PL\_PRD\_ADMINWEB = /sbin/service httpd \*, /sbin/service memcached \* |

最后把上面这些信息组合起来，给user1和user2两个帐号在服务器群组sh1.prd.foo.pl.adminweb.linux上面开通apache这个role帐号的sudo访问权限（即让user1和user2两个普通帐号可以用apache这个role帐号的身份执行包含shell在内的任何命令，如果是执行的是shell就相当于完成了从普通用户到web应用帐号apache的身份切换），则按表5-11设置。

表一九一一年一月五日5-11授予特定命令组访问

|  |
| --- |
| FOO\_PL\_ADMINWEB\_DEV\_SUDO\_APACHE FOO\_PL\_ADMINWEB\_PRD=(apache)ALL  FOO\_PL\_ADMINWEB\_DEV\_SUDO\_APACHE FOO\_PL\_ADMINWEB\_PRD=(root)FOO\_PL\_PRD\_ADMINWEB |

表5-11中，第二行则进一步的让这两个普通帐号可以以root帐号的身份执行FOO\_PL\_PRD\_ADMINWEB 所指定的两个命令来重启apache服务和memcached服务。之所以特别的需要root来执行这些命令，是因为有些服务的启动需要特权才能做到，比如监听80这样的低端口，比如操作/var目录等等。

### 帐号及登录映射信息的存放

至此，还有一个问题就是如何存放这样三类映射数据：

1. Hostgroup和host映射数据，这个已经存放在nagios中，只需要将之自动映射、导入到LDAP即可。
2. user和role的映射数据，也就是哪些用户可以切换到哪些role，这是一个多对多的关系，比如user1需要切换到web的role去管理web应用，同时还需要切换到hadoop去管理hadoop集群；
3. 以及role和该role所需的命令行映射数据，而role和role所需的命令行映射数据是一个一对多的关系，比如说一个web role需要执行的命令可以有很多个，但是有一个简化的策略即是用特殊关键词”ALL”来指定，即所有命令。

鉴于此，我们对LDAP的schema进行了扩展，openldap schema的扩展是有固定规范的[[[9]](#endnote-9)]，参照此我们扩展除了我们自己的schema，其核心部分如表5-12所示，这样在我们的LDAP中支持了设置hostgroup和host映射，以及为用户设置role映射这样的数据。

表一九一一年一月五日5-12 OpenLDAP Schema

|  |
| --- |
| attributetype ( 1.1.1.1.1 NAME 'hgName'  DESC 'A string identifying a nagios hostgroup'  EQUALITY caseExactIA5Match  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )  attributetype ( 1.1.1.1.2 NAME 'roleAccess'  DESC 'A string identifying a role which has access to this nagios hostgroup'  EQUALITY caseExactIA5Match  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 )  attributetype ( 1.1.1.1.3 NAME 'role'  DESC 'A string:time pair identifying a role which this person belongs to'  EQUALITY caseExactIA5Match  SUBSTR caseExactIA5SubstringsMatch  SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 ) |

结果这样的设计以及对LDAP的扩展，就完成了role帐号管理的支持。

上面这些host别名、user别名、command别名的命名规范、映射数据分别存放在监控系统以及LDAP中，监控系统里的host和hostgroup有一一对应关系的，LDAP中有user和role的对应关系。

我们利用Python程序LDAP模块，获取存放在LDAP的role和普通帐号的映射关系，而利用Python的socket功能，基于表4-6中的LQL语句，获取hostgroup/host映射数据，然后将这两组数据合并，便可自动生成出符合规范的sudoer文件，然后利用CFEngine的copy 指令将之分发到所有的服务器节点之上，实现所有服务器的/etc/sudoers集中管理，这样就完成了所有服务器节点上面的role帐号访问管理。

