



Puppet 实战

◎ 作者：刘宇

前言

为什么要写这本书

子曰：“工欲善其事，必先利其器。”作为系统管理员，最应该具备的一种技能就是利用各种优秀开源软件快速完成自己的工作，让自己变得更轻松。但并不是所有的软件都适合你，你需要根据自己的环境及需求进行选择。

Puppet 就是系统管理员的一把利器。几年前我开始学习 Puppet 的时候，不少朋友都问我：为什么选择 Puppet？它有什么优势？当时 pssh、CFEngine 等都已经很成熟，并且也能帮助系统管理员有效地解决繁重的工作。为什么还要选择 Puppet？“需求决定产品”。Luke Kanies (Puppet 作者，以下简称 Luke) 曾经全面参与了重写 CFEngine 的解析器和开发 ISConf3 的工作，但是他还是觉得现有的工具已经无法满足他的工作需求，需要自己创造一个全新的工具，才能彻底使他的工作更加高效、便捷。这便是 Puppet 诞生的背景和原因。

Puppet 注重设计简洁、先框架后应用两个中心思想。这也造就了 Puppet 今日的成功，它的“三板斧”——资源、类、模板，可以轻松地帮我们完成复杂的业务逻辑关系管理。同时，Puppet 并不具备执行功能，因此在某些程度上存在短板，（比如 `exec` 是为了解决特定系统管理员蹩脚的执行命令需求而开发的，但 Luke 一再强调不建议使用。）

或许更多的系统管理员抱怨 Puppet 没有命令执行功能。Luke 早就考虑到了这一点，并提前收购了 MCollective，采用消息型总线的中间件来实现命令执行、系统管理、Puppet 客户端管理等，以弥补了 Puppet 在这方面的不足，可见 Puppetlabs 非常有远见。伴随着 Puppet 各子功能及扩展工具的遍地开花，Puppet 的商业化及各种开源社区的支持，Puppet 可谓是蒸蒸日上。

在新兴的同类工具中，我的另一个关注点是 SaltStack，它天生具备命令执行和配置管理两大核心功能，相比 Puppet 来说有一定优势，但 SaltStack 的成长还需要我们耐心等待。Puppet 打败 1993 年“出生”的老牌系统配置管理工具 CFEngine 就花费了近 7 年时间。而 Puppet 与 Salt Stack 真正鹿死谁手，我们拭目以待。相信以后的市场竞争会越来越激烈。

为了让更多的系统管理员了解并深入学习 Puppet，可以利用 Puppet 的集成方案解决系统管理复杂而繁重的任务，而不是盲目地寻找问题，我觉得有必要结合自己在学习 Puppet 过程中走过的弯路将工作中的经验和总结以实战形式呈现给大家。我也希望更多的人能加入开源社区，拥抱开源，拥抱变化，通过学习 Puppet，体会到与开源爱好者交流的乐趣，而不是为了工具选型而犹豫甚至争斗。这也是我写作本书的初衷。

本书技术深入而阅读简单，涉及系统管理员所需的很多方面的基础知识，同时通过穿插各种实例及代码详解以便使读者能够快速掌握 Puppet，并迅速将其运用到工作环境当中。通过这样一本以系统管理员为核心的书，希望能给读者带来的不只是技术能力提升，更多的是开源与奉献精神。也希望更多的系统管理员通过多阅读、多交流，建立起享受分享的技术氛围。

读者对象

根据本书内容定位，适合阅读本书的读者有：

- ❑ Puppet 用户和爱好者
- ❑ UNIX 与 Linux 系统管理员
- ❑ 运维工程师

如何阅读本书

本书分为四大部分：

第一部分为准备篇（第 1 ~ 4 章），简单地介绍了 Puppet 的发展历程和相关理论，帮助读者了解一些基础背景知识，并快速搭建测试环境。

第二部分为基础篇（第 5 ~ 10 章），着重讲解 Puppet 的基础理论知识，包括语法、资源、类、模板、模块、节点、Facter、数组、函数、变量。结合不同实例让读者感觉理论知识不再那么枯燥。

第三部分为实战篇（第 11 ~ 13 章），从实战角度进行讲解，结合流行监控系统 Nagios 和 Zabbix，包括最为热门的云计算 OpenStack 的部署，使读者能快速掌握 Puppet 并运用到实践中。

第四部分为进阶篇（第 14 ~ 20 章），通过对 Puppet 扩展模式、版本控制、报告系统、控制台、扩展工具及 MCollective 的综合讲解，让读者了解一个完整的 Puppet 生产流程。

其中第三部分以实战来讲解 Puppet 应用，相比于前两部分更加复杂。如果你是一名经验丰富的资深用户，能够理解 Puppet 的相关基础知识和工作原理，那么你可以直接阅读这部分内容。但是如果你是一名初学者，请一定从第 1 章的基础理论知识开始学习。

勘误和支持

由于作者的水平有限，加之编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。为此，我特意在博客（<http://liuyu.github.io>）的基础上开辟了在线支持与应急预案的站点 <http://puppet.bubbyroom.com>。你可以将书中的错误以及遇到的有关 Puppet 的任何问题，通过访问该站点的 Q&A 页面提交给我。书中的全部源文件除可以从华章网站^①下载外，还可以从这个站点下载。同时我也会将相应的功能更新在这个站点上及时发布出来。如果你有更多的宝贵意见，也欢迎发送邮件至邮箱 liuyu105@gmail.com (`sed 's/#/@/g'`)，标题请注明《Puppet 实战》，期待能够得到你们真挚反馈。同样你也可以关注我为本书建立的微信公共平台（[puppetchina](#)），我会在此公共平台定期发布 Puppet 相关信息。

致谢

首先要感谢伟大的 Puppet 作者 Luke Kanies，他开发了一款影响我整个人生的软件。

特别感谢刘长元、王广胜、吴问志、刘继伟、王哲，感谢你们对本书提出的宝贵修改意见。

感谢为本书撰写的童剑先生、窦哲先生、田逸先生、程辉先生，感谢你们在繁忙的工作中抽出时间，阅读了本书的样稿并写下推荐和评论。

感谢机械工业出版社华章公司的编辑杨福川，是你的引导才促使我完成这本书。特别感谢姜影编辑，在这一年多的时间中始终帮我校对并支持我的写作，你的鼓励和帮助引导我顺利完成全部书稿。

最后感谢我的笨笨，一如既往地支持我写作，给予我无尽的支持与灵感，并时时刻刻为我灌输着信心和力量！

谨以此书献给我即将出生的 Baby，以及众多热爱 Puppet 的朋友们！

刘宇（守住每一天）

于中国北京

① 参见华章网站 www.hzbook.com。——编辑注

目 录

前 言

第一部分 准备篇

第 1 章 认识 Puppet.....2

1.1 Puppet 的起源与发展现状.....2

1.1.1 什么是 Puppet.....2

1.1.2 Puppet 起源与发展.....2

1.1.3 版本语言特征.....3

1.1.4 命令差异.....4

1.1.5 Puppet 3.0 新特性.....4

1.2 为什么要使用 Puppet.....5

1.2.1 都有谁在使用 Puppet.....5

1.2.2 常见集中化管理工具对比.....5

1.2.3 推荐 Puppet 的理由.....6

1.3 Puppet 的作用和特色.....6

1.3.1 为什么要有自己的语言.....6

1.3.2 为什么是 Ruby.....6

1.3.3 管理任何机器.....6

1.4 Puppet 组织结构.....7

1.5 Puppet 工作原理.....7

1.5.1 Puppet 基本结构.....7

1.5.2 Puppet 是如何工作的.....8

1.5.3 Puppet 数据流.....8

1.5.4 文件结合.....9

1.5.5 详细交互过程.....9

1.5.6 安全与认证.....10

1.6 Puppet 核心配置文件详解.....11

1.6.1 主配置文件 puppet.conf.....11

1.6.2 主机配置文件 site.pp.....13

1.6.3 认证与安全配置文件.....14

1.6.4 客户端自动认证配置.....16

1.6.5 报告系统配置.....16

1.6.6 文件系统配置文件.....16

1.7 本章小结.....17

第 2 章 Puppet 安装与配置.....18

2.1 Puppet 对各系统平台的支持.....18

2.2 Puppet 对 Ruby 的支持.....19

2.3 Puppet 的安装步骤.....19

2.4 在 Linux 下安装.....20

2.4.1 包管理器方式安装.....21

2.4.2 从源代码进行安装.....23

2.4.3 从 Git 版本库进行安装.....24

2.4.4 通过 Gems 进行安装.....25

2.5 在 Mac OS X 下安装.....25

2.5.1 通过二进制发布包进行 安装.....25

2.5.2 从 Git 版本库进行安装·····	26	5.1.3 类·····	52
2.5.3 从 Ports 仓库进行安装·····	27	5.1.4 模块·····	52
2.6 配置 Puppet·····	28	5.1.5 节点·····	52
2.7 在 Windows 下安装与使用·····	30	5.2 主机、模块和类的命名·····	53
2.7.1 包管理器方式安装·····	31	5.2.1 主机的命名·····	53
2.7.2 在 Windows 下使用 Puppet···	33	5.2.2 模块的命名·····	54
2.7.3 Puppet 在 Windows 下的 功能·····	35	5.2.3 类的命名·····	54
2.8 如何升级·····	35	5.3 资源、变量、参数和标签的命名···	54
2.9 本章小结·····	36	5.3.1 资源的命名·····	54
第 3 章 创建你的第一个 Puppet 配置·····	37	5.3.2 变量的命名·····	55
3.1 配置一个测试节点·····	37	5.3.3 参数的命名·····	55
3.2 检测你的配置文件·····	38	5.3.4 标签的命名·····	55
3.3 客户端运行配置·····	39	5.4 Puppet 语法风格·····	55
3.4 查看运行结果·····	40	5.4.1 间距、缩进和空白字符·····	56
3.5 增加 httpd 模块·····	40	5.4.2 注释·····	56
3.6 本章小结·····	42	5.4.3 变量的引用·····	56
第 4 章 Puppet 运行环境·····	43	5.4.4 资源·····	57
4.1 服务器端配置·····	43	5.4.5 条件语句·····	60
4.2 客户端配置·····	44	5.4.6 类·····	61
4.3 如何运用环境配置·····	45	5.5 检查命令的用法·····	64
4.4 本章小结·····	48	5.5.1 语法检查·····	64
第二部分 基础篇		5.5.2 代码调试·····	65
第 5 章 Puppet 语法与命令详解·····	50	5.6 Puppet 命令详解·····	66
5.1 Puppet 的命名规范·····	51	5.6.1 Puppet 常用命令·····	68
5.1.1 资源·····	51	5.6.2 帮助命令详解·····	73
5.1.2 属性·····	51	5.6.3 模块和不常用命令·····	75
		5.7 本章小结·····	76
		第 6 章 Puppet 资源详解·····	77
		6.1 什么是资源·····	78
		6.1.1 图解核心资源·····	81
		6.1.2 什么是 manifests·····	81

6.1.3 资源的依赖·····	82	6.8.4 资源引用·····	120
6.2 虚拟资源·····	85	6.8.5 数字·····	120
6.2.1 虚拟资源的定义·····	85	6.8.6 哈希类型·····	121
6.2.2 虚拟资源的用法·····	86	6.8.7 正则表达式·····	121
6.3 常用资源的用法·····	87	6.8.8 数组·····	122
6.3.1 用户资源·····	88	6.9 标签·····	123
6.3.2 用户组资源·····	90	6.10 stage 运行阶段·····	123
6.3.3 软件安装·····	91	6.11 本章小结·····	124
6.3.4 文件管理·····	94		
6.3.5 服务管理·····	97	第 7 章 Puppet 模块、类、模板·····	125
6.3.6 定时脚本·····	99	7.1 图解模块结构·····	125
6.3.7 执行命令·····	101	7.2 模块管理·····	126
6.3.8 调试与输出·····	103	7.2.1 实例：创建一个模块·····	127
6.4 Puppet 作用域与变量·····	104	7.2.2 模块布局·····	129
6.4.1 作用域·····	104	7.3 类管理·····	130
6.4.2 变量·····	108	7.3.1 类的定义·····	131
6.5 条件语句·····	110	7.3.2 类的继承·····	131
6.5.1 if 语句·····	110	7.3.3 参数化类·····	132
6.5.2 case 语句·····	112	7.4 模板管理·····	135
6.5.3 selector 选择器·····	112	7.4.1 定义与声明·····	135
6.6 表达式·····	113	7.4.2 ERB 模板语法·····	136
6.6.1 什么是表达式·····	113	7.5 融合·····	139
6.6.2 运用位置·····	114	7.6 从 Puppet Forge 获取模块·····	141
6.6.3 操作顺序·····	114	7.7 从 Example42 获取模块·····	142
6.6.4 比较运算符·····	114	7.8 本章小结·····	143
6.6.5 布尔运算符·····	115		
6.6.6 算术运算符·····	116	第 8 章 节点管理·····	144
6.7 函数·····	116	8.1 什么是节点·····	144
6.8 数据类型·····	118	8.2 主机名命名规范·····	145
6.8.1 布尔类型·····	118	8.3 节点继承·····	146
6.8.2 未定义·····	119	8.3.1 节点继承关系·····	146
6.8.3 字符串·····	119	8.3.2 继承变量覆盖·····	147

11.4 实施步骤·····	183	13.2.1 环境准备·····	214
11.4.1 前期准备：创建软件 仓库·····	183	13.2.2 安装软件及 Puppet 模块···	216
11.4.2 Puppet 配置文件管理·····	185	13.2.3 部署 controller·····	218
11.4.3 初始化操作系统·····	187	13.2.4 部署 compute·····	220
11.4.4 编写 nginx 模块·····	187	13.2.5 验证 OpenStack 部署·····	221
11.4.5 采用 Forge 的 nginx 模块·····	192	13.3 本章小结·····	222
11.5 本章小结·····	194		
第 12 章 分布式监控系统部署方案···	195	第四部分 进阶篇	
12.1 利用 Puppet 部署 Zabbix·····	196	第 14 章 Puppet 版本控制·····	224
12.1.1 Zabbix 简介·····	196	14.1 Puppet 版本控制方法·····	224
12.1.2 Zabbix 架构·····	197	14.1.1 为什么要使用版本控制···	224
12.1.3 利用 Puppet 部署 Zabbix···	198	14.1.2 版本控制的架构与原理···	225
12.1.4 Zabbix 自定义监控·····	201	14.1.3 Git 与 SVN 的区别·····	226
12.2 利用 Puppet 部署 Nagios·····	202	14.1.4 为什么采用 Git·····	226
12.2.1 Nagios 简介·····	202	14.2 使用 Git 实现 Puppet·····	226
12.2.2 Nagios 架构·····	203	14.2.1 安装与配置 Git·····	227
12.2.3 Nagios 服务端安装·····	204	14.2.2 将 Puppet 加入 Git·····	228
12.2.4 Nagios 模块应用·····	206	14.2.3 使用 Rake 自动更新副本···	229
12.2.5 创建 Nagios 客户端监控···	208	14.2.4 使用 hook 实现自动 语法检查·····	231
12.3 本章小结·····	210	14.3 本章小结·····	232
第 13 章 OpenStack 快速部署 方案·····	211	第 15 章 Puppet 架构扩展与分 布式·····	233
13.1 OpenStack 简介·····	211	15.1 Puppet 瓶颈分析·····	233
13.1.1 什么是 OpenStack·····	211	15.1.1 单台 Puppet Master 瓶颈···	233
13.1.2 OpenStack 的组件、服务 及逻辑架构·····	212	15.1.2 认证的瓶颈·····	234
13.1.3 OpenStack 版本说明·····	213	15.1.3 文件的瓶颈·····	234
13.2 部署 OpenStack·····	214	15.1.4 网路的瓶颈·····	234
		15.2 架构扩展之单台 Puppet Master···	234
		15.2.1 Nginx+Mongrel 模式·····	235

15.2.2	Apache+Passenger 模式	238	16.7	File 资源的缺点	276
15.2.3	Nginx+Passenger 模式	242	16.8	本章小结	276
15.3	架构扩展之多台 Puppet Master	244	第 17 章	强大的报告系统	277
15.3.1	配置前的准备	248	17.1	report 介绍	277
15.3.2	Puppet CA 认证服务器 部署	250	17.2	Puppet 信息记录方式	278
15.3.3	Puppet LB 负载均衡器 部署	251	17.3	tagmail 发送邮件报告	279
15.3.4	Puppet Master 服务器 部署	252	17.4	rrdgraph 图形化报告	280
15.3.5	Puppet 客户端配置	254	17.5	自定义报告处理器	282
15.3.6	验证架构	254	17.6	本章小结	284
15.4	架构扩展之利用 Git 构建 分布式的 Puppet	254	第 18 章	必须了解的控制台	285
15.4.1	实现原理	255	18.1	Puppet DashBoard	285
15.4.2	安装与部署	256	18.1.1	简介	285
15.5	本章小结	259	18.1.2	DashBoard 安装	285
第 16 章	File 资源管理优化	260	18.1.3	配置 DashBoard	287
16.1	深入理解 File 资源	260	18.1.4	集成 DashBoard	292
16.2	操作实践	262	18.2	Foreman	297
16.3	File 资源配置方法	269	18.2.1	Foreman 简介	297
16.3.1	模块文件目录配置	269	18.2.2	安装 Foreman	298
16.3.2	统一文件目录配置	270	18.2.3	配置 Foreman	303
16.3.3	content 属性	271	18.2.4	使用 Foreman 管理 Puppet	311
16.4	File 资源的优化	271	18.2.5	从 Foreman 显示报告	313
16.4.1	配置 Nginx 代理	272	18.2.6	Foreman 其他功能	314
16.4.2	选择 File 资源还是 ERB	272	18.3	本章小结	314
16.4.3	大文件下发方法	272	第 19 章	Puppet 扩展工具	315
16.5	从 filebucket 检索文件	272	19.1	生成 HTML 文档	315
16.6	备份与恢复文件	275	19.1.1	利用 puppet doc 生成 HTML	316