## CFEngine中的SELinux策略管理

为了让系统3.2节中TcsEC的C2安全级别，我们引入了SELinux，

SELinux是一个在Linux安全模型框架（Linux Security Modules）下实施的强制访问控制（mandatory access control），被公认为最杰出的新的安全子系统于2002年合并进Linux 2.6内核主线，在此之后，多个Linux发行版开始在2.6内核中使用不同程度的SELinux安全策略，最主要的贡献者和推动者要数NSA和Red Hat以Fedora Core Linux为基础进行的SELinux集成，通过对用户空间工具和服务的修改，是Fedora Core Linux具备了企业级的应用能力，一直到2005年，Red Hat发布的Enterprise Linux 4（RHEL 4）中默认开启SELinux，以此为标志，Linux平台的强制访问控制和SELinux才正式进入主流操作系统，在此情况下，研究显示在SELinux的保护下，Linux安全级别可以达到B1级别[[[10]](#endnote-10)]

关于它的策略配置和工具集已有一些研究[[[11]](#endnote-11)]，遗憾的是业界对SELinux的利用深度非常有限，互联网上最常见的话题居然是“如何关闭SELinux”，本节重点介绍如何在CFEngine环境中如何利用这些用户空间工具集进行SELinux的配置。

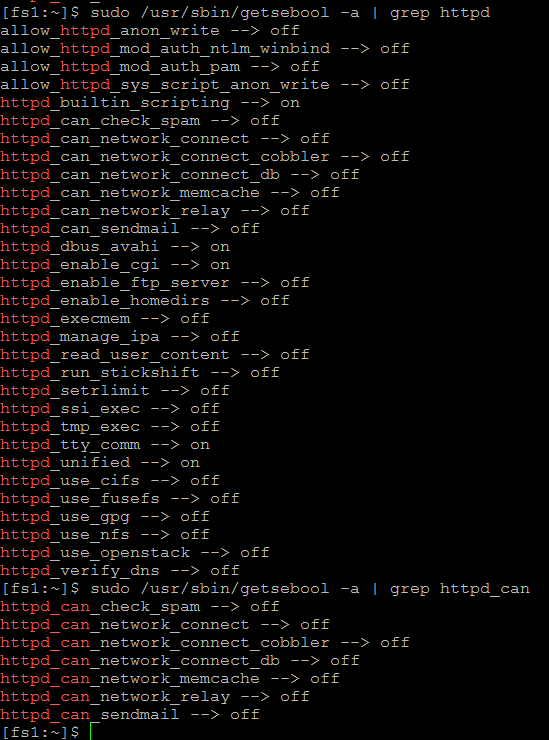
SELinux对服务器安全方面的设置有一些预置的开关，比如针对apache，有一些bool类型的开关，比如httpd\_can\_network\_connect等，默认时这些开关的值是off，即禁止apache（其进程名是httpd）进行网络连接。

除了socket这类的特殊文件之外，还有普通文件的安全上下文设置可以进一步的细化对服务器和服务器进程进行安全管理。



### SELinux中bool型配置

对于服务器尤其是服务进程，SELinux内置了大量的bool型参数，以控制服务器是否可以对相关资源进行访问，图5‑3显示了最常见的apache服务有关的bool型设置



图一九一一年一月五日5‑3 SELinux中httpd的bool型设置

它设定对apache进程了诸多的权限：能否访问HOME、能否进行网络接、能否连接数据库、能否访问NFS等等，默认情况下这些设置都是否，即不能访问。

表一九一一年一月五日5-13 SELinux中允许apache访问网络

|  |
| --- |
| # /usr/sbin/setsebool httpd\_can\_network\_connect=1 |

以httpd\_can\_network\_connect为例，如果需要允许apache访问网络，则需要将之设置为on，如表5-13所示，执行用户空间工具setsebool命令即可完成这个工作。

这个命令会将/selinux/booleans/httpd\_can\_network\_connects的内容设置为1，以立刻开启apache的网络访问权限。如果我们给setsebool加了-P参数则会将这个设置写入策略配置文件以在下次重启服务器之后这个设置已让生效。