19.1.2 puppet doc 的其他使用 方法.....	318	20.1.4 MCollective 给 Puppet 带来的改变.....	338
19.2 生成依赖关系图.....	319	20.2 消息中间件.....	339
19.2.1 什么是关系图.....	319	20.2.1 Stomp.....	339
19.2.2 配置方法.....	319	20.2.2 ActiveMQ.....	339
19.2.3 关系图说明	322	20.2.3 RabbitMQ.....	339
19.3 PuppetDB.....	322	20.3 标准化部署 MCollective.....	340
19.3.1 PuppetDB 功能与特性.....	322	20.3.1 体系结构与配置.....	340
19.3.2 安装 PuppetDB.....	324	20.3.2 安全模型.....	340
19.3.3 PuppetDB 配置文件详解.....	326	20.3.3 未来扩展.....	341
19.3.4 配置与使用 PuppetDB.....	329	20.4 部署 MCollective 步骤.....	341
19.3.5 PuppetDB 瓶颈.....	332	20.4.1 创建和收集证书.....	342
19.4 Hiera.....	335	20.4.2 部署和配置中间件.....	344
19.4.1 Hiera 的特点.....	335	20.4.3 MCollective 安装与配置.....	348
19.4.2 Hiera 的使用.....	335	20.5 如何使用 MCollective.....	355
19.5 本章小结.....	335	20.5.1 mco 基本命令的用法.....	355
第 20 章 MCollective 结合.....	336	20.5.2 执行 RPC 请求.....	356
20.1 MCollective 简介.....	336	20.5.3 过滤器的使用.....	358
20.1.1 什么是 MCollective.....	336	20.6 MCollective 使用 Shell Commands.....	359
20.1.2 MCollective 角色互换.....	337	20.7 MCollective 控制 Puppet.....	360
20.1.3 MCollective 的特点.....	338	20.8 本章小结.....	362

第一部分

准 备 篇

Puppet 可谓是配置管理工具中的旗舰产品，有别于 chef 等其他配置管理工具，它可以使系统管理员的工作变得更轻松，从而提升工作效率，并且只需要维护简单的关系就能掌握一切。使用 Puppet，我们可以从系统安装、配置管理、系统监控来实现运维体系的一体化、流水线化、产品化。通过简单的后台系统我们就可以完成所有的动作，并且能查看机器的进度与当前状态。是不是觉得这项工作非常让人振奋呢？那就让我们一起探究 Puppet 的奥秘吧！

本篇主要是为了更好地学习和使用 Puppet 而做的准备。我们通过第 1 章来了解 Puppet，看看它的发展历程，重点掌握它的工作原理；通过第 2 章对安装、配置 Puppet 的讲解进一步加深对 Puppet 工作原理的理解；通过第 3 章的实例，创建一个属于自己的配置，进而掌握基本的配置方法；最后通过第 4 章学习 Puppet 的几种工作环境，我们以后能够更好地利用 Puppet 实现不同环境的配置管理。

第 1 章

认识 Puppet

本章首先介绍什么是 Puppet，为什么它具有这么优秀的特征，以及它的发展历程。随后将 Puppet 和当前主流开源配置管理工具进行对比，阐述为什么要使用 Puppet。了解 Puppet 的组织结构并掌握 Puppet 的工作原理，这是本章的重点，希望读者多次阅读这部分内容，以便完全理解。Puppet 最新版本为 3.0（截至 2013 年 9 月），不同版本之间新的特性及是否可以混合使用，以及配置文件的相关知识，这些都是本章将要介绍的内容。

1.1 Puppet 的起源与发展现状

1.1.1 什么是 Puppet

官方的定义是这样的：Puppet 是一个开源的新一代的集中化配置管理工具，它由自己所声明的语言表达系统配置，通过客户端与服务端之间的连接，维护着关系库。Puppet 的设计目标是让 Puppet 成为一个由富有表现力的语言支撑的足够强大的库。这样只需要编写短短的几行代码的自动化应用程序即可实现设计目标。同时 Puppet 是开放的，允许添加任何新的功能。

通常这样定义：Puppet 是一个跨平台的集中化配置管理系统，它使用自有的描述语言，可管理配置文件、用户、Cron、软件包，系统服务等，Puppet 把这些统称为“资源”。Puppet 的设计目标就是简化对这些资源的管理以及妥善处理资源之间的依赖关系。

Puppet 是基于 Ruby 语言（<http://www.ruby-lang.org/>）并使用 Apache 协议授权的开源软件（在 Puppet 2.7.0 与 Facter 1.6.0 之前是基于 GPLv2 协议授权的），它既能以客户端 - 服务端（C/S）的方式运行，也能独立运行。客户端默认每 30 分钟会与服务端确认一次更新，以确保配置的一致性。

1.1.2 Puppet 起源与发展

Puppet 主要由 Luke Kanies 和他的公司 Puppet Labs 开发，于 2005 年正式面世。Luke

Kanies 于 1997 年就涉足 UNIX 和系统管理, 由于平常需要进行大量的开发工作, 在试用当时的配置管理软件不满意后, 他便想到自己开发一套工具来管理资源。他认为这套工具实现的关键是如何使每个资源成为一个合集, 每个资源又有自己的行为, 并且产生相应的行为动作。在 Luke Kanies 的努力下终于有了今天这款出色的 Puppet 产品。当然也感谢 Ruby 语言。

Puppet 于 2005 年面世, 发行版本也由 0.2 发展到了 3.0。Puppet Labs 的目标不只是把它当做配置工具, 还把它当做“拯救世界”的工具。特别是在近几年, 其发展势头迅猛。截至 2011 年 11 月 Puppet Labs 共融资 1600 万美元, 加上商业软件 Puppet Enterprise 的发布及对 Mcollective 软件的收购, 版本更新也更加迅速, 但 Puppet Labs 的目标不只是让版本发行更迅速, 还要极大地提升性能。我们可以看到 2.7 较之前版本在编译性能上提升了 60% 以上, 3.0 版本对 Ruby1.9 提供了完美支持、Master 在 CPU 消耗上提升了 6 倍的性能、提供了更多发行版操作系统的支持。特别是在 2012 年 5 月, Puppet Labs 率先与 OpenStack 整合, 这非常振奋人心。由此可见, Puppet Labs 的前景不可估量。

Puppet 从 0.25.x 直接跳到大版本 2.6 且逐渐不再维护 0.25.x, 在 2010 年时通过 Yum 安装的 Puppet 还是 0.25.x 版本。有人会问 2.6 就比 0.25.x 性能好吗? 其实不然, 2.6 主要是在功能上的改进, 为此 Puppet Labs 改变了版本命令方式, 增加了新的特性并取消了 XML-RPC 传输层, 除此之外没有其他大的改动。因此我们可以理解为, 只是命令方式的变更。

虽然目前 Puppet 在短短 2 年内由 2.6 发展至 3.0, 从 3.0.0 版本开始, Puppet 使用的是严格的 3 段版本号, 最左边为大版本号 (功能向后兼容), 中间为新功能变化, 最右为修复 bug。伴随着版本的更新官方文档也在快速更新当中, 这为我们学习提供了很大的便捷, 也增强了我们掌握 Puppet 的信心。

由于 Puppet 0.2x 支持的特性非常少, 不建议再安装使用它。对于正在使用此版本的用户, 建议升级成最新版。对于新用户, 建议直接安装 Puppet 3.0 版本。对于正在使用 Puppet 2.6 或 2.7 的用户, 建设逐步升级, 且先升级 Master, 然后再升级 Agent, 平滑过渡, 以避免遇到不可预知的问题。



提示

更多 Puppet 版本信息可参考: http://projects.puppetlabs.com/projects/1/wiki/Release_Notes。

1.1.3 版本语言特征

Puppet 各版本之间的语言也存在着差异, 当然, 版本越高, 支持的特征越多, 具体如表 1-1 所示。

表 1-1 Puppet 各版本语言特征差异对比

语法	0.24.x	0.25.x	2.6.x	2.7.x	3.x
运算符	支持	支持	支持	支持	支持

(续)

语法	0.24.x	0.25.x	2.6.x	2.7.x	3.x
多资源的关系	支持	支持	支持	支持	支持
类的继承	支持	支持	支持	支持	支持
追加变量	支持	支持	支持	支持	支持
类命名	支持	支持	支持	支持	支持
C 注释风格	支持	支持	支持	支持	支持
节点正则	不支持	支持	支持	支持	支持
变量表达式	不支持	支持	支持	支持	支持
正则表达式	不支持	支持	支持	支持	支持
条件表达式	不支持	不支持	支持	支持	支持
链接资源	不支持	不支持	支持	支持	支持
哈希	不支持	不支持	支持	支持	支持
类的参数化	不支持	不支持	支持	支持	支持
运行阶段	不支持	不支持	支持	支持	支持
In 语法	不支持	不支持	支持	支持	支持
Unless 语法	不支持	不支持	不支持	不支持	支持

1.1.4 命令差异

Puppet 命令在 2.6 版本发布时进行了变更，且在发布 3.0 版本时将 2.6 版本之前的旧命令完全丢弃。具体差异对比如表 1-2 所示。

表 1-2 不同版本 Puppet 的命令差异对比

2.6 版本之前	2.6 版本之后
Puppetmasterd	Puppet master
Puppetd	Puppet apply
Puppetca	Puppet cert
Ralsh	Puppet resource
Puppetrun	Puppet kick
Puppetqd	Puppet queue
Filebucket	Puppet filebucket
Puppetdoc	Puppet doc
Pi	Puppet describe

1.1.5 Puppet 3.0 新特性

目前 Puppet 最新版为 3.0，其中增添了许多新特性。

- 性能提升：目录编译采用 JSON 作为目录缓存。
- 增强操作系统与平台支持：对 Ruby1.9 的完美支持，对操作系统 Windows、Solaris 包和服务的更多支持，Yumrepo 对 SSL 的支持。

- ❑ 使用 Rubygems 加载插件：可以通过 Rubygems 来安装和使用 Puppet 的扩展代码。
- ❑ Server 自动发现：通过 DNS SRV 寻找 CA、Master、Report 和 Fileserver。
- ❑ DSL/config 变化：auth.conf 增加 allow_ip 配置选项，unless 支持，插件同步默认改为 True，更新 configure 的语法。
- ❑ 其他 BUG 的修复：修复了 3.0 Agent 与 2.7Master 工作的问题、kick 无法工作问题、rack 安装启动出错问题等。



注意

Puppet 3.0 将不再支持 Ruby 1.8.5 以下版本。

1.2 为什么要使用 Puppet

1.2.1 都有谁在使用 Puppet

在笔者编写本书时，Puppet 已拥有 250 家客户，包括 Zynga、Twitter、纽约证券交易所、迪斯尼、Citrix、Oracle/Sun、Constant Contact、Match.com、Shopzilla、Google、RedHat 等。国内越来越多的大公司也在使用 Puppet，例如新浪、阿里巴巴、百度、腾讯、奇虎 360、小米、United Stack、豆瓣、好乐买、趣游、PPTV 等。

1.2.2 常见集中化管理工具对比

目前开源的工具非常多，很多人在选择的时候犹豫不定，不知道哪个好。如果担心在使用过程中遇到问题没有人回答，资料太少而无从查起，那么选择当前最流行的工具无疑是最明智的。现在毫无疑问，应该选 Puppet。

针对当前开源的配置管理工具，这里进行简单的汇总：Puppet、Chef、Func、Fabric、Capistrano、Cfengine 等。下面我们就最常用的 Puppet 与 Chef 进行简单对比，主要是从用户、开发、平台、实例等几个角度出发，帮助读者从中发现 Puppet 的优势。在用户与第三方支持方面，Puppet 更胜于 Chef。详细对比如表 1-3 所示。

表 1-3 Puppet 和 Chef 对比

	Puppet	Chef
使用用户	Google、RedHat、Apple、Sina、…	Admeld…
开发支持	第三方 Foreman	无
商业运行	Enterprise	Enterprise
程序语言	Php、Django、Ruby、…	Ruby
文档环境	Mail-List、IRC	Mail-List、IRC
平台支持	ALL os	较多
现成实例	Example42	无
依赖关系	灵活处理	无

**提示**

Puppet 所支持的操作系统：<http://docs.puppetlabs.com/guides/platforms.html>

Chef 所支持的操作系统：<http://wiki.opscode.com/display/chef/Installing+Chef+Server> Chef

使用者：<http://www.opscode.com/customers/> Puppet

使用者：http://projects.puppetlabs.com/projects/1/wiki/Whos_Using_Puppet

1.2.3 推荐 Puppet 的理由

有很多人问笔者：学习 Puppet 是否需要具有开发基础？是否需要会 Ruby？笔者的回答都是“不需要”。其实我们使用任何一个工具，都不需要掌握此工具所采用的语言，只要会使用工具即可。如何更深入地理解工具的特性，最大化地发挥出它的优势才是我们的首要任务，除非想在它的基础上做二次开发。通常做二次开发都有现成的框架，调用工具的 API 接口即可完成。因此我们不需要掌握这类开发语言。而且开发类语言都互通的，掌握一门，其他的学起来也不难。初学者在入门时一定要意识到这一点：明确自己的目的，而不要被工具本身吓住了，“Just Do It”。

1.3 Puppet 的作用和特色

1.3.1 为什么要有自己的语言

为什么不使用 XML or YAML 配置格式？为什么不直接采用 Ruby 输出？Puppet 开发者非常巧妙地用一个反问回答了这个问题：在使用浏览器访问网站的时候为什么不直接读 HTML 呢？所以说 Puppet 使用自有语言是为了更好地处理人机接口。而 Ruby 输出实际上在 Puppet2.6 版本中已经支持，只是没有当做主输出而已。

1.3.2 为什么是 Ruby

用 Kanies 的话来说，Ruby 太适合 Puppet 了。最开始 Kanies 的大量工作都是使用 Perl 语言来完成的，然而当他想要开发工具时，发现在 Perl 中得不到想要的类关系。于是试用了一下 Python，虽然很多人都说它非常棒，但也满足不了需求。这时有人说 Ruby 很酷，Kanies 进行了尝试，没想到 4 个小时就做出了所需工具的原型。为此直到后来为 Puppet Labs 选择语言时，Kanies 一直也没有后悔选择了 Ruby。因此选择适合自己的语言才是硬道理。我们不能一味地追求时尚，就像我们选择 Puppet 一样。

1.3.3 管理任何机器

目前 Puppet 支持所有的客户端，主流的有 RedHat、Centos、Gentoo、FreeBSD、Debian、OpenBSD、Mac os x、Ubuntu、SuSE、Solaris、Windows 等。我们不需要担心所管理的设备无法使用 Puppet，它已经尽可能地支持一切操作系统。不过使用 Windows 的朋友需要注意：

目前 Puppet 所支持的资源有 File、Package、Host、Group、Service、Exec，支持的类型有限。不过刚刚发布的 Puppet3.0 对 Windows 的支持进行了加强。



提示

更多信息可参考 <http://docs.puppetlabs.com/windows/writing.html>。

处理资源与资源之间的依赖关系是 Puppet 的优势。Puppet 管理一台主机的整个生命周期，即初始化安装、升级、维护、服务迁移及下载。在 Puppet 世界中，一台主机的每个生命周期内的每个动作都被抽象成一个“资源”。我们更多的是要维护一台主机上的每个“资源”，梳理好关系后将其交给 Puppet 维护。对于系统管理员而言，一切的配置将变得简单而有趣。后续我们将在第 6 章讲解 Puppet 的资源。

1.4 Puppet 组织结构

在了解了什么是 Puppet 后我们再来看它的组织结构，以便在后续章节中能更好地掌握其工作原理。

通常在安装好 Puppet 之后的 /etc/puppet 目录下运行 tree 就能看到下面的树结构：

```
|-- auth.conf           #ACL 权限控制文件
|-- fileserver.conf     # 文件服务配置文件
|-- manifests          # 节点存储目录 (Puppet 会首先加载 site.pp)
|   |-- site.pp        # 定义 Puppet 变量和默认配置
|-- modules            # 模块配置目录
|   |-- nginx          # 以 Nginx 为例
|       |-- manifests
|       |   |-- init.pp # 模块主配置文件，定义类 class 相关信息。读取模块后先读取它
|       |   |-- templates
|       |       |-- nginx.conf.erb # 模板配置文件 (erb 为主)
|-- namespaceauth.conf # 命名空间配置文件 (配置权限)
|-- puppet.conf        #Puppet 主配置文件
|-- tagmail.conf       # 邮件报告配置文件
```



注意

不同安装包的树结构会不一样，为了便于读者理解，这里仅对主要的配置文件进行了说明。

1.5 Puppet 工作原理

下面进入本章重点——Puppet 工作原理，这部分将分别从基本结构、Puppet 是如何工作的、数据流、文件结构、详细交互过程等几方面循序渐进地讲解，以便读者理解并掌握这些知识。

1.5.1 Puppet 基本结构

我们先看一下 Puppet 基本结构，如图 1-1 所示。Puppet 模块的编写主要是：应用程序、

文件、服务及操作系统底层相关。客户端解析完成后发送给报告系统（见图 1-1 右上角）。

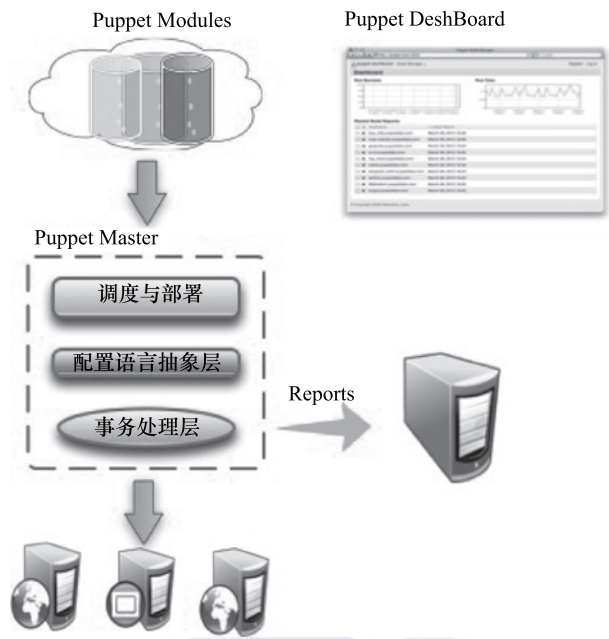


图 1-1 Puppet 基本结构

1.5.2 Puppet 是如何工作的

- 接着我们来看一下 Puppet 是如何工作的。简单来说分 4 步进行。
- 1) 定义：使用 Puppet 特定的语言定义基础配置信息。通常我们把这些信息写在 Modules 中。
 - 2) 模拟：在配置执行之前检测代码，但并不真正执行。
 - 3) 执行：按步骤 1) 定义的配置自动部署。检测并记录下所发生变化的部分。
 - 4) 报告：将期待的变化、实际发生的变化及任何修改发送给报告系统。
- 整个工作过程如图 1-2 所示。

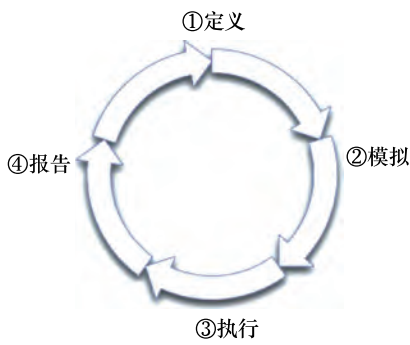


图 1-2 Puppet 工作过程

1.5.3 Puppet 数据流

在了解了 Puppet 如何工作之后，我们需要继续了解它的数据流走向。

- 1) Node 节点将 Facts 和本机信息发送给 Master。
- 2) Master 告诉 Node 节点应该如何配置，将这些信息写入 Catalog 后传给 Node。
- 3) Node 节点在本机进行代码解析验证并执行，将结果反馈给 Master。

4) Master 通过 API 将数据发给分析工具。报告完全可以通过开放 API 或与其他系统集成。

整个数据流的走向是基于 SSL 安全协议的, 如图 1-3 所示。

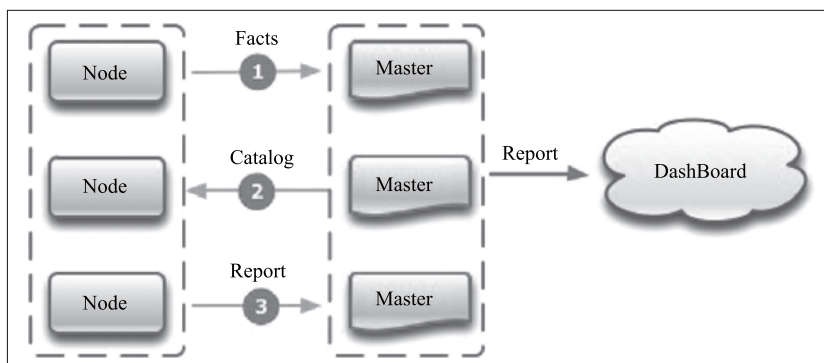


图 1-3 Puppet 数据流

1.5.4 文件结合

在 Puppet 组织结构中我们了解了 Puppet 的树结构。那么结合数据流, 这些目录的关系是如何关联起来的呢?

1) Puppet 通过编译 Manifest 中的内容, 将编译好的代码存入 Catalog。

2) 在执行前先进行代码的验证, 再执行, 完成最开始所定义好的状态。

代码编译过程如图 1-4 所示。

1.5.5 详细交互过程

通过以上的学习, 相信大家应该对 Puppet 的工作原理有了基本的了解, 接下来我们将分析 Puppet 的 Agent 与 Master 的详细交互过程。通过学习这部分内容, 可加深对 Puppet 的理解, 以便在今后的使用过程中遇到任何问题都能在大脑中呈现如图 1-5 所示的内容, 进而能快速定位故障并解决它。所有交互过程都是建立在签发证书的前提下执行的。

1) Puppet 客户端 Agent 将节点名与 facts 信息发送给 Master。

2) Puppet 服务端 Master 通过分类判断请求的客户端是谁, 它将要做什么。这个判断是

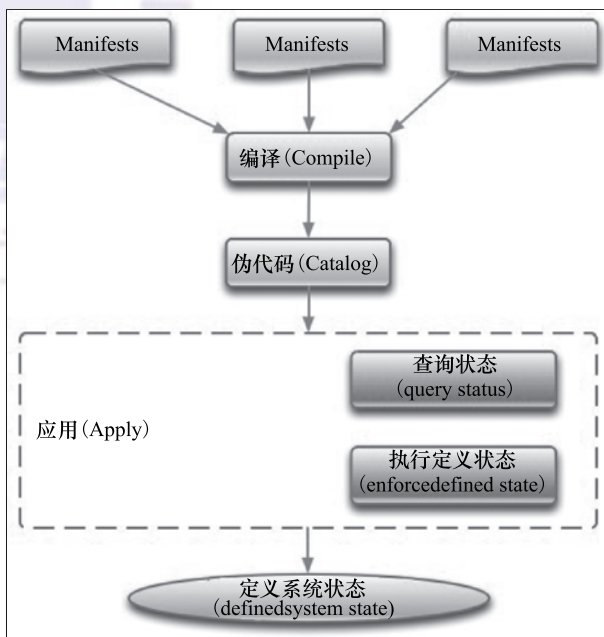


图 1-4 文件编译过程

通过 site.pp 中包含的 Node.pp 配置文件定义的。

3) Puppet 服务端 Master 将所需要的 Class 类信息进行编译后存入 Catalog 并发送给 Puppet 客户端 Agent, 到此完成第一次交互。这一步是 1.5.4 节的内容。

4) Puppet 客户端 Agent 对 Catalog 进行代码验证 (语法检查及错误检查) 并执行。主要是代码的验证, 并将执行过程的信息及结果写入日志。

5) Puppet 客户端 Agent 最终达到最开始所定义的状态, 并且将结果及任何执行数据通过开放 API 的形式发送给 Puppet 服务端 Master。

Puppet Master 和 Puppet Agent 的交互过程如图 1-5 所示。

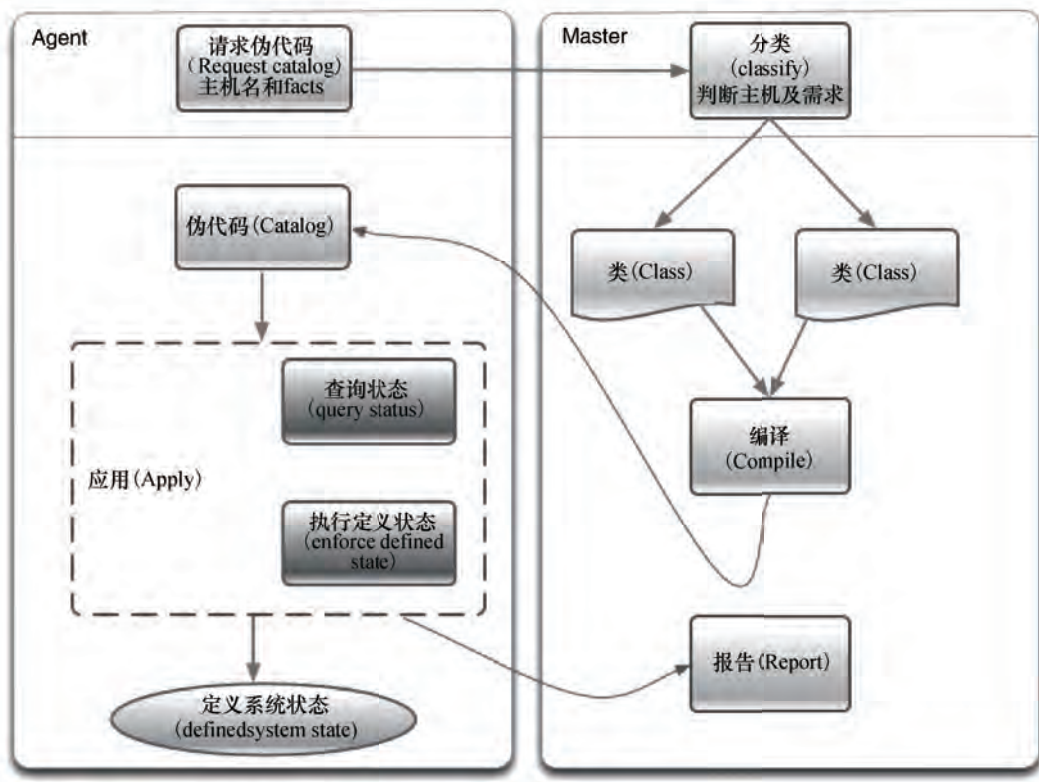


图 1-5 Puppet Master 与 Puppet Agent 详细交互过程

1.5.6 安全与认证

Puppet 通信都采用 SSL 安全加密协议, 以保障所有数据传输的安全性, 为此我们不用担心数据的安全性问题。通常我们使用 Puppet 管理设备时也有可能建立在公司内网的基础之上, 这样就更加安全了。当然, 即使你所在的公司没有属于自己的内网也没有关系, Puppet 所采用的 SSL 安全加密协议已经为我们解决了数据传输的安全问题。



提示

SSL (Secure Sockets Layer, 安全套接层) 及其继任者 TLS (Transport Layer Security, 传输层安全) 是为网络通信提供安全及数据完整性的一种安全协议。TLS 与 SSL 在传输层对网络连接进行加密。

在学习了 Puppet 安全之后, 我们来学习 Puppet 的认证。由于 Puppet 采用的是 SSL 加密协议, 因此申请证书验证是必需的。

- ❑ Puppet Master 在启动后会向自己签发证书和 key。我们可以在 `/var/puppet/ssl` (3.1 版本在 `/var/lib/puppet/ssl/` 下面) 目录下看到它们。
- ❑ Puppet Agent 在运行 `puppet apply --test` 时添加参数 (`--verbose`), 可以在客户端终端看到申请证书的详细过程。
- ❑ Puppet Master 同样也可以使用 `puppet cert list` 查看申请证书的客户端列表。使用命令 `puppet cert sign agent_name` 来签发证书。
- ❑ Puppet Agent 就可以收到 `notice:Finished` 等相关字样。如果 Master 一直不签发证书, 客户端会每 2 分钟请求一次。

证书相关的命令将会在第 5 章详细讲解。

Puppet 只允许经过安全认证的客户端发送请求, 这极大地保证了数据的安全。同时当客户端发起 `https://master.domain.com:8140/{environment}/{resource}/{key}` 这样的请求时, 我们需要配置 `auth.conf` 与 `namespaceauth.conf`。有关这两个配置文件详见 1.6 节。



思考

当客户端要输出 “Hello World!” 时, 整个 Puppet 的数据流走向?

1.6 Puppet 核心配置文件详解

在学习了 Puppet 原理及版本差异后, 我们需要掌握它的核心配置文件。Puppet 所有的配置都围绕着 `Puppet.conf` 展开。读者可以回顾在 1.4 节讲的组织结构, 以便快速掌握本节内容。

1.6.1 主配置文件 puppet.conf

主配置文件 `puppet.conf` 主要用于设置相关的参数、认证文件、文件系统配置文件、插件配置等。Puppet 版本不同, 配置选项命名方式也有所不同。Puppet 2.6 以前的命名方式还以 `[puppetmaster]`、`[puppetagent]` 为主。Puppet 2.6 以后都以 `[main]`、`[master]`、`[agent]` 为主。为了防止读者混淆, 建议都采用 Puppet 2.6 以上版本。本书都以 Puppet 2.6 以上版本为主。

先看如何生成 Puppet 配置文件 `puppet.conf`:

```
puppetd --genconfig> /etc/puppet/puppet.conf
puppetmasterd--genconfig> /etc/puppet/puppet.conf
```


以上命令会覆盖 Puppet.conf 原文件。

在本地用如下命令查看配置参考手册：

```
puppet doc --reference configuration
```

如果不知道配置文件在哪个目录下，可以使用以下命令查看：

```
puppet agent --configprint confdir
```

默认目录是 /etc/puppet。

由于 puppet.conf 配置文件内容较多，下面笔者将列举核心配置、常用配置选项（不区分 Master 与 Agent）：

```
[main] # 通用配置选项
vardir = /var/lib/puppet
confdir = /etc/puppet
logdir = /var/log/puppet
rundir = /var/run/puppet
ssldir = $vardir/ssl
filesERVERconfig = /etc/puppet/filesERVER.conf
manifestdir = /etc/puppet/manifests # 可不指定默认读取此目录
manifest = /etc/puppet/manifests/site.pp # 主机文件默认读取
modulepath = /etc/puppet/modules:/usr/share/puppet/modules
authconfig = /etc/puppet/namespaceauth.conf
# 如果开启 Listen 为 true 需要配置此文件
pluginsync = true # 插件同步配置对 facter 自定义有效
reportdir = /var/lib/puppet/reports
# 报告文件生成目录，目录以主机名命令开头
reports = log, foreman # 报告的方式与类型
# environment = production 运行环境配置，默认为生产环境

[agent] # 客户端配置选项
classfile = $vardir/classes.txt
localconfig = $vardir/localconfig
runinterval = 1800 # 客户端默认探测时间，可按需求修改
listen = true # 是否监听，执行 puppet kick 时需要配置
report = true # 客户端的报告系统配置，不同于 Master
# 此项的主要目的是将报告发送至 Master，主要用于客户端 puppet.conf 配置
report_port = 8140 # 监听端口，如果服务器配置有防火墙，需开放此端口
report_server = server # 默认不填，此时以下面的 $server 变量值为准
server = server.domain.com

[master] # 服务端配置选项
certname = server.domain.com # 也可以不定义，以主机名为准
reports = store, http, tagmail, log
reporturl = http://server.domain.com:3000/reports/upload
# 报告发送地址，可配置在 dashboard 或 foreman 配置文件中
# 有关报告系统的配置我们将在第 17 章讲解
autosign = /etc/puppet/autosign.conf # 自动认证配置文件
```



提示

不同版本的配置文件可参考：<http://docs.puppetlabs.com/references/>。

通常我们应该如何配置主配置文件呢？下面分别进行介绍。

Puppet Master 服务器端主配置文件——puppet.conf。

```
[main] # 全局配置
#vardir 用来存放缓存数据、配置、客户端传回的报告及文件备份
vardir = /var/lib/puppet
# Puppet 日志文件配置目录默认为 '$vardir/log'
logdir = /var/log/puppet
#PID 文件目录，默认为 '$vardir/run'
rundir = /var/run/puppet
#SSL 签发认证文件目录，默认为 '$confdir/ssl'
ssldir = $vardir/ssl

[master] # 服务器端配置
pluginsync = true                # 开启插件同步
reports = log, foreman          # 报告发送至 log 和 foreman
environment = production        # 指定运行环境为生产环境
certname = server.domain.com    # 配置主机名为 server.domain.com
```

Puppet Agent 客户端主配置文件——puppet.conf。

```
[main]
logdir = /var/log/puppet
rundir = /var/run/puppet
ssldir = $vardir/ssl
pluginsync = true

[agent] # 客户端配置
# 关联与检索配置文件目录，默认为 '$confdir/classes.txt'，也可以使用
# (--loadclasses) 参数指定
classfile = $vardir/classes.txt

# 本地缓存配置目录，默认为 '$confdir/localconfig'
localconfig = $vardir/localconfig
runinterval = 1800                # 检测时长 1800 秒，30 分钟
listen = true                     # 监听进程
report = true                     # 发送报告
server = puppet.domain.com        # 指定 Master 地址
```

1.6.2 主机配置文件 site.pp

site.pp 的目的主要是告诉 Puppet 去哪里寻找并载入所有主机相关的配置。site.pp 默认存放在 /etc/puppet/manifests 目录中。通常我们会在会此文件中定义一些全局变量。



注意

“清单”（Manifests）是 Puppet 中的术语，指的是包含配置信息的文件。清单文件的后缀是 .pp。

生成 Puppet 主机配置文件 `site.pp` 的命令如下：

```
puppet apply --genmanifest> /etc/puppet/manifests/site.pp
```