而在集群环境中，我们必须借助FEngine来协助我们完成这个命令。

要允许前面的sh1.prd.foo.pl.adminweb.linux服务器群组中的apache服务访问网络、访问数据库，可以使用SELinux工具集中的setsebool命令或者直接编辑系统的/selinux/booleans/httpd\_can\_network\_connect\_db和/selinux/booleans/httpd\_can\_network\_connect两个文件将其内容设置为1。当然在CFEngine的帮助下，无需登录到该服务器群组中的单个服务器节点上去做这两个动作；只需要使用CFEngine的editfiles和shellcommands两个动作即可以完成此功能。代码如表5-14所示。

表一九一一年一月五日5-14 CFEngine中管理SELinux bool形配置

|  |
| --- |
| editfiles:  DATACENTER\_sh1.LEVEL\_prd.COMPANY\_foo.PRODUCT\_pl.SERVICE\_adminweb.OS\_linux::  { /selinux/booleans/httpd\_can\_network\_connect  BeginGroupIfNoLineContaining "1"  DefineInGroup "Enable\_httpd\_can\_network\_connect"  EndGroup  }  { /selinux/booleans/httpd\_can\_network\_connect\_db  BeginGroupIfNoLineContaining "1"  DefineInGroup "Enable\_httpd\_can\_network\_connect\_db"  EndGroup  }    shellcommands:  DATACENTER\_sh1.LEVEL\_prd.COMPANY\_foo.PRODUCT\_pl.SERVICE\_adminweb.OS\_linux::  "/usr/sbin/setsebool -P httpd\_can\_network\_connect\_db=1"  "/usr/sbin/setsebool -P httpd\_can\_network\_connect=1" |

### SELinux中文件content管理

以上是限制web服务器对网络socket进行安全管理，在linux系统中，socket是特殊文件，除此之外还可以限制对普通文件的访问限制，普通文件的SELinux属性主要集中在安全上下文的配置。与管理socket类似，只是CFEngine调用的SELinux命令不同而已，比如设置文件安全上下文使用”semanage fcontext”命令，从配置文件中恢复安全上下文使用“restorecon”等命令。

表一九一一年一月五日5-15 CFEngine中管理SELinux安全上下文

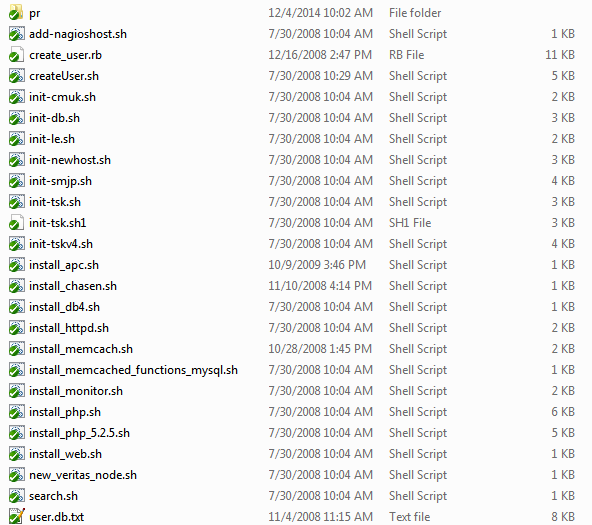
|  |
| --- |
| shellcommands: COMPANY\_foo.PRODUCT\_hadoop.SERVICE\_ganglia.LEVEL\_prd::  '/usr/sbin/semanage fcontext -a -f "" -s system\_u -t httpd\_sys\_content\_t "/u0/hadoop/ganglia/html(/.\*)?"'  '/sbin/restorecon -R "/u0/hadoop/ganglia/html(/.\*)?"'  '/usr/sbin/semanage fcontext -a -f "" -s system\_u -t httpd\_sys\_content\_t "/u0/hadoop/ganglia/gmetad\_root/rrds(/.\*)?"'  '/sbin/restorecon -R "/u0/hadoop/ganglia/gmetad\_root/rrds(/.\*)?"' |