通常不使用以上命令生成 `site.pp` 配置文件，主要填写的内容为：

```
# site.pp - all agent configure
Exec { path => "/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin" }
# 设置环境变量
$fileservers = "puppet.domain.com" # 指定全局 fileservers 变量
$ntpserver = "ntp.domain.com" # 指定 ntpserver 变量
# 建议读者动手配置以上内容

Package { provider => "yum" } # 指定软件包的安装方法为 yum
# 可以直接写节点配置文件，在所有 Agent 上创建其内容为 "Hello World" 的
# /tmp/templ.txt 文件
#node default {
#   file { ["/tmp/templ.txt": content => "Hello World!"; ]
# }

import "modules.pp" # 加载模块配置文件，可以不配置
import "nodes/*.pp" # 加载主机信息，可以使用通配符，也可以定义多组目录
import "test.domain.com.pp" # 加载测试主机
```

1.6.3 认证与安全配置文件

`namespaceauth.conf` 文件用于指定允许谁访问每个名称空间。要创建 `namespaceauth.conf` 文件必须添加对每个名称空间的访问权限。以下就是针对不同名称空间的权限配置，通常不用做大的变动。

以下是配置的内容：

```
[fileservers]
allow *.domain.com

[puppetmaster]
allow *.domain.com

[puppetrunner]
allow culain.domain.com

[puppetbucket]
allow *.domain.com

[puppetreports]
allow *.domain.com

[resource]
allow server.domain.com
```



注意

当 `puppet.conf` 配置 `Listen=true` 时必须配置此文件，否则 Puppet 启动时会报出 “Will

not start without authorization file /etc/puppet/namespaceauth.conf” 的错误提示。

auth.conf 认证配置文件是 Puppet 中的 ACL，主要应用于 Puppet's REST API。例如，当客户端请求 `https://yourpuppetmaster:8140/{environment}/{resource}/{key}` 时，服务器可以根据资源定义访问权限。其配置的参考如下：

```
# 以 auth.conf 为例

path /
auth any
environment override
allow magpie.example.com

path /certificate_status
auth any
environment production
allow magpie.example.com

path /facts
method save
auth any
allow magpie.example.com

path /facts
auth yes
method find, search
allow magpie.example.com, dashboard.example.com
```

只看上面 auth.conf 配置文件，读者可能难以理解配置文件的意义，参考 Puppet ACL 配置语法就容易理解了。其语法规则如下：

```
path [~] {/path/to/resource|regex}      # 目录配置
[environment {list of environments}]    # 环境配置
[method {list of methods}]              # 方法命令配置
[auth[enticated] {yes|no|on|off|any}]    # 授权配置
[allow {hostname|certname|*}]            # 允许配置
```

我们再来看 auth.conf 的配置文件：

```
path /facts
auth yes
method find, search
allow magpie.example.com, dashboard.example.com
# 现在就不难理解这段配置的意思了。允许主机名为 "magpie.example.com" 和
#"dashboard.example.comd" 的两台客户端在 facts 目录进行 find 和 search 操作
```



提示

详细信息可参考：http://docs.puppetlabs.com/guides/rest_auth_conf.html。

1.6.4 客户端自动认证配置

`autosign.conf` 允许配置文件中的客户端自动进行签名验证，省去人工交互的过程。这对自动化配置来说省去了很多工作。在下一章介绍安装时也会讲到。

配置文件参考如下：

```
# 允许单一客户端或者域名匹配，主机名匹配
rebuilt.example.com
*.scratch.example.com
*.local
```

同时 Puppet 客户端的证书也可以采用提前在 Master 上生成的方法，将生成的证书文件复制到客户端对应目录下，以实现自动认证的配置。

自动生成证书的命令为：

```
puppet cert generate client.domain.com
```

这时将在本地生成 `client.domain.com` 客户端证书文件，文件及其所在目录为：

```
/var/lib/puppet/ssl/private_keys/client.domain.com.pem
/var/lib/puppet/ssl/certs/client.domain.com.pem
/var/lib/puppet/ssl/certs/ca.pem
```

将这些文件复制到客户端相同的目录下即可。

1.6.5 报告系统配置

`tagmail.conf` 将配置的 Puppet 报告以邮件形式按要求发送给收件人。要使用此项功能，需要在 Puppet Master 端配置 `reports=tagmail`，需要在 Puppet Agent 端配置 `report=true`，同时也可以配置 `smtp` 由谁来发送。在 Puppet Master 端配置 `reportform` 为 `smtpserver` or `sendmail` 即可。默认由本机 `sendmail` 发送。

```
all: log-archive@example.com
webserver, !mailserver: httpadmins@example.com
emerg, crit: james@example.com, zach@example.com, ben@example.com
# 日志的级别可以是: debug、info、notice、warning、err、alert、emerg、crit 或者 verbose
# 也可以是一个类的名字 class names, 一个标签名 tags
```

1.6.6 文件系统配置文件

`fileserv.conf` 是一项安全配置，结合 `puppet.conf`、`auth.conf` 一起使用。配置 Puppet 客户端允许 / 禁止访问 Master 的文件目录。`fileserv.conf` 的使用非常灵活，我们将在第 16 章节来讲解它的优化及配置。`fileserv.conf` 配置如下：

```
# 配置方法
[mount_point]
    path /path/to/files
```

```
allow *.example.com
deny *.wireless.example.com
# 举例：允许通配主机为 "*.domain.com" 的主机获取 /etc/puppet/files 目录下
# 的文件。以下为其配置方法
[files]
    path /etc/puppet/files
    allow *.domain.com
```

1.7 本章小结

通过对本章的学习，我们首先了解了 Puppet 的发展历程、各版本存在的差异、Puppet 2.6 版本的大幅度变动、Puppet 3.0 版本的最新特征。接着通过学习 Puppet 组织结构、Puppet 的工作原理、核心配置文件的相关知识，使读者能够熟练掌握 Puppet 的基本用法和工作原理。

如果你没有了解过 Puppet，通过本章便能大致了解 Puppet 是什么、它有什么用途、是否适合在你的环境当中使用。如果你用过 Puppet 并有使用经验，通过本章的学习能够更深入地掌握它的原理。

将 Puppet 原理及核心信息放在第1章来讲解，是为了让大家能更全面地了解 Puppet，毕竟任何一款软件的使用都离不开基础原理的掌握。

接下来一章将介绍 Puppet 在不同操作系统平台的安装与使用。

第 2 章

Puppet 安装与配置

第 1 章介绍了什么是 Puppet，以及 Puppet 的作用及优势。上一章我们通过学习 Puppet 的组织结构、工作原理、工作流程、数据流走向帮助读者逐步掌握 Puppet。从本章起我们将真正开始使用 Puppet。

本章首先介绍 Puppet 对各操作系统平台的支持及对 Ruby 语言的支持，并简述 Puppet 的安装步骤。其次讲解 Puppet 在主流 Linux 操作系统平台的安装方法及被称为“最完美操作系统”的 Mac 平台上的安装。接着详细讲解 Puppet 在主流 Linux、Mac 平台上的使用方法，并创建 File 资源来应用实践。然后介绍在 Windows 平台的安装方法、使用方法和支持。最后介绍如何升级 Puppet 及升级时的注意事项。

2.1 Puppet 对各系统平台的支持

我们下面就看看 Puppet 主流版本（2.6、2.7、3.0）对各操作系统平台的支持。

（1）Linux

Puppet 主流版本支持的 Linux 版本如下：

- ☐ RedHat Enterprise Linux 4 及更高版本
- ☐ CentOS version 4 及更高版本
- ☐ Scientific Linux 4 及更高版本
- ☐ Oracle Linux 4 及更高版本
- ☐ Debian version 5 (Lenny) 及更高版本
- ☐ Ubuntu, version 8.04 LTS 及更高版本
- ☐ Fedora 15 及更高版本
- ☐ SUSE Linux Enterprise Server 11 及更高版本
- ☐ Gentoo Linux
- ☐ Mandriva Corporate Server 4

❑ ArchLinux

(2) BSD

Puppet 主流版本支持的 BSD 版本如下：

❑ FreeBSD 4.7 及更高版本

❑ OpenBSD 4.1 及更高版本

(3) Other UNIX

Puppet 主流版本支持的 Other UNIX 版本如下：

❑ Mac OS X 10.5 及更高版本（Puppet 2.7 以前的版本只支持 Mac OS X 10.4）

❑ Oracle Solaris 10 及更高版本

❑ AIX 5.3 及更高版本

❑ HP-UX

(4) Windows

Puppet 主流版本支持的 Windows 版本如下：

❑ Windows Server 2003 和 Windows Server 2008（Puppet 2.7.6 及更高版本）

❑ Windows 7（Puppet 2.7.6 以上更高的版本）

2.2 Puppet 对 Ruby 的支持

在安装 Puppet 前，首先要确认已经正确安装了 Ruby。Puppet 可以运行在 Ruby 不同版本之上，并且已经支持最新版 Ruby1.9。介于 Puppet 对 Ruby1.8.7 的完美支持，建议读者安装 Ruby 1.8.7 版本。有关 Puppet 与 Ruby 的版本对应关系如表 2-1 所示。

表 2-1 Puppet 与 Ruby 版本对应关系

Ruby version	Puppet 2.6	Puppet 2.7	Puppet 3.0
1.8.5*	支持	支持	不支持
1.8.7	支持	支持	支持
1.9.3	不支持	不支持	支持
1.9.0/1/2	不支持	不支持	不支持
1.8.1/6	不支持	不支持	不支持

2.3 Puppet 的安装步骤

Puppet 的安装可以细分 8 步进行，详细如下：

- 1) 安装 Ruby、Ruby-libs 和 Ruby-shadow，用于进行用户和组管理。
- 2) 安装 Facter、Puppet 和 Puppet-server。
- 3) 设置主机名、域名解析或指定 hosts。
- 4) 通过命令 `/etc/init.d/puppetmaster start` 启动 Server 或者以非进程的方式启动 Server。

通过命令 `puppet master --no-daemonize --verbose` 可以查看到相关日志与输出。

5) 在客户端配置文件 `Puppet.conf` 中指定 Server 端, 在终端运行 `puppet agent -test` 命令或直接运行 `puppet agent --test --server server.domain.com` 与 Master 交互完成签名认证。

6) 在 Server 上配置节点信息, 告诉客户端要做什么。

7) 检查语法是否正确 (通常采用 `puppet parser validate test.pp` 命令进行语法检查)。

8) 客户端再次运行配置 (`puppet agent --test`)。

根据以上步骤我们可以看到, 安装 Puppet 其实非常简单, 较为复杂的是配置过程。通过第 1 章的学习, 我们掌握了 Puppet 的原理, 再结合以上 8 步, 我们再安装 Puppet 将变得非常容易。

接下来我们学习 Puppet 在各操作系统平台下的安装及配置方法。



Factor (或者称为 fact) 用于收集关于主机的信息, 来帮助定制 Puppet 配置。

2.4 在 Linux 下安装

在 Linux 下安装 Puppet 是非常容易的事, 特别是在 Centos 操作系统下, 直接配置官方源即可完成所有包的安装。安装的时候会自动安装 Ruby 相关依赖及其相关组件。

对于各平台系统, 建议使用官方包管理方式对 Puppet 进行安装:

1) FreeBSD、OpenBSD 可以使用 `ports` 命令进行安装。

2) Gentoo 使用 `Portage` 命令进行安装。

3) 对于 SUSE, 在 <http://software.opensuse.org/122/en> 处可以搜索到软件包。

4) Solaris 使用 `Blastwave` 命令进行安装。

接下来本节主要讲解 Puppet 在 RHEL、Centos、Ubuntu&Debian、Fedora、Windows 系统上通过包管理器、源码及 Gems 工具来进行安装与使用。在编写本书的时候, Puppet 的最新版本是 3.0.x。Puppet 3 并不完全兼容 2.x 以下的低版本, 而且比 Puppet 2.x 性能更好, 在编译 catalog 时效率更高。对于首次接触 Puppet 的读者来说, 推荐使用 Puppet 3。如果之前一直在使用 Puppet, 而现在正犹豫是否要升级, 那就大可不必担心, 尽快尝试一下 Puppet 3 吧。



Puppet 3 为 Puppet 3.x.x 版本的简写。Puppet 2 为 Puppet 2.x.x 版本的简写。为了阅读方便, 本书统称 Puppet 3.xx 和 Puppet 3.0.x 为 Puppet 3。



Puppet 3 的新特征参考 1.1.3 节。升级 Puppet 时需要保持 Master 版本高于 Agent 版本, 升级可参考 2.8 节。

2.4.1 包管理器方式安装

用 Linux 发行版的包管理器来安装 Puppet 最为简单，而且会自动处理好所有依赖。这里安装的包也是最新版本的 3.0，如果需要安装较早的版本，就需要在安装时指定。笔者推荐读者采用 Linux 发行版的包管理器进行安装。

(1) 在 RHEL 5 和 RHEL 6、Centos 5 和 Centos 6 上安装 Puppet

使用 EPEL 源 (<http://fedoraproject.org/wiki/EPEL>) 或 PuppetLabs 源 (<http://yum.puppetlabs.com/>) 直接安装。注意选择自己系统版本的 Yum 源。

提示

Yum (全称为 Yellow dog Updater, Modified) 是一个在 Fedora、RedHat、SUSE 及 CentOS 中的 Shell 前端软件包管理器。基于 RPM 包管理，能够从指定的服务器自动下载 RPM 包并进行安装，可以自动处理依赖性关系，并且一次安装所有依赖的软件包，无须繁琐地多次下载和安装。

下面介绍具体的安装过程。

在安装之前需要确认自己的 Ruby 版本。如果系统并没有安装 Ruby，可以采用如下方法进行安装：

```
$ sudo yum -y install ruby ruby-libs ruby-shadow
# 如果系统默认 Yum 源里没有 Ruby 及相关软件包，可以配置 EPEL 源进行安装
```

提示

CentOS 5.x 中自带的 Ruby 版本是 1.8.5，而在 Puppet 3.0 发布之后，官方源中增加了 CentOS 5 的 Ruby 1.8.7 的 RPM 包，并且不再支持 1.8.5。所以建议在安装官方 Puppet 源后再安装或升级 Ruby 一次，或者直接采用官方 Puppet 源进行安装。

运行如下命令来检查 Ruby 的安装：

```
$ ruby -v
ruby 1.8.7
```

1) 根据操作系统版本配置 Yum 源，我们这边使用的操作系统为 Centos 6 x64，因此选择 puppetlabs-release-6-6 软件包。配置方法如下：

```
$ sudo rpm -Uvh http://yum.puppetlabs.com/el/6Server/products/x86_64/puppetlabs-release-6-6.noarch.rpm
$ sudo yum clean all
```

提示

Yum 源的官方地址为 <http://yum.puppetlabs.com>。Yum 支持 RedHat、CentOS、Fedora 操作系统。

2) 在服务器端安装 Puppet。运行如下代码会自动匹配安装 Facter 等相关依赖包 (包括 Facter):

```
$sudo yum install puppet-server
```

运行如下命令来检查 Puppet 的安装:

```
$ puppet -V
3.0.1
$ facter -v
2.0.0-rc4
```

3) 在客户端安装 Puppet。同理运行如下代码会自动匹配安装相关依赖包 (包括 Facter):

```
$sudo yum install puppet
```

到此在 Linux 系统下的 Puppet 安装就完成了, 配置方法将在 2.7 节讲解。接下来我们看 Puppet 在 Fedora 15 和 Fedora 16 上的安装方法。

(2) 在 Fedora 上安装 Puppet

在 Fedora 上安装 Puppet 也非常简单, 和 Centos 方法一样, 配置好 PuppetLabs 官方源, 采用 Yum 安装。注意选择自己系统版本的 Yum 软件包进行配置。这里以 Fedora 16 为例进行介绍。下面介绍具体的安装步骤。

1) 配置 Yum 源, 配置方法如下:

```
$ sudo rpm -Uvh http://yum.puppetlabs.com/fedora/f17/products/x86_64/puppetlabs-release-17-6.noarch.rpm
$ sudo yum clean all
```

2) 在服务器端安装, 安装方法如下:

```
$sudo yum install puppet-server
```

3) 在客户端安装, 安装方法如下:

```
$sudo yum install puppet
```

安装完成后检查安装状态, 具体命令与前面的一样 (采用 -v 查看, 注意 puppet 后是大写 V)。

(3) 在 Debian 和 Ubuntu 上安装 Puppet

在 Debian 和 Ubuntu 上安装 Puppet 的方法也非常简单, 只需要配置好仓库就可以了。

提示

apt-get 是一条 Linux 命令, 适用于 deb 包管理式的操作系统, 主要用于自动从互联网的软件仓库中搜索、安装、升级、卸载软件或操作系统。目前主要应用于 Debian 和 Ubuntu。

注意先安装 Ruby。我们也可以从 PuppetLabs 源 (<http://apt.puppetlabs.com/>) 来配置和安

装 Ruby。

1) 配置 puppetlabs 仓库，配置方法如下：

```
$ sudo echo -e "deb http://apt.puppetlabs.com/ lucid main\ndeb-src http://apt.puppetlabs.com/ lucid main" >> /etc/apt/sources.list.d/puppet.list
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv 4BD6EC30
$ sudo apt-get update
```

2) 在服务器端安装，安装方法如下：

```
$ sudo apt-get install puppetmaster
```

3) 在客户端安装，安装方法如下：

```
$ sudo apt-get install puppet
```



提示

此方法同样适用于如下操作系统：Scientific Linux 5 和 6、Ascendos 5 和 6。

2.4.2 从源代码进行安装

相比以 Yum 方式安装 Puppet，源代码安装显得较为复杂，需要按照相关依赖逐步进行。通常我们需要到相关软件下载页面下载对应平台的版本源代码文件，并逐步进行编译安装。这里以 Centos 6 安装为例进行讲解。

(1) 安装 Ruby

建议采用 Yum 安装好 Ruby 再进行 Puppet 源代码安装。当然都采用源代码安装也没有关系。我们首先到 Ruby 的下载页面（<http://www.ruby-lang.org/en/downloads/>）选择 Ruby 软件包，建议选择 Ruby 1.8.7 版本。

下载 ruby-1.8.7-p370.tar.gz 软件包，进入相应目录，进行编译安装，安装方法如下：

```
$ mkdir /opt/puppetlabs
$ cd /opt/puppetlabs
$ wget ftp://ftp.ruby-lang.org/pub/ruby/1.8/ruby-1.8.7-p370.tar.gz
$ tar zxvf ruby-1.8.7-p370.tar.gz
$ cd ruby-1.8.7-p370
$ ./configure && make
$ sudo make install
```

(2) 安装 Facter

下载最新版本的 facter-1.6.13.tar.gz 软件包进行安装，安装方法如下：

```
$ wget http://downloads.puppetlabs.com/facter/facter-1.6.13.tar.gz
$ tar -zxf facter-1.6.13.tgz
$ cd facter-1.6.13
$ sudo ruby install.rb
```

(3) 安装 Puppet

下载最新版本的 puppet-3.0.1.tar.gz 软件包进行安装，安装方法如下：

```
$ wget http://downloads.puppetlabs.com/puppet/puppet-3.0.1.tar.gz
$ gzip -d -c puppet-3.0.1.tgz | tar xf -
$ cd puppet-3.0.1
$ sudo ruby install.rb
# 默认安装在 /etc/puppet 目录中，如果需要指定安装目录可以使用如下命令
$ sudo ruby install.rb --destdir=/opt/puppet
```

运行如下命令来检查 Ruby 及 Puppet 的安装：

```
$ ruby -v
ruby 1.8.7
$ puppet -V
3.0.1
$ facter -v
1.6.13
```

2.4.3 从 Git 版本库进行安装

如果在本地克隆了一个 Git 项目的版本库，就可以通过版本库的方式获取 Puppet 的最新版本的信息。这样在下载不同版本的 Puppet 时采用的是增量的方法，每个版本直接采用 git pull 命令即可更新至最新版。也可以选择任意版本进行安装。缺点是打包批量部署时不是非常方便。注意先安装好 Ruby。

1) 获取版本库到本地，命令如下：

```
$ sudo mkdir -p /usr/src
$ cd /usr/src
$ sudo git clone git://github.com/puppetlabs/facter
$ sudo git clone git://github.com/puppetlabs/puppet
```

提示

如果系统没有安装 Git 可以通过 yum install git 命令进行安装。

2) 安装 Facter，命令如下：

```
$ sudo cd facter
$ sudo ruby install.rb
```

3) 我们可以通过 git tag 命令查看 Git 中的 tag（即标签），并通过 git checkout 命令选择自己所需要的版本进行安装。本例将选择 3.0.1 版本进行安装，命令如下：

```
$ git tag
...
2.7.9
3.0.0
```

```
3.0.1
3.0.1-rc1
...
$ git checkout 3.0.1
HEAD is now at 1d7d7cb... Update PUPPETVERSION for 3.0.1
```

4) 安装 Puppet, 命令如下:

```
$ ruby install.rb
```

2.4.4 通过 Gems 进行安装

和大部分基于 Ruby 的程序一样, Puppet 和 Facter 也能通过 RubyGems 进行安装, 前提是先在操作系统上安装 Ruby 和相应的 RubyGems 包。在 RedHat、CentOS、Fedora、SUSE/SLES、Debian 和 Ubuntu 上, 这个软件包的名字是 RubyGems。当这个包安装好后, 就能使用 gem 命令像下面这样来安装 Puppet 和 Facter 了。默认安装的 Puppet 版本为 3.0。



提示

RubyGems (简称 Gems) 是一个用于对 Rails 组件进行打包的 Ruby 打包系统。它提供一个分发 Ruby 程序和库的标准格式, 还提供一个管理程序包安装的工具。RubyGems 的功能类似于 Linux 下的 Yum。有关细节可参考: <http://baike.baidu.com/view/663386.htm>。

1) 安装 RubyGems, 命令如下:

```
$ sudo wget http://production.cf.rubygems.org/rubygems/rubygems-1.8.24.tgz
$ cd rubygems-1.8.24
$ ruby setup.rb
```

2) 安装 Puppet, 命令如下:

```
$ sudo gem install puppet
```

2.5 在 Mac OS X 下安装

2.5.1 通过二进制发布包进行安装

Mac OS X 被认为是操作系统中“高富帅”的象征, 是最为人性化的操作系统之一。在 Mac 系统上工作是一件非常惬意的事。下面我们看看在 Mac 下如何安装 Puppet。

(1) 下载二进制包

从官方网站可以下载编译好的 dmg 二进制包并进行安装, 命令如下:

```
http://downloads.puppetlabs.com/mac/
$ wget http://downloads.puppetlabs.com/mac/facter-1.6.13.dmg
$ wget http://downloads.puppetlabs.com/mac/puppet-3.0.1.dmg
```




提示

.dmg 是 Mac 磁盘镜像格式的软件发布包。

(2) 挂载镜像文件

Mac 的 .dmg 格式软件就是一个磁盘映像，双击即可完成挂载，打开后如图 2-1 所示。

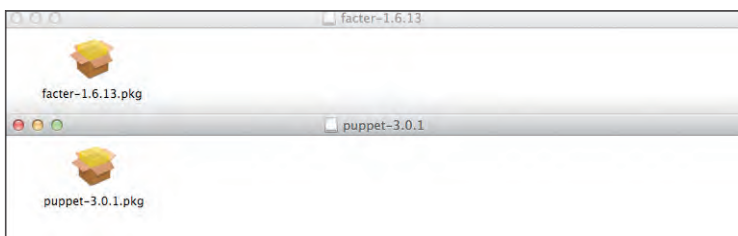


图 2-1 在 Mac OS X 下打开 .dmg 格式磁盘映像

(3) 安装 Factor 和 Puppet

图 2-1 显示了两个 pkg 文件。Puppet-3.0.1.pkg 就是 Puppet 安装程序。分别点击扩展名为 .pkg 的安装程序，开始安装 Factor 和 Puppet。注意，需要先安装 Factor，输入 root 密码后即可在默认安装程序目录下完成安装。两个软件安装完成后的效果一样。以 Puppet 安装为例，效果如图 2-2 所示。

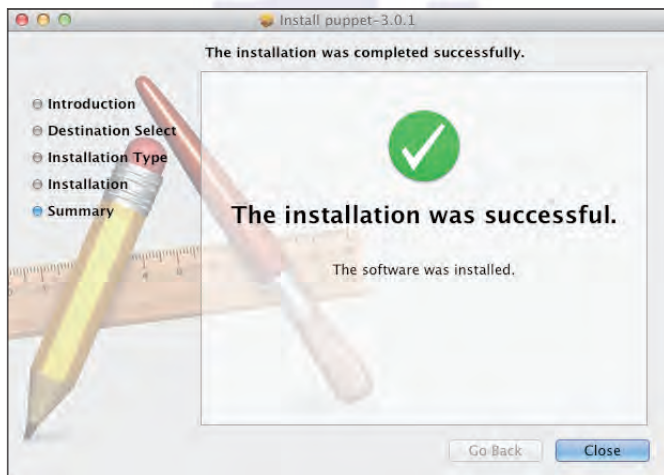


图 2-2 在 Mac OS X 下成功安装 Puppet

2.5.2 从 Git 版本库进行安装

在 Mac 下采用 Git 安装 Puppet 与在 Linux 下的安装方法一样，这里不再详细介绍，只简单介绍一下安装步骤。

1) 创建目录来下载版本文件, 命令如下:

```
$ sudo mkdir -p /opt/puppetlabs
$ cd /opt/puppetlabs
$ sudo git clone git://github.com/puppetlabs/facter
$ sudo git clone git://github.com/puppetlabs/puppet
```

2) 安装 Facter, 命令如下:

```
$ cd facter
$ sudo ruby install.rb
```

3) 安装 Puppet, 命令如下:

```
$ cd puppet
$ sudo ruby install.rb
```

4) 确认 Puppet 版本, 命令如下:

```
$ puppet -V
3.0.1
```

2.5.3 从 Ports 仓库进行安装

在 Mac 下面除了可以用 dmg、pkg 来安装软件外, 还可以利用 MacPorts 比较方便地安装其他应用程序, 跟 BSD 中的 Ports 道理一样。MacPorts 就像 apt-get、yum 一样, 可以快速安装软件。

MacPorts 的官方网址: <http://www.macports.org>。

从 MacPorts 官方网址下载对应的 Mac 系统的软件包并进行安装, 具体如下。

1) 在 Mac OS X Lion 系统下安装 MacPorts, 命令如下:

```
https://distfiles.macports.org/MacPorts/MacPorts-2.1.2-10.8-MountainLion.pkg
# 安装过程省略。双击即可完成安装
```

2) 使用 port 命令安装 Puppet。查找 ports 仓库源中 Puppet 的安装包, 使用 port 命令进行在线安装。如果查询到多个版本的 Puppet, 必须指定版本才可以。这里安装 Puppet 时只支持 2.7.6 版本。

```
$ port search puppet # 用于查找 ports 仓库提供安装的 Puppet 版本
Warning: port definitions are more than two weeks old, consider using selfupdate
puppet @2.7.6 (sysutils)
  Puppet is a configuration management solution.
$ sudo port install puppet # 进行安装操作
```

至此, 我们已经讲完了 Puppet 在主流 Linux 下的安装方法, 但并没有讲到如何去配置使用 Puppet, 这些内容我们将在 2.6 节集中讲解, 因为 Puppet 在不同操作系统下的配置与使用方法是相同的。

2.6 配置 Puppet

在 2.4 节和 2.5 节中我们讲解了 Puppet 在各 Linux 操作系统和 Mac 操作系统上的安装方法，接下来我们将讲如何配置 Puppet，并且配置一个简单的 File 资源进行测试。默认安装好的 Puppet 可以直接启动使用。

1) 服务的启用方法如下：

```
$ /etc/init.d/puppetmaster      # 服务端
{start|stop|status|restart|reload|force-reload|condrestart|genconfig}
$ /etc/init.d/puppet            # 客户端
{start|stop|status|restart|reload|force-reload|condrestart|once|genconfig}
```



提示

如果安装完 Puppet 后没有相应的配置文件，需要参考 1.6 节创建核心配置文件。

2) 设置主机名和指定 hosts。分别向 Puppet-Master 和 Puppet-Agent 添加如下配置文件：

```
#Master
$ sudo hostname agent.domain.com
#Agent
$ sudo hostname puppet.domain.com
# 两端分别添加
$ sudo vim /etc/hosts
192.168.1.22 agent.domain.com
192.168.1.2 puppet.domain.com
```



提示

可以采用 DNS 来管理主机信息，笔者将在第 8 章详细讲解主机管理。

3) 配置防火墙。如果 Puppet Master 开启了防火墙，需要做如下配置：

```
#Puppet 会监听 8140 端口，因此我们需要开放 8140 端口
iptables -A INPUT -p tcp --dport 8140 -j ACCEPT
or iptables -A INPUT -p tcp -s 10.10.0.0/24 --dport 8140 -j ACCEPT
```

4) 配置主机配置文件。修改 site.pp 增加输出文件配置，在这里我们定义了一个 File 资源：

```
$ sudo vim /etc/puppet/manifests/site.pp
node default { file { ["/tmp/puppettest1.txt": content => "Hello,First Puppet test"; ] }
```

5) 客户端发起验证，命令如下：

```
$ puppet agent --server puppet.domain.com --test
Info: Creating a new SSL key for agent.domain.com
Info: Creating a new SSL certificate request for agent.domain.com
Info: Certificate Request fingerprint (SHA256): 78:1D:CF:9E:F2:37:6C:33:C0:13:52:
```

```
7C:C9:25:ED:0D:49:14:09:94:8C:9D:6A:62:A8:AB:9D:23:9B:0A:14:8C
Exiting; no certificate found and waitforcert is disabled
```

在上面的代码中我们使用了两个参数来运行 Puppet Agent，第一个参数 `--server` 指定了需要连接的 Puppet Master 的名字或地址。

提示

如果不使用 `--server` 参数指定 Puppet Master 服务器，Puppet Agent 会默认寻找一个名为 “puppet” 的主机，所以需要为 Puppet Master 创建一个别名 (CNAME)。

客户端必须要能够解析需要连接到的主机名。这就是为什么我们要在第 2 步时配置主机名并指定 `hosts` 的原因。如果我们没有指定 `--server` 参数，需要在客户端的配置文件 `/etc/puppet/puppet.conf` 的 `main` 段指定这一参数，具体如下：

```
[main]
server=puppet.example.com
```

第二个参数 `--test` 使 Puppet 客户端运行时只是测试模式。如果想在前台输出日志到标准输出，可以使用参数 `--no-daemonize`。在默认情况下，Puppet 客户端是以守护进程的方式运行的。如果想在客户端输出详细的日志，可以使用参数 `--verbose`。如果认为还不够，再加上 `--debug` 参数就能提供更加详细的输出，这在解决问题的时候非常有用。

在第 5 步中可以看到我们发起的连接的输出。Agent 发起了一个证书验证请求并且使用一个私钥来加密连接。Puppet 使用 SSL 证书来验证 Master 和 Agent 之间的连接。Agent 向 Master 发出证书验证请求，然后等待 Master 签名并返回证书。

现在 Agent 依然在运行并等待已被签名的证书。在证书到达或退出之前，Agent 会持续每两分钟检查一次是否存在被签名的证书。

提示

可以使用 `--waitforcert` 参数改变 Puppet Agent 的等待时间。可以按秒指定，如果指定为 0，则表示不等待证书（在这种情况下 Agent 会自动退出）。

6) 服务端完成验证。为了完成连接并对 Agent 进行验证，我们需要对 Agent 发送到 Master 的证书进行签名，可以通过使用 Master 上的 `puppet cert` 命令 (`--list` 参数) 查看等待被签名的证书，使用 `sign` 参数对等待被签名的证书进行行签名。如果请求验证主机比较多可以使用 `--all` 参数给所有主机签署认证。

```
$ puppet cert --list
agent.domain.com

$ puppet cert sign agent.domain.com
Signed certificate request for agent.domain.com
Removing file Puppet::SSL::CertificateRequest agent.domain.com at '/var/lib/puppet/ssl/ca/requests/agent.domain.com.pem'
```

7) 客户端再次运行配置。我们已经通过第 4 步为 agent.domain.com 主机配置了一个 File 资源。我们还可以再次运行 puppet agent 命令配置 File 资源, 代码如下:

```
$ sudo puppet agent --server puppet.domain.com --test
Info: Caching certificate for agent.domain.com
Info: Caching certificate_revocation_list for ca
Info: Retrieving plugin
Info: Caching catalog for agent.domain.com
Info: Applying configuration version '1351506299'
Info: Creating state file /var/lib/puppet/state/state.yaml
```

8) 验证配置, 命令如下:

```
$ more /tmp/puppettest1.txt
Hello,First puppet test
```

很多读者在进行签名验证时很容易遇到一些疑惑, 为此笔者增加有关签名验证的说明, 以帮助大家解决一些常见的签名故障。

客户端重新签名时需要删除 'hostname'.pem 证书文件, 操作命令如下:

```
Find /var/lib/puppet/ssl/ -iname 'hostname'.pem -exec /bin/rm -rf {}
```

如果偷懒可以删除 ssl 整个目录, 操作命令如下:

```
$ sudo rm -rf /var/lib/puppet/ssl
```

服务端重新签名同样需要删除签名相关文件, 操作命令如下:

```
$ sudo find $(puppet master --configprint ssl_dir) -name "$(puppet master --configprint certname).pem" -delete
```

如果需要删除某台客户端的认证信息, 可以使用如下命令:

```
$ sudo puppet cert --clean {node certname}
```

同时需要在客户端删除 ssl 目录。



提示

Puppet 命令与帮助命令的使用方法有所区别, 特别是在 Puppet 3.0 与 Puppet 2.6 以下版本中。相关内容将在第二部分中详细讲解。

2.7 在 Windows 下安装与使用

Puppet 对 Windows 提供了友好的支持, 而且在 3.0 版本中进行了功能加强, 不过 Puppet 在 Windows 下的功能全面性略逊于在 Linux 下。

Puppet 所支持的 Windows 操作系统有:

❑ Windows Server 2003 和 Windows Server 2003 R2

- ❑ Windows Server 2008 和 Windows Server 2008 R2
- ❑ Windows 7

2.7.1 包管理器方式安装

- 1) 从官方网址 (<http://downloads.puppetlabs.com/windows>) 下载 Windows 下编译好的 msi 软件包。
- 2) 双击运行 puppet-3.0.1.msi, 会出现如图 2-3 所示的安装向导。