表5-15展示了，在一组ganglia的系统中对ganglia的rrd数据文件设置特殊的安全上下文的操作。

这样通过对CFEngine扩展类与edit、shellcommand两个action的综合使用，即完成了在CFEngine中进行SELinux的管理工作。

# 效果分析

在此系统实施之前，基本上是使用手工的方式完成各个子系统的配置和管理的，在所有的环节中，系统配置这一部分最为繁复，极其容易出错，发展到后来如图6‑1所示的使用脚本部署的式，比如安装软件的脚本，系统配置脚本，账号管理脚本等等，



图一九一一年一月六日6‑1 传统的运维模式

但是新的问题随之而来受限于shell脚本语言本身，用其开发的系统配置脚本成长到一定规模之后难以维护，而且稳定性受到严重挑战，很多的改动常常出现牵一发而动全身的困境，在我们运用新的以CFEngine为核心的系统之后，有了CFEngine类的精准定义， 对系统的改动不仅仅可以精确到一个hostgroup，更是可以精确定位到单台主机，这大大提高了系统整体的鲁棒性。

而一旦碰到有需要临时的扩展服务器规模的场景时，可以在10分钟之内，将服务器池中的任意数量的空闲服务器完成部署，推上线提供服务。而且这些新上线的服务器无论是系统配置、应用配置还有安全配置，都与在线运行的系统，完全一致。这在管理员手工维护时代和脚本维护时代，是不可想象的。

目前该系统投入生产，稳定运行已数年之久年，以下数据简要罗利本系统目前管理的系统规模

1. 32908行CFEngine代码
2. 2454台服务器：2417(up)+37(down)
3. 16,840项服务,其中包括，15911项OK，85项警告，533项未知，311项紧急

随着业务的发展，这些数据还在持续增加中，极大的提高业务承载能力和服务水平。

这套系统也不是没有缺点，随着以投产之后的长时间使用，和不断演化组件暴露出下面一种业界称之为中心化的问题，在本系统中，数据存储大多依靠中央储存来解决；而数据交互为了支持负载均衡和失效冗余，大多通过负载均衡器来进行数据交互；从而出现了下面两个问题：



## 数据的存储中心化

数据存储中心化导致对存储设备依赖严重，众所周知，无论是NAS还是SAN，中央储存的费用非常昂贵，目前每个数据中心都必须部署多套netapp为主的NAS设备，考虑搭配对应级别的核心交换机以及机柜租用等等费用，更是巨大的开销，以最低端的FAS2000系列为例，一套造价约25万人民币左右，最低首次投入需要50万开销。而高端的FAS6000系列，造价约66万左右，搭配网络设备和机柜相关费用，首次投入则高达100万。而近年IT界知名大公司都在寻求去中心化的方法降低这方面的投入，尤其以google的GFS、淘宝TFS[[[12]](#endnote-12)]、阿里OceanBase为代表[[[13]](#endnote-13)]，这更是为我们下一阶段的系统发展方向指明了方向。

## 数据流量中心化

在整个系统中，为了保证每个服务单元的高可靠性，都以hostgroup为单元，在负载均衡器上面建立了虚拟IP(VIP)，该服务以VIP的方式暴露给服务使用方，比如表6-1中所示，5台web服务器的web流量全都经过负载均衡器来调度，