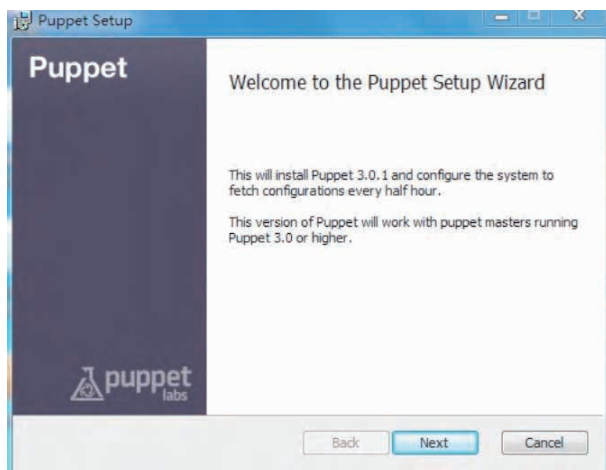


图 2-3 双击运行 puppet-3.0.1.msi 后的安装向导

- 3) 勾选接受同步授权协议的复选框, 如图 2-4 所示。

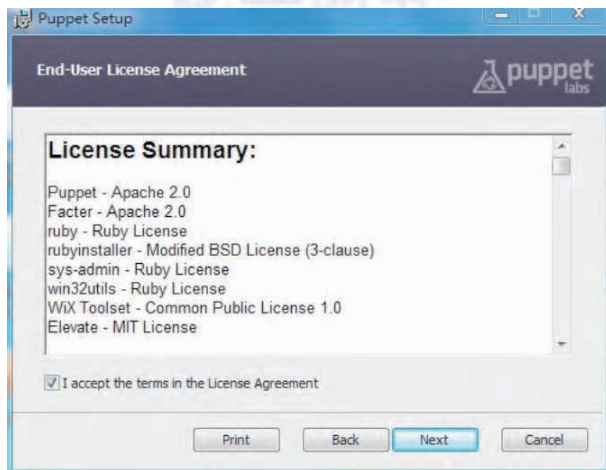


图 2-4 同意授权协议

- 4) 选择安装的路径与 Master 主机名, 如图 2-5 所示。

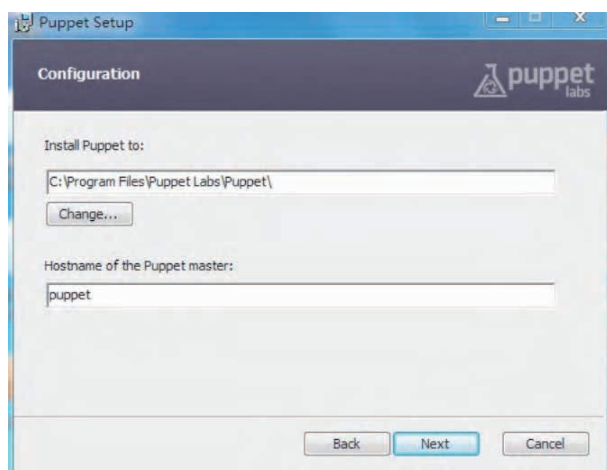


图 2-5 选择安装路径与 Master 主机名

5) 完成安装后我们就可以依次选择“开始→所有程序→Puppet”，打开图 2-6，右键选择 Run as administrator。

- ☐ Run Facter
- ☐ Run Puppet Agent
- ☐ Start Command Prompt with Puppet (类似 UNIX 命令行工具)

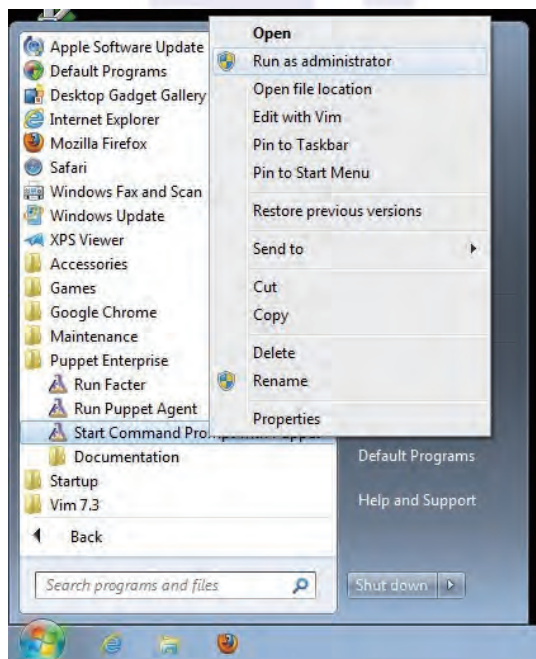


图 2-6 在 Windows 下打开 Puppet Command



提示

Start Command Prompt with Puppet 是 puppetlabs 给我们提供的在 Windows 简化操作的一个工具。

6) 我们需要先运行 Run Factor, 再打开 Start Command Prompt with Puppet。由于笔者习惯命令行操作, 因此采用这种方式。我们可以在这里查看相应的帮助 (运行 Puppet help 即可)。相关命令操作与在 Linux 中一样。

2.7.2 在 Windows 下使用 Puppet

对于习惯在命令行界面操作的系统管理员来说, 在 Windows 中使用 Puppet 总归不是那么顺手, 不过 puppetlabs 为我们提供了相对便利的 Command 命令提示符工具。笔者将使用 Command 方式在 Windows Server 2008 下完成所有认证、配置等操作。

1) 客户端发起认证请求, 命令如下:

```
C:\Program Files (x86)\Puppet Labs\Puppet\bin>puppet agent --server puppet.domain.com
```

2) 在服务端完成认证, 命令如下:

先使用 cert list 查看当前未签署的主机认证请求, 再使用 puppet cert sign 命令签署认证:

```
$ puppet cert list # 查看客户端请求认证列表
"win-102qi54ri1o" (SHA256) D6:57:20:50:B6:50:AA:A4:00:4D:0C:B6:64:5A:CF:B2:C8
:45:A1:D4:46:BD:E3:AD:A9:C5:64:C2:86:FE:CD:D8
$ puppet cert sign win-102qi54ri1o # 向客户端颁发签名
Signed certificate request for win-102qi54ri1o
Removing file Puppet::SSL::CertificateRequest win-102qi54ri1o at '/var/lib/
puppet/ssl/ca/requests/win-102qi54ri1o.pem'
```

3) 在服务端配置客户端信息。我们需要在 site.pp 增加对这台 Windows 客户端的配置, 并为客户端配置 File 资源。在 C 盘创建内容为 “Hello, I am Liuyu, I come from LinuxTone.org!” 的 readme.txt 文件。

```
$ sudo vim /etc/puppet/manifests/site.pp # 编辑 site.pp 文件, 增加 win 节点主机
import "nodes/win-102qi54ri1o.pp"
$ sudo vim /etc/puppet/manifests/nodes/win-102qi54ri1o.pp
# 创建 win 节点主机信息, 配置 File 资源
node 'win-102qi54ri1o' {

  file { ["C:/Readme.txt":
    ensure => present,
    content => "Hello,I am Liuyu,I come from LinuxTone.org!",
  ]
}
```

4) 在客户端运行配置。在运行配置前, 可以先使用 --noop 参数, Puppet 将会自行检测

语法与代码验证，即执行相应的代码变更但并不实际运行。命令运行后显示的信息如下。如果出现 Finished 说明运行成功，配置没有错误。

```
C:\Program Files (x86)\Puppet Labs\Puppet\bin>puppet agent --server puppet.domain.
com --test --noop
Info: Retrieving plugin
Info: Caching catalog for win-102qi54rilo
Info: Applying configuration version '1351582640'
/Stage[main]/Node[win-102qi54rilo]/File[C:/Readme.txt]/ensure: current_value ab
sent, should be present (noop)
Node[win-102qi54rilo]: Would have triggered 'refresh' from 1 events
Class[Main]: Would have triggered 'refresh' from 1 events
Stage[main]: Would have triggered 'refresh' from 1 events
Finished catalog run in 0.16 seconds
# 去掉 noop 参数，使其运行生效
C:\Program Files (x86)\Puppet Labs\Puppet\bin>puppet agent --server puppet.
domain.com --test
Info: Retrieving plugin
Info: Caching catalog for win-102qi54rilo
Info: Applying configuration version '1351582825'
/Stage[main]/Node[win-102qi54rilo]/File[C:/Readme.txt]/ensure: created
Finished catalog run in 0.12 seconds
```

5) 检查配置是否生效。我们可以看得到在 C 盘成功创建了文件 Readme.txt，其内容为：“Hello, I am Liuyu, I come from LinuxTone.org!”，如图 2-7 所示。

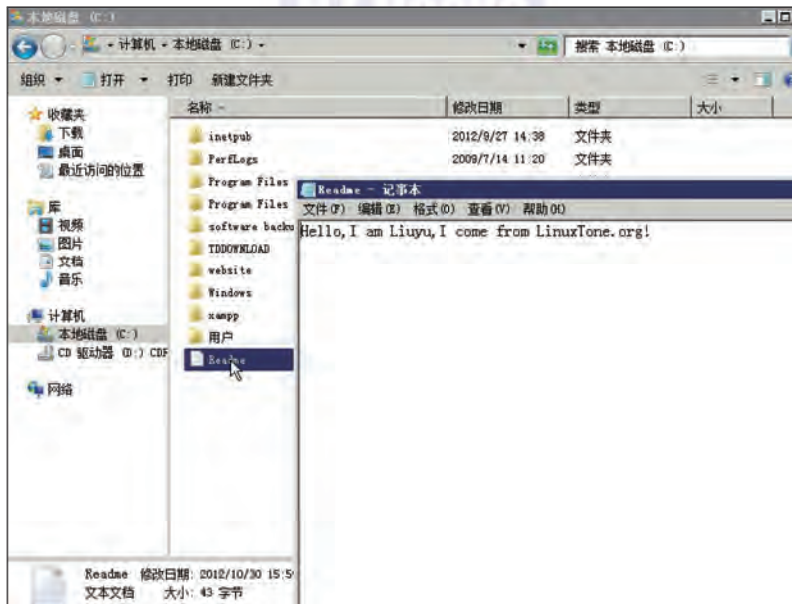


图 2-7 确认客户端执行结果

2.7.3 Puppet 在 Windows 下的功能

(1) Puppet 在 Windows 上可安装的版本

在 Windows 上可安装 Puppet 2.6 以上版本。Puppet 3.0 对 Windows 的支持更为友好，增加了新的功能。

(2) Puppet 对 Windows 系统提供的命令支持

☐ Puppet Agent 配置文件及启用服务。如果需要连接 *UNIX 的 Master，则需要 Puppet 2.7 版本以上。

☐ Puppet Apply 配置本地节点信息。

☐ Puppet resource 资源操作，配置操作系统相关。

☐ Puppet inspect 检查相关报告。

(3) Puppet 对 Windows 所提供的资源类型支持

Puppet 支持 Windows 提供的如下资源类型：

☐ file

☐ user

☐ group

☐ scheduled_task (Windows 专属资源类型)

☐ package

☐ service

☐ exec

☐ host

2.8 如何升级

笔者建议在升级 Puppet 前阅读官方提供的 Release_Notes (http://projects.puppetlabs.com/projects/puppet/wiki/Release_Notes) 文档，了解官方对于版本间区别的描述。不建议跳级升级，例如。现在是 Puppet 2.6，不建议直接升级至 Puppet 3.0，需要升级至 Puppet 2.7 过渡一段时间，待稳定后再升级到 Puppet 3.0。升级时需要先升级 Master 并启用测试进程，再升级 Agent。通过版本的过渡避免版本之间的不明故障。升级大体可以分如下 6 步：

1) 为 Master 安装新版本。建议采用源代码安装。使用命令 `ruby install.rb --destdir=newdir` 指定安装目录。

2) Master 启动时采用新的端口 (`puppet master --no-daemonize --verbose --port 8141`)，相当于两个 Master 同时提供服务。

3) 将 Agent 配置指定到 Master 8141 端口。

4) 运行无故障后，升级 Agent 至最新版本。修改 Agent 配置指定到 8141 端口。

5) 反复确认是否有故障、查看日志等,看是否按预期完成。

6) 对 Master 进行系统升级。取消之前的测试进程。

其实升级非常简单,通常不会发生太大的问题。如果觉得以上步骤过于繁琐,可以直接升级 Master,让原有 Agent 运行一段时间,在确保没有问题后,直接采用 Yum 升级 Puppet 即可。

2.9 本章小结

本章详细介绍了 Puppet 在各种操作系统上的安装和配置方法,使读者加深对 Puppet 的理解。读者可以根据自己的需求选择不同的方法进行安装。在介绍完 Puppet 安装后,通过创建简单的 File 资源学习了 Puppet 基本使用方法,详细讲解了签名验证。下一章我们将通过创建相关的实例带领读者逐步学习如何构建 Puppet 的配置。



第 3 章

创建你的第一个 Puppet 配置

通过上一章，我们已经搭建好了 Puppet 环境，并且学会了如何配置简单的 File 资源。本章我们将学习创建一个测试实例的方法进而使读者加深对 Puppet 的理解，掌握 Puppet 最基本的用法。主要操作步骤为：创建一个测试节点、创建测试 httpd 模板、创建测试模块、在客户端检测运行、确认运行结果。在配置过程中可能会有迷惑或暂时不理解的概念，没有关系，记下来，下一部分我们将继续学习。

本章的知识点为：节点创建、类的创建、模块的创建、Ruby 模板的配置方法、如何指定 Yum 源安装、Httpd 模块的配置等。

3.1 配置一个测试节点

一个节点的目录结构可以任意定义，只要在 site.pp 配置文件中 import 即可。笔者建议采用如下目录结构配置。

□ 节点信息：/etc/puppet/manifests/nodes

□ 模块信息：/etc/puppet/modules

要按照以上目录结构进行配置，我们需要先创建一个模块信息。

1) 创建 test 模块。模块目录为 test，class 类名也必须是 test。test 类里有一个 File 资源，File 资源的目的是在 /tmp 目录中创建一个以 Agent 主机名命名的 txt 文件，其内容为“Hello World!”。执行命令如下：

```
# 创建必要的目录。Manifests 存放着模块的主配置文件 init.pp
# Puppet 在应用类后会首先加载 init.pp 读取必要的信息
# 如果 init.pp 使用 templates 函数，会读取 templates 目录下面的 ERB 模块文件
# 当前变量也会传入 templates 函数中
$ sudo mkdir -p /etc/puppet/modules/test/{manifests,templates,files}
$ sudo vim /etc/puppet/modules/test/manifests/init.pp
class test {
    file { ["/tmp/$hostname.txt": content => "Hello World!"; ]
}
```

2) 在模块配置文件中定义了一个变量“\$hostname”，需要将该变量传给 ERB 模块文件。此文件存放在 test 类目录下的 templates 目录中，文件名与类名保持一致。主机名变量值通过 facter 命令获取，以变量“\$hostname”的形式传递给 agent.domain.com.pp。配置方法如下：

```
$ sudo vim /etc/puppet/modules/test/templates/test.erb
hostname <%= fqdn %>
```

3) 通过以上两步我们创建了测试节点所需要的模块信息。现在我们来创建测试节点 agent.domain.com，配置代码如下：

```
$ sudo vim /etc/puppet/manifests/nodes/agent.domain.com.pp
node 'agent.domain.com' {
  # 加载 test 类
  include test
}
```

4) 最后我们需要将测试节点载入到 Puppet，也就是修改 site.pp 配置文件。在第 1 章中我们讲到了 Puppet 会首先读取 site.pp。因此，在创建客户端节点时需要在 site.pp 配置文件中设置 import，代码如下：

```
$ sudo vim /etc/puppet/manifests/site.pp
# 追加如下一行
import "nodes/agent.domain.com.pp"
# 也可以使用 import "nodes/*.pp"
```

import 指令告诉 Puppet 载入 nodes 目录中所有以 .pp 结尾的文件。当 Puppet 启动时就会载入并处理 agent.domain.com.pp 文件了。

到此一个测试节点就配置完毕了，接下来我们需要检查配置文件的语法。

3.2 检测你的配置文件

在完成所有节点的配置后，笔者不建议直接运行这些配置。我们需要对配置文件的语法进行检查，以避免在客户端运行时因遇到报错而重复修改。

检测语法也分两步进行：第一步是在 Puppet Master 服务端采用 puppet parser validate 命令检查配置语法，第二步是在 Puppet Agent 客户端采用 --noop 参数检测语法及代码。

(1) Puppet Master 服务端检查

在 Puppet Master 服务端执行 puppet parser validate init.pp，如果看到 Finished 表示成功，无语法错误。有关语法规则及常见语法错误的处理我们将在第 5 章中详细讲解。

```
$ puppet parser validate /etc/puppet/modules/test/manifests/init.pp
Info: Applying configuration version '1351521612'
Info: Creating state file /var/lib/puppet/state/state.yaml
Finished catalog run in 0.03 seconds
```

如以上代码所示，配置文件无任何语法错误，可以进行下一步操作。

(2) Puppet Agent 客户端检查

客户端的检查主要是通过增加 `--noop` 参数来实现的，其目的只是验证配置、模拟执行，但并不实际生效。执行命令及结果如下：

```
$ puppet agent --test --server puppet.domain.com --noop
Info: Retrieving plugin
Info: Caching catalog for agent.domain.com
Info: Applying configuration version '1351522305'
/Stage[main]/Test/File[/tmp/agent.txt]/ensure: current_value absent, should be
file (noop)
# 如上可以看出要创建 agent.txt 文件，由于采用的是 noop 模式，代码没有被执行
Class[Test]: Would have triggered 'refresh' from 1 events
Stage[main]: Would have triggered 'refresh' from 1 events
Finished catalog run in 0.04 seconds
#--noop 的意思是验证配置，但并不执行

# 使用 ls 查看文件并没有被创建
$ ls /tmp/agent.txt
ls: cannot access /tmp/agent.txt: No such file or directory
```



注意

`puppet agent` 命令，如果不使 `--server` 指定 master，agent 会读取 `/etc/puppet/puppet.conf` 中 `server` 参数所指定的 Master。