表一九一一年一月六日6-1 Web服务VIP范例

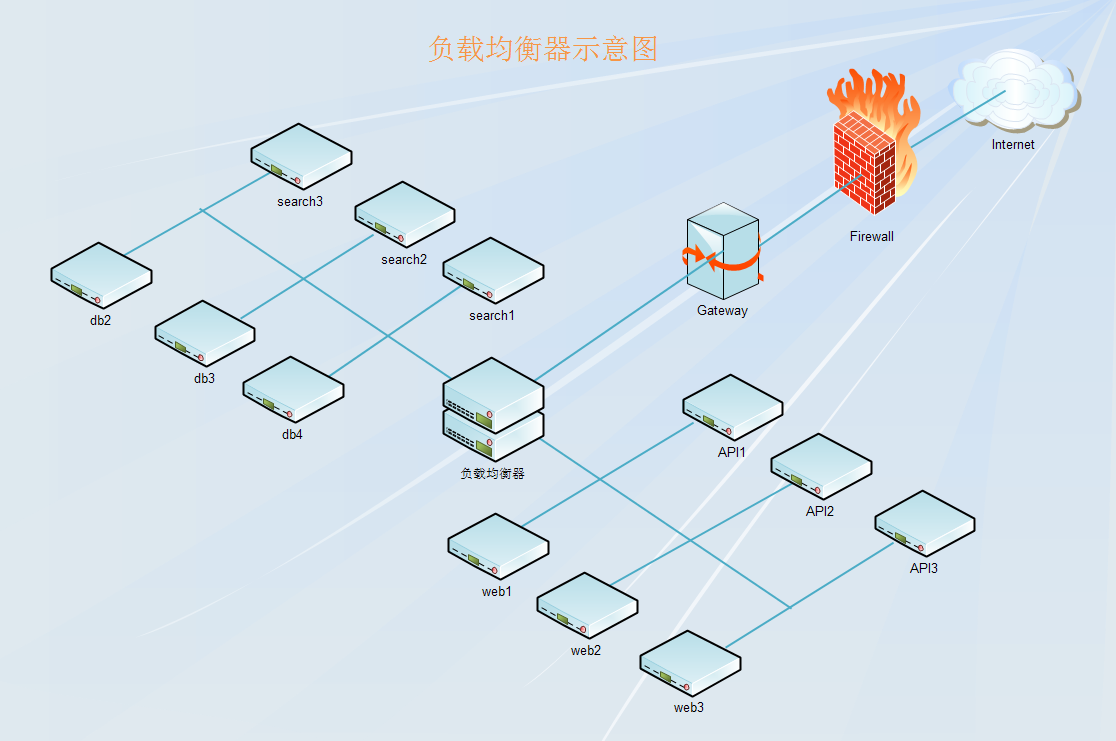
|  |
| --- |
| Bound Service Groups:  1) Group Name: LA1-MEZI-CSUS-SFD-HTTP-SG  1) LA1-MEZI-CSUS-SFD-HTTP-SG (192.168.211.25: 80) - HTTP State: UP Weight: 1  2) LA1-MEZI-CSUS-SFD-HTTP-SG (192.168.211.26: 80) - HTTP State: UP Weight: 1  3) LA1-MEZI-CSUS-SFD-HTTP-SG (192.168.211.27: 80) - HTTP State: UP Weight: 1  4) LA1-MEZI-CSUS-SFD-HTTP-SG (192.168.211.28: 80) - HTTP State: UP Weight: 1  5) LA1-MEZI-CSUS-SFD-HTTP-SG (192.168.211.29: 80) - HTTP State: UP Weight: 1  6) LA1-MEZI-CSUS-SFD-HTTP-SG (192.168.211.30: 80) - HTTP State: UP Weight: 1  1) CSPolicy: LA1-MEZI-CSUS-SFD-HTTP-PL CSVserver: VCB-drupal-contentsites-HTTP-CS Priority: 85 Hits: 1428713  2) CSPolicy: LA1-MEZI-CSUS-SFD-HTTP-PL-351363 CSVserver: VCB-drupal-contentsites-HTTP-CS Priority: 50 Hits: 266254648 |

同样的，中所示的数据库流量也都经过负载均衡器来调度，

表一九一一年一月六日6-2

|  |
| --- |
| Bound Service Groups:  1) Group Name: VCINV-db-3306-TCP  1) VCINV-db-3306-TCP (10.16.3.25: 3306) - TCP State: UP Weight: 1  2) VCINV-db-3306-TCP (10.16.3.26: 3306) - TCP State: UP Weight: 1  3) VCINV-db-3306-TCP (10.16.3.27: 3306) - TCP State: UP Weight: 1  4) VCINV-db-3306-TCP (10.16.3.51: 3306) - TCP State: UP Weight: 1 |

当越来越多的这种VIP上线之后，这样一来就形成了一个逻辑上以负载均衡器为中心结点的网状结构，如图6‑2所示，在这个结构中负载均衡器成为整个系统逻辑上的中心，它负担着处理整个系统的TCP连接直至请求结束该TCP连接被关闭的重任，它成为整个系统的关键点，它的性能决定着这个系统的性能，它的吞吐量决定着这个系统的吞吐量，它的稳定性决定着整个系统的稳定性，即便我们部署了双负载均衡器的情况下，依然出现过宕机事件，原因是当其中一个负载均衡器失效时，原来双负载均衡器承担的流量全部集中在剩下的负载均衡器上，导致它很快也因为过载而失效。



图一九一一年一月六日6‑2 负载均衡器形成逻辑上的中心

解决此问题，除了通过内容分发网络CDN（租用CDN服务或者自建CDN集群）加速来分摊负载均衡器流量外，可选的方案有改进型的负载均衡调度工作模式以及基于异地多活数据中心的DNS轮询式负载均衡。

改进型的负载均衡调度工作模式是指目前业界通用的负载均衡器都是使用NAT技术实现负载均衡的，而除此之外还有基于IP隧道[[[14]](#endnote-14)]和直接路由[[[15]](#endnote-15)]以及TCP迁移[[[16]](#endnote-16)]这三种模式，三种模式各有优劣，IP隧道要求后端服务器支持IP隧道协议，直接路由则要求网络层关闭ARP功能，IP隧道和直接路由在开源项目LVS有直接支持，而TCP迁移也被TCPHA[[[17]](#endnote-17)]所实现，相较而言，后面三种实施方案中，负载均衡器都只是负责将请求的TCP包简单快速的迁移给后端服务器，再有后端服务器直接和客户端进行后续TCP请求的处理，而在负载均衡器处无需保持整个TCP会话，这无疑极大的提升负载均衡器的吞吐量进而整体提高了整个系统的吞吐量。

而对于异地多活数据中心DNS轮询方案，这是一个基于全国乃至全球范围内的异地多活机房的方案，所以在F5和Akamai等商业解决方案中，也将之命名为global traffic management，这是一个实施难度与数据实时性成正比的方案，越是方便静态化的服务这个方案越容易实现，比如国内的网易新闻、新浪新闻等等网站比较适合，而实时性要求越高，其实现难度越大，国内较早开展异地多活数据中心项目的阿里在2014年双十一上线数据中心异地双活项目，而在2015年5月27号，支付宝杭州光纤被挖之后，完成异地多活数据中心业务切换也耗时2个小时，其难度可见一斑。

两种方案实施成本来看改进型的负载均衡调度无疑是最容易实施且成本最低的，这也是本系统将来可能的一个演化方向。

# 总结与展望

随着以及业务规模的发展以及以虚拟化技术为代表的云平台的技术发展，也对我们这套系统提出了新的挑战，尤其是以虚拟化为方向的整合。

虚拟化作为IT业近年的发展浪潮，我们也尝试过将之融入到现有的系统，因为多一层hypervisor导致额外的资源开销，IO密集或者CPU密集类应用并不适合，而对于低IO或者低CPU需求类应用，切换到虚拟机平台之后，也作为一个独立的逻辑主机，最终依然纳入到以CFEngine为核心的平台来进行管理。因为CFEngine来说，它不关心管理的对象是否是真是的物理机，只要是逻辑上一个服务器即可，这在一个侧面的展示了该系统良好的扩展能力。