3.3 客户端运行配置

通过 3.2 节的检测我们看到语法与代码检测都没有问题。现在准备好了一切，可以在客户端运行了。取消 `noop` 参数，执行结果如下：

```
$ puppet agent --test --server puppet.domain.com
Info: Retrieving plugin
Info: Caching catalog for agent.domain.com
Info: Applying configuration version '1351522305'
/Stage[main]/Test/File[/tmp/agent.txt]/ensure: defined content as '{md5}ed076287
532e86365e841e92bfc50d8c'
Finished catalog run in 0.04 seconds
```

通过以上代码我们可以看到 Puppet 的配置运行成功，并且提示完成 Catalog 的运行花费的时间是 0.04 秒。

让我们来看看运行过程中都发生了什么。首先 Agent 缓存了主机信息，接下来可以看到资源被应用了，`/tmp/agent.txt` 文件被创建，并根据文件内容计算出 MD5 值。最后告知整个过程所消耗的时间。同样在 Master 日志也可以看到运行过程被记录下来。

3.4 查看运行结果

在 3.3 节的配置执行成功后，我们可以在客户端验证运行结果。客户端运行的结果将会和我们在 Puppet Master 服务端所定义的一样，显示的内容如下：

```
$ more /tmp/agent.txt
Hello World!
```

3.5 增加 httpd 模块

通过上面的配置实践，我们掌握了 Puppet 基本的配置方法。现在我们加强难度，采用自定义 Yum 源的方式安装 httpd 软件。这里我们需要创建相应的模块与类，命名都可以是 httpd。

提示

在配置前读者可以自己先创建一次，看是否与笔者创建的类似。

1) 创建 httpd 模块相应的目录，命令如下：

```
$ sudo mkdir -p /etc/puppet/modules/httpd/{manifests,templates,files}
```

提示

本例并没有使用 templates 函数与 files，实际上，为了保证目录的一致性这些都需要创建。

2) 编辑 httpd 模块文件，指定源配置相关信息。这里我们将采用 163 的源进行安装，具体代码如下：

```
$ sudo vim /etc/puppet/modules/httpd/manifests/init.pp
class httpd {
  yumrepo { "repo163":
    descr    => "163 repo",
    baseurl  => "http://mirrors.163.com/centos/6/os/x86_64/",
    gpgcheck => "0",
    enabled  => "1";
  }

  package {
    "httpd":
      ensure => installed,
      require => Yumrepo["repo163"];
  }
}
```

3) 修改 agent.domain.pp, 增加 httpd 模块, 命令如下:

```
$ sudo vim /etc/puppet/manifests/nodes/agent.domain.com.pp
node 'agent.domain.com' {
    include test
    include httpd
}
```

在上述代码中, httpd 模块的 init.pp 文件包含了一个单独的类 httpd。该类中包含了两个资源: yumrepo 和 package。

第一个资源 yumrepo 指定了 yumrepo 的配置文件信息。第二个资源 package 使用 ensure=>installed 确保 httpd 包的安装, 并指定了一个新的属性 require。require 属性是一个元参数。元参数是属于 Puppet 框架一部分的资源属性, 并不属于某一个特定的资源类型。

可以说元参数指定了资源之间的依赖关系。Package["httpd"] 依赖于 Yumrepo["repo163"] 资源, 因此 Yumrepo["repo163"] 资源肯定首先被执行。资源之间的关系是 Puppet 的重要部分, 我们会在第 6 章中详细讲解。

4) 检查配置文件语法的正确性, 命令如下:

```
$ puppet parser validate /etc/puppet/modules/httpd/manifests/init.pp
Error: Could not parse for environment production: Syntax error at '::'; expected
'}' at /etc/puppet/modules/httpd/manifests/init.pp:2 on node puppet.domain.com
Error: Could not parse for environment production: Syntax error at '::'; expected
'}' at /etc/puppet/modules/httpd/manifests/init.pp:2 on node puppet.domain.com
# 可以看到 " /etc/puppet/modules/httpd/manifests/init.pp" 的第二行配置
# 中的 '::' 不符合语法规则, 这是半角输入的问题, 切换至英文状态将其修改

# 修改过后再次运行
$ puppet parser validate /etc/puppet/modules/httpd/manifests/init.pp
Info: Applying configuration version '1351523883'
Finished catalog run in 0.03 seconds
```

5) 在客户端运行配置, 命令如下:

```
$ puppet agent --test --server puppet.domain.com
Info: Retrieving plugin
Info: Caching catalog for agent.domain.com
Info: Applying configuration version '1351524268'
/Stage[main]/Httpd/Package[httpd]/ensure: created
Finished catalog run in 8.40 seconds
```

6) 检查 httpd 是否已经成功安装, 命令如下:

```
$ rpm -q httpd
httpd-2.2.15-15.el6.centos.1.x86_64
```


3.6 本章小结

通过本章的学习，读者应该已经学会如何创建一个模块和类、如何安装软件，以及如何指定安装源。其实 Puppet 配置就这么简单，掌握基础的配置方法，将要实现什么告诉 Puppet 就可以了。随着配置越来越复杂，应该考虑将它们放到版本控制当中，针对不同系统环境的配置还应该考虑环境的因素。因此我们将在下一章学习 Puppet 多种环境的部署。结合 Git 版本控制跟踪变更，当需要时可以回滚到之前的状态，或者在不影响正常运行的情况下做出变更，这将是非常有用的。



6.2.2 虚拟资源的用法

虚拟资源常用于用户的管理。下面将通过实例讲解如何运用虚拟资源来管理用户。

假设有三个用户：seven、jeck、lucy。为了简化管理，定义两个用户组：dba、sysadmin。其中 seven 属于 dba。Jeck 属于 sysadmin。lucy 既属于 dba 又属于 sysadmin。如果直接配置 Puppet 会产生冲突，为了避免冲突，需要将这些用户配置成虚拟资源，并定义在一个单独的类 user::virtual 中。在需要实例的时候采用 realize 函数应用。下面介绍配置方法。

1) 创建用户模块目录，代码如下：

```
$ mkdir -p /etc/puppet/modules/user/manifests
```

2) 创建 user::virtual 类的 virtual.pp 文件，代码如下：

```
$ vim /etc/puppet/modules/user/manifests/virtual.pp
class user::virtual
{
    @user {'seven':
        ensure => present,
        uid    => 540,
        gid    => 540,
        home   => '/home/seven',
        shell  => '/bin/bash',
    }
    @user {'jeck':
        ensure => present,
        uid    => 541,
        gid    => 541,
        home   => '/home/jeck',
        shell  => '/bin/bash',
    }
    @user {'lucy':
        ensure => present,
        uid    => 542,
        gid    => 'sysadmin',
        home   => '/home/lucy',
        shell  => '/bin/bash',
    }
}
```

上面的代码创建了 3 个虚拟用户及每个用户对应的属性值。



注意

用户组必须是已经存在的 gid。

3) 创建用户组类 user::dba 的配置文件 dba.pp，使用 realize 实例化用户 seven 和 lucy，代码如下：

```
$ vim /etc/puppet/modules/user/manifests/dba.pp
```

```
class user::dba
{
    realize( User['seven'],
            User['lucy'] )
}
```

4) 创建用户组类 `user::sysadmin` 的配置文件 `sysadmin.pp`, 使用 `realize` 实例化用户 `jeck` 和 `lucy`, 代码如下:

```
$ vim /etc/puppet/modules/user/manifests/sysadmin.pp
class user::sysadmin
{
    realize( User['jeck'],
            User['lucy'] )
}
```

5) 在测试节点 `agent.domain.com` 的配置文件添加如下代码:

```
$ vim /etc/puppet/manifests/nodes/agent.domain.com.pp
include user::virtual
include user::dba
include user::sysadmin
# 如果只想应用 dba 组, 取消 include user::sysadmin
```

6) 在测试节点 `agent.domain.com` 中运行如下命令:

```
$ puppet agent --server puppet.domain.com
# 执行后会实例化相关的虚拟资源
```

在讲解虚拟资源时需要对 `define` 进行说明, `define` 也可以实现虚拟资源的需求。如果一个模块内有多个相同的定义, 则 `define` 是不二选择。但是在主机没有进行 `define` 引用时, Puppet 会编译所有 `define` 的定义, 默认执行所有 `define` 定义的资源, 这样会带来不必要的性能消耗。而只有在对虚拟资源进行实例化时 Puppet 才会进行编译。`defined type` 是 `class` 的类比事务, 它实际上就是可以多实例化的类。而虚拟资源实际上就是把资源定义的过程拆分成了两步, 就像类一样, 定义一次, 多次声明, 只存在一个实例。

比较虚拟资源与 `define` 二者可以归纳为如下两点:

- ☐ 功能类似
- ☐ 性能不同

6.3 常用资源的用法

Puppet 所支持的资源类型有 48 种, 而常用的资源才 8 种, 什么样的资源被归档为常用资源?

我们先来看一下运维工程师拿到已经安装好操作系统的服务器时, 都需要做什么: 新建用户及组管理→安装应用软件→修改应用软件配置文件→启用应用服务, 等等; 根据服务器

第 11 章

大规模 Nginx 集群部署方案

在当前 Web Server、透明代理等诸多软件满天下的时代，Nginx 无疑是个全能型选手，它既有出色的 Web 表现，也有杰出的代理功能，近几年已经赢得大多数系统管理员及各大公司的青睐。因此我们将选用 Nginx 作为大规则集群部署方案的软件，从多个角度与实用方面出发，结合 Puppet 快速完成大型方案的部署。

11.1 应用场景

系统管理员小明需要为公司的 100 多台 Web 服务器安装应用管理软件 Nginx。服务器分布在天津网通和广州电信两个运营商所在地。服务器操作系统类型有 Centos 和 FreeBSD 两种，并且服务器硬件配置不统一。

为应对公司的业务需求，系统管理员小明要求自己：能快速变更 Nginx 的配置文件，比如若有新增虚拟主机则应能在短时间内生效。考虑成本节约，希望在部署时 Nginx 配置文件能根据硬件的配置不同而变化，这样能大大提升服务器的承载，提升利用率。为提升安全性，Nginx 要求以 www 用户及组运行。Nginx 所产生的日志，每天凌晨进行压缩处理保证磁盘有足够的空闲空间。

11.2 场景需求分析

首先我们需要对小明的需求进行分析，接着再做规划与实施。需求分析在软件开发的前期显得尤为重要，它完成的好坏直接影响着软件开发的质量与进度，而在运维领域直接关系到项目的成败、项目实施的进展，以及后续的扩展性。可见需求分析在任何地方都很关键，因此我们将小明的需求拆解成日常变更、网络及架构、软件安装、软件配置、节点管理 5 个环节。下面将分别针对这 5 个环节进行分析。

11.2.1 日常变更分析

作为系统管理员需要不断关注软件漏洞、新软件特性，因此需要对软件进行定期升级。

小明同时还要分析定期增加虚拟主机、新服务器上线部署、服务器下线等变更。因此小明分析的日常变更可以概括为如下两大类：

□ 软件升级

□ 配置变更

软件安装与升级可以采用 `package` 资源进行管理，如果有更严格的控制，可以自己搭建软件仓库进行软件包管理。尽管搭建软件仓库会带来一定的维护成本，但它的好处还是很明显的，比如可用性、扩展性、速度都能得到保障。

配置变更可以采用 `Puppet file` 文件资源同步虚拟主机目录的方式进行。每增加一个虚拟主机就在主机目录生成相应虚拟主机配置文件，采用 `File` 资源同步此目录完成更新。也可以采用 `define` 或虚拟资源进行虚拟主机的配置管理。每次更新时采用 `notify` 触发 `exec` 资源执行 `reload` 动作。在虚拟主机比较多且大部分虚拟主机配置不相同，建议采用一个虚拟机一个文件的方式。如果每个虚拟主机的配置文件都相同，可以定义好固定的模板生成。

这里需要读者熟悉 `Puppet` 的资源用法，这样才能灵活运用。不熟悉也没有关系，笔者将通过逐步操作完成需求，进而使大家熟练掌握 `Puppet` 相关资源的用法。

11.2.2 网络及架构分析

由于服务器分属天津网通和广州电信。跨运营商网络质量存在极其不稳定的因素，如果将中心端部署在天津网通节点会导致广州电信客户端请求延时，大部分资源将消耗在网络上。中国的网络结构可以简要概括为：以北京、上海、广州为超级核心，各运营商建立互联互通设备，通过这些互联互通设备与其他运营商进行流量互访，各运营商再建立自己的核心网络。再加上各运营商之间的计费方式不同，导致不同运营商之间网络有所延迟，不同的接入层依路由的跳数和其他影响的不同而导致延迟与丢包各有所不同。

为避免网络质量对整体架构的影响，需要考虑选择 `BGP` 机房或者在网通和电信各部署一组管理机。如果选择 `BGP` 机房则当 `BGP` 机房出现故障时，将成为架构的单点，因此笔者推荐在网通和电信各部署一组管理机。尽管这样会带来成本的增加，但做到了异地容灾，即一个机房出现故障时，另一个机房勉强能用。两个机房之间的解析和调度可以依赖 `DNS` 划分两个 `view` 来解决。跨运营商的 `Puppet` 架构图如图 11-1 所示。

部署两组管理机带来的是难度系数的增加（认证颁发管理、`report` 的配置），服务器也会达到一定规模。为缓解单台及管理机 `Puppet Master` 的压力，可以在两个 IDC 上各部署多台 `Master`，前端采用 `Nginx` 或 `Apache` 做代理，即两个 IDC 各部署一组服务器。详细配置可参考第 15 章对 `Puppet` 架构扩展及分布式部署的介绍。

为实现负载均衡，另一种较好的方式是采用 `DNS` 轮循。目前 `DNS` 应用已经非常成熟，基于公司内部使用，我们可以自己搭建 `DNS` 服务器，也可以采用目前非常稳定的 `DNSPod` 进行分地域解析，每台 `Puppet Master` 采用同相的安装与配置方法，通过 `DNS` 轮到不同的 `Puppet Master` 主机，为扩展 `Puppet Master` 提供了便利。

对于这两种不同的方案，读者可以根据自己的环境进行选择。

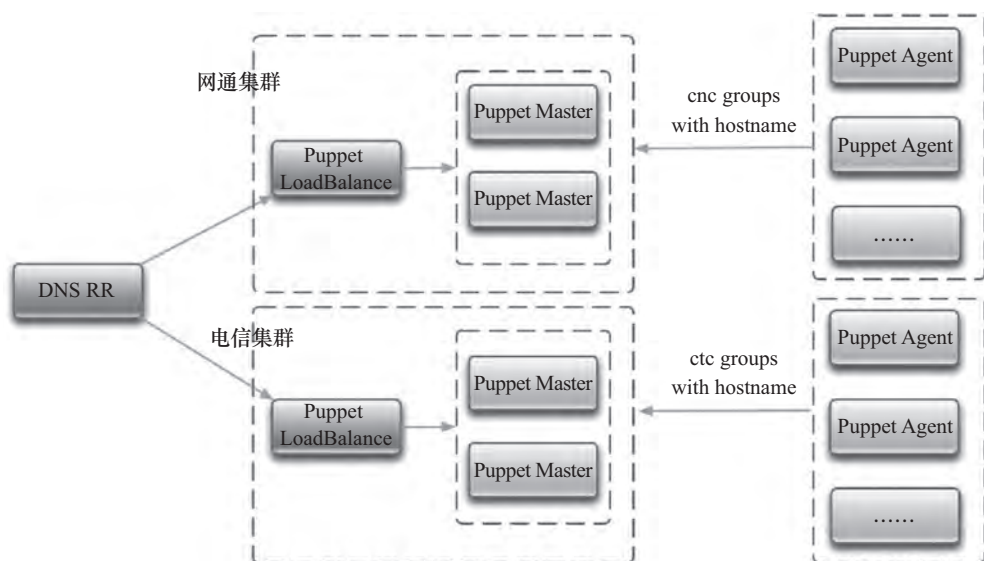


图 11-1 跨运营商的 Puppet 架构部署示意图

11.2.3 软件安装分析

软件安装时可以采用系统自带的软件包管理进行安装。一般软件包都能从官方源或 EPEL 源找得到。如果涉及过多的软件或软件版本则需要自行控制，可以在本地搭建软件仓库，用来管理所有软件包及其依赖。这样做还可以提升软件包部署速度。缺点就是操作系统不统一时软件仓库维护复杂，不但要维护多个软件包，并且依赖性处理起来比较棘手。

下面介绍一下自建软件源的优缺点。

具体优点如下：

- ☐ 管理方便；
- ☐ 部署快；
- ☐ 灰度发布，回滚便捷；
- ☐ 开发软件包统一管理。

缺点是多系统维护复杂。

客户端软件安装时采用 Puppet 的 package 资源，通过 case 或 selection 条件语句确定 \$operatingsystem，即操作系统类型进行软件安装。Nginx 在 Centos 与 FreeBSD 中的软件包名称相同，因此可以不采用条件语句。直接采用系统自带的软件安装命令，Puppet 在 Centos 会调用 yum，在 FreeBSD 上会调用 port。

11.2.4 软件配置分析

为达到设备利用率最大化，运行用户的安全限制，在 nginx 配置文件 nginx.conf 中配置系统变量时采用 facter 命令获取到的系统变量进行配置。Nginx 系统进程数可以使用 ERB 模

板中的 `<%=processorcount%>` 配置。由于需要对日志进行切割处理，所以需要利用 File 资源与 cron 资源创建定时任务。虚拟主机的配置可以采用目录的方式进行推送，也可以采用 `nginx.conf` 追加 `server` 的方式进行配置。笔者推荐采用 `include` 虚拟主机的方式进行管理，这样规范的好处就是将主配置文件与虚拟主机文件分开，并且一个虚拟主机配置文件错误不会影响所有配置文件，也方便系统管理员排查。

我们将讲解如何通过编写模块和使用开源模块两种方法实现 Nginx 软件部署需求。

- ❑ 编写模块可采用在 `nginx.conf` 中包含虚拟主机目录的方式进行虚拟主机管理，对 File 资源同步目录进行更新。每次更新时采用 `notify` 触发 `exec` 资源执行 `reload` 动作。
- ❑ 采用 Puppetlabs 上提供的开源模块，该模块采用 `default` 的方式实现虚拟主机的管理。

11.2.5 节点管理分析

考虑到服务器数量比较多，所有主机名统一采用 DNS 进行管理。划分两个 view 将网通的主机名解析到网通管理机，将电信的主机名解析到电信管理机。主机名采用如下规范：

运营商简称．地区．角色．服务名．IP 地址．域名

例如：

```
cnc.tj.web.nginx.ip.domain.com
ctc.gz.web.nginx.ip.domain.com
```

节点管理所有主机名时分两个节点目录：

```
mkdir /etc/puppet/manifests/nodes/{cnc,ctc}
```

所有的服务器都采用相同的配置，因此可以采用 `default`、`basenode`、`general` 三种方式进行节点的全局定义。在配置变量时可以区别运营商，因此可以采用正则匹配相同运营商的主机。

详情可参考第 8 章。

11.3 合理规划

11.3.1 系统安装

服务器操作系统的安装可以利用 Cobbler 进行。Cobbler 是一个集成工具，集成了 PXE、DHCP、DNS 和 Kickstart 服务管理，以及 yum 仓库管理。Cobbler 的目的是为了实现快速网络安装环境的 Linux 安装服务器，可以为数量众多的 Linux 服务自动化执行任务。Cobbler 的官方网站：<https://fedorahosted.org/cobbler/>。

利用 Cobbler 可以进行操作系统定制化，将 Puppet Agent 软件包集成在操作系统当中。众所周知，要使用 Puppet 必须先安装客户端程序、配置好主机名、完成认证，否则服务端将查找不到客户端，无法实现客户端的管理。因此我们需要将这些都在系统安装时进行集成。

最为有效的方式是在服务器出厂时通过 SN 号对该服务器进行盘点号唯一化，通过系统后台配置好该服务器的角色、SN 号、主机名等信息。当服务器使用 Cobbler 安装完并启动时，定制化程序通过后台 API 获取该主机信息，进行主机名配置及 Puppet 客户端程序安装。最后运行 puppet agent 命令完成认证。

这时一台服务器就已经加入到我们的管理队列当中。剩下的所有事件都可以交给 Puppet 来完成。

如果并没有实现 Cobbler 和系统定制化，却想要将现有的服务器加入 Puppet 管理时，可以通过 Python 组合 expect 完成主机名配置和 Puppet 客户端的安装，或者使用最熟悉的、线上正在运行的命令运行相关的软件或工具来实现。

11.3.2 系统初始化

当服务器操作系统安装完成后，要想使用系统提供的服务还需要进行系统的初始化。这样做的目的有多个，而更多的是出于便捷性管理、安全等角度的考虑。系统初始化可以归类为如下几大类。

- ❑ 系统配置：ntp、limits、sysconfig、sysctl。
- ❑ 用户管理：user、group。
- ❑ 安全配置：iptables。
- ❑ 登录配置：sshd。

这些我们可以直接使用官方 Forge 提供的模块 erwbgy-system 和 puppetlabs-firewall 来实现。

11.3.3 部署规划

结合所有的需求，经过深入的分析，需要对 Puppet 的部署进行合理规划。为解决快速认证，需要配置自动认证，当有紧急更新时需要使用 kick 命令，在 Puppet 3.x 中建议采用 MCollective，详见第 20 章。为避免执行 kick 时全网资源全部更新，需要配置标签。要掌握所有的客户端情况，需要配置报告系统。Puppet 服务器的目录按编写 nginx 模块的形式进行图 11-2 所示的规划。

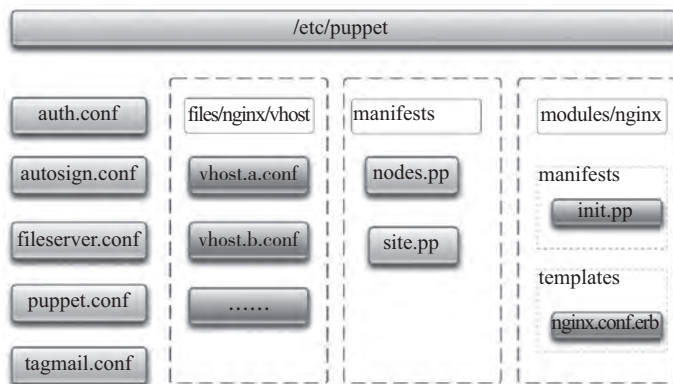


图 11-2 Puppet Master 目录结构规划图

nginx 模块的虚拟主机文件也可以统一放在 modules/nginx/files 中。使用统一文件目录的目的是便于统一管理。读者可以根据自己的特征选择自己的统一管理模式。

11.3.4 关注点

我们再来思考一下系统管理员小明还有哪些关注点，或者说需要思考哪些问题：

- ☐ 服务部署情况，及是否部署成功？
- ☐ 系统是否支持扩展？
- ☐ 单个 Master 支持多少客户端请求？什么时候需要扩容？
- ☐ 系统性能的瓶颈有哪些？
- ☐ 紧急触发客户端更新时，服务端是否能支撑？
- ☐ 如何实现灰度发布？

相关问题对应的策略为：

- ☐ 部署 Puppet 控制台，用于查看客户端部署情况（参考第 18 章）。
- ☐ 采用分布式架构（参考第 15 章）。
- ☐ 通过 Puppet 控制台进行性能分析，优化 Puppet 资源（参考第 6 章）。
- ☐ 触发客户端更新时，采用分组分批次进行，避免 Puppet Master 压力过大而崩溃。
- ☐ 利用 Puppet environment 多环境配置，实现灰度发布（参考第 4 章）。

如果按照官方语言风格及笔者的建议编写模块，或采用官方提供的模块，则单台 Puppet Master 服务器在采用 Web 替代 WEBRick 的情况下将支持 500 台左右的客户端。当然这也依赖于服务器硬件配置及资源依赖等多方面因素。我们可以以此为参照，根据实际环境需求来决定采用 Puppet Master 部署的数量。

11.4 实施步骤

在完成整体规划后，就可以按规划的方案来实施了。实施主要从前期准备搭建环境开始。我们将配置自己的软件仓库进行软件管理，并采用 yumrepo 资源指定到自己的软件源。在进行 Puppet Master 主配置时配置自动认证、权限等，最后创建 nginx 模块，虚拟主机采用每个虚拟主机一个配置文件模式，或者使用官方 nginx 模块进行 Nginx 安装与配置。

本节都以 Centos5.4 64bit 为例进行讲解。

11.4.1 前期准备：创建软件仓库

搭建自己的软件仓库，进行软件包自定义、升级、控制等。当服务器具有一定规模时建议搭建自己的软件仓库。我们安装的 Nginx 来源于官网或 epel 源，因此可以使用 yum downloadonly 软件（downloadonly 为 Yum 的插件，默认不安装）只下载软件包，并不安装，再采用 createrepo 创建软件仓库。这样能有效解决软件包依赖的问题。详细配置过程如下：

CPU 与内存的消耗。我们可以通过判定 Puppet Master 编译清单时长来决定是否需要进行扩容。在通常情况下，单台 Puppet Master 在使用 WEBRick 启用时能支持百台规模。当然这也依赖于服务器的硬件配置、资源的类型和数量、文件资源数量。目前也有人拿一台虚拟机当做 Puppet Master 完成近千台 Hadoop 集群的管理。

15.1.2 认证的瓶颈

Puppet 管理所有客户端都是采用颁发证书的方式进行的，并且每次客户端发起请求时都会到 Puppet Master 上验证一次该客户端是否已授权，否则将会拒绝该请求。当服务器规模较大时，Puppet CA 也会成为瓶颈，但 Puppet CA 的负载总体不如 Master 大，扩展时有两种方案可以实现：

- ❑ 每台 Master 都接受 CA 请求，实时同步证书目录，并指定某台机器作为颁发证书服务器。
 - ❑ 独立 CA 服务器，只处理 CA 请求，采用另一台机器进行 CA 热备。
- 两种方法各有优缺点，但都能完成扩展需求。

15.1.3 文件的瓶颈

Puppet 只是配置管理工具，有些读者会使用 Puppet 下发大量文件，除了应用软件配置的 File 资源外，还有 ERB 模板文件，也有目录同步等，因此会造成 Puppet Master 瓶颈，进而导致处理时间过长，影响效率。据笔者观察，很多 Puppet 环境中 60% 左右的效率都消耗在 File 资源上。为此在第 16 章将重点讲解 File 资源的优化。

15.1.4 网路的瓶颈

中国的网络可以说是全世界最复杂的，“搞定了中国网络，就等于征服了全世界的网络”，这话一点也不为过。我们每天都在努力奋斗，处理着各种网络带来的问题。当我们优化了一个又一个环节，扩容了一切能扩容的，最终会发现因为网络瓶颈而无法完成部署，这是多么可悲的事情啊。为避免这种情况，我们需要在各大运营端节点部署 Puppet Master，以保证效率。

而在选择部署 Master 节点时，可以采用开源的 smokeping 来采集各节点之间的网络延迟及丢包。选择最好的节点进行 Master 部署，以保证连接至该 Master 节点的可靠性、稳定性及效率。

15.2 架构扩展之单台 Puppet Master

Puppet Master 架构扩展可以从单台扩展、集群架构、分布式三个方面进行。我们首先要讲的就是单台 Puppet Master 的扩展。采用最常用的 Web 服务器 Apache 或 Nginx 来代替 Puppet 原生态的 WEBRick 以提升 Master 吞吐性能。在管理机上启动 WebServer 以负责监听 8140 端口并处理客户端请求、file 文件以及已验证的客户端请求；将编译部分代理转发到后端 Master。Mongrel 启动多个 Master 实例。而 Passenger 会随 WebServer 启动而动态启动多

个 Master 实例。Mongrel 与 Passenger 有着一定的差异，但 Mongrel 将会被淘汰。采用 Web 扩展架构单台可以承载大概 500 个节点。

目前比较流行的 Web 代码组合模式有：

- ❑ Nginx+Mongrel
- ❑ Apache+Passenger
- ❑ Nginx+Passenger

如上三种扩展模式的架构示意如图 15-1 所示。

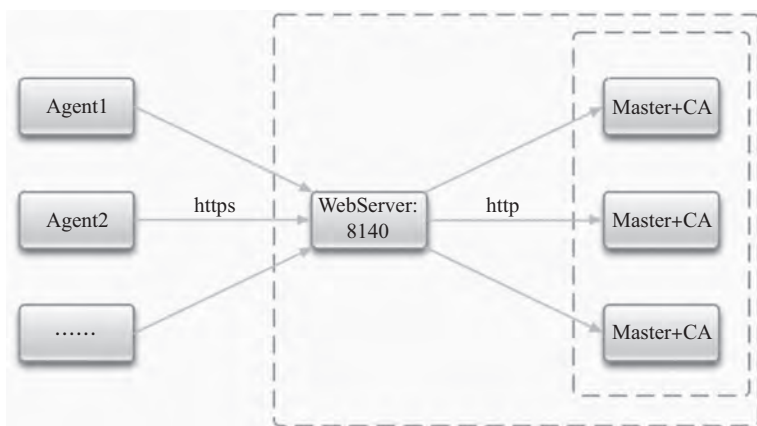


图 15-1 Puppet 单台扩展 WebServer 的代理架构示意

15.2.1 Nginx+Mongrel 模式

推荐使用 Nginx 的理由是它比 Apache 并发能力更强、处理文件效率更高、转发请求也优于 Apache。Mongrel 可以启用多个 Master 实例，解决 WEBRick 只需使用一个进程多线程模式，以实现更高的并发与处理能力。值得注意的是，Puppet 3.0 已取消 Mongrel，但我们仍可以继续使用 Nginx 作为前端，详情请参考 15.2.3 节。为照顾还在使用老版本的读者，这里简要说明其安装、配置过程。

我们将以 Puppet 2.7 Centos 5.4 X86_64 为例讲解此模式的安装、配置过程。

1) 安装 Nginx 和 Mongrel。

安装 Nginx 的方法如下：

```
$ sudo yum -y install ruby-devel pcre pcre-devel openssl zlib nginx
```

采用源码包安装 Nginx。从 Nginx 官网 (<http://nginx.org/>) 下载 Nginx 源码包并手动编译支持 SSL，具体如下：

```
$ cd /usr/local/src
$ wget http://nginx.org/download/nginx-1.3.11.tar.gz
$ tar xzf nginx-1.3.11.tar.gz
$ cd nginx-1.3.11
```

```
$ ./configure --user=daemon --group=daemon --prefix=/usr/local/nginx/ --with-http_stub_
status_module --with-http_ssl_module --with-http_sub_module --with-md5=/usr/lib --with-
sha1=/usr/lib --with-http_gzip_static_module
$ sudo make && make install
```

安装 Mongrel 的方法如下：

```
$ sudo gem install mongrel
```

2) 修改 `/etc/sysconfig/puppetmaster` 文件。Puppet Master 在启动时会加载此文件，读取配置信息，并追加如下内容，让 Master 启动 4 个进程，监听 4 个端口，指定 `servertype` 为 Mongrel 模式：

```
$ sudo vim /etc/sysconfig/puppetmaster
PUPPETMASTER_PORTS=( 18140 18141 18142 18143 )
PUPPETMASTER_EXTRA_OPTS="--servertype=mongrel --ssl_client_header=HTTP_X_SSL_
SUBJECT"
```

3) 创建 `puppet.domain.com.conf` 文件，具体如下：

```
$ vim /usr/local/nginx/conf/vhosts/puppet.domain.com.conf
upstream puppetmaster {
    server 127.0.0.1:18140;
    server 127.0.0.1:18141;
    server 127.0.0.1:18142;
    server 127.0.0.1:18143;
}

server {
    listen 8140;
    root                                /etc/puppet;

    ssl                                 on;
    ssl_session_timeout                 5m;
    ssl_certificate                     /var/lib/puppet/ssl/certs/puppet.domain.com.pem;
    ssl_certificate_key                 /var/lib/puppet/ssl/private_keys/puppet.domain.com.pem;
    ssl_client_certificate              /var/lib/puppet/ssl/ca/ca.crt.pem;
    ssl_crl                            /var/lib/puppet/ssl/ca/ca_crl.pem;
    ssl_verify_client                  optional;

    # File sections
    location /production/file_content/files/ {
        types { }
        default_type application/x-raw
        alias /etc/puppet/files/;
    }

    # Modules files sections
    location ~ /production/file_content/modules/.+/ {
        root /etc/puppet/modules;
        types { }
    }
}
```

```

    default_type application/x-raw;
    rewrite
    ^/production/file_content/modules/([^/]+)/(.+)$ /$1/files/$2 break;
}

# Ask the puppetmaster for everything else
location / {
    proxy_pass            http://puppetmaster;
    proxy_redirect        off;
    proxy_set_header      Host                $host;
    proxy_set_header      X-Real-IP           $remote_addr;
    proxy_set_header      X-Forwarded-For     $proxy_add_x_forwarded_for;
    proxy_set_header      X-Client-Verify     $ssl_client_verify;
    proxy_set_header      X-SSL-Subject       $ssl_client_s_dn;
    proxy_set_header      X-SSL-Issuer        $ssl_client_i_dn;
    proxy_buffer_size     16k;
    proxy_buffers          8 32k;
    proxy_busy_buffers_size 64k;
    proxy_temp_file_write_size 64k;
    proxy_read_timeout     65;
}
}

```

以上代码中定义了：

- ❑ **Upstream**。每个实例都在配置 Mongrel 时进行了定义。参考步骤 1)。
- ❑ **ssl on**。开启 SSL 服务。
- ❑ **ssl_certificate**。配置 PuppetMaster CA 相关目录，用于处理验证 CA 请求。
- ❑ **location files**。处理文件请求，注意 alias 别名服务，可以将所有文件统一至此目录后交予 Nginx 处理。
- ❑ **location modules**。Puppet 模块目录。
- ❑ **location /**。转发 Puppet Master 实例请求。

4) 修改 nginx.conf，增加如下配置：

```

$ sudo/usr/local/nginx/conf/nginx.conf
include      vhosts/*.conf;

```

5) 重启 Puppet Master，方法如下：

```

$ sudo/etc/init.d/puppetmaster restart
Stopping puppetmaster:
Port: 18140 [FAILED]
Port: 18141 [FAILED]
Port: 18142 [FAILED]
Port: 18143 [FAILED]
Starting puppetmaster:
Port: 18140 [ OK ]
Port: 18141 [ OK ]
Port: 18142 [ OK ]
Port: 18143 [ OK ]

```

6) 启用 Nginx，方法如下：

```
$ sudo/etc/init.d/nginx start
Starting nginx: [ OK ]
```

7) 确认端口已启动，方法如下：

```
$ sudo netstat -ntlp |grep