另外近年出现的一些新的虚拟化配置管理框架诸如puppet，Chef等等，相比而言它们更具备一些典型的开放的特点，比如OpenAPI等等，在这方面CFEngine依然有其天然的不足，因为这会影响其他系统与之的交互能力。这也是本系统将来持续演进的一个方向。

参考文献

致谢

在复旦大学攻读工程硕士学位期间，我得到了许多老师、同学的帮助，在此谨向各位老师、同学致以真诚的谢意！

感谢导师的悉心的指导和严格的要求。在本文的写作过程中，给予了我热情的关怀和指导，这将使我终生难忘。老师渊博的知识、严谨的治学态度使我受益匪浅，这些无疑将对我未来的工作产生很大的影响。在此谨向恩师表示衷心的感谢！

最后，感谢所有帮助过我的人！

**论文独创性声明**

本论文是我个人在导师指导下进行的研究工作及取得的研究成果。论文中除了特别加以标注和致谢的地方外，不包含其他人或其它机构已经发表或撰写过的研究成果。其他同志对本研究的启发和所做的贡献均已在论文中作了明确的声明并表示了谢意。

作者签名： 日期：\_\_\_\_\_\_\_\_\_\_\_\_

**论文使用授权声明**

本人完全了解复旦大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。保密的论文在解密后遵守此规定。

作者签名： 导师签名： 日期：\_\_\_\_\_\_\_\_\_\_\_\_

1. [] Paul Albitz, 房向明等译, DNS与BIND(第5版) [M], 人民邮电出版社 [↑](#endnote-ref-1)
2. [] Gerald Carter , LDAP System Administration [M], O'Reilly Media, Inc, 2003-4-1 U.S.A [↑](#endnote-ref-2)
3. [] Matt Butcher, Mastering OpenLDAP: Configuring, Securing and Integrating Directory Services [M], Packt Publishing Limited, U. S. A. [↑](#endnote-ref-3)
4. [] <http://www.ntp.org/downloads.html> [EB/OL] [↑](#endnote-ref-4)
5. [] <http://www.spamhaus.org/sbl/query/SBL258296> [EB/OL] [↑](#endnote-ref-5)
6. [] <http://www.emailsecuritygrader.com/results?id=134258> [EB/OL] [↑](#endnote-ref-6)
7. [] <http://www.openspf.org/SPF_Record_Syntax> [EB/OL] [↑](#endnote-ref-7)
8. [] <http://mathias-kettner.de/check_mk.html> [CP/OL] [↑](#endnote-ref-8)
9. [] http://www.openldap.org/doc/admin24/schema.html#Extending Schema [EB/OL] [↑](#endnote-ref-9)
10. [] Bill McCarty, SELinux [M], O'Reilly Media, Inc. 1,4 [↑](#endnote-ref-10)
11. [] 倪继利, linux安全体系分析与编程[M], 电子工业出版社 [↑](#endnote-ref-11)
12. [] <http://tfs.taobao.org/> [EB/OL] [↑](#endnote-ref-12)
13. [] 杨传辉,大规模分布式存储系统:原理解析与架构实战[M],机械工业出版社 [↑](#endnote-ref-13)
14. [] <http://zh.linuxvirtualserver.org/node/27> [EB/OL] [↑](#endnote-ref-14)
15. [] <http://zh.linuxvirtualserver.org/node/28> [EB/OL] [↑](#endnote-ref-15)
16. [] 汪黎，TCP 迁移技术报告 [R/OL].2005. <http://zh.linuxvirtualserver.org/files/TcpHandoff.pdf> [↑](#endnote-ref-16)
17. [] TCPHA Project [CP/OL]. <http://dragon.linux-vs.org/~dragonfly/htm/tcpha.htm> [↑](#endnote-ref-17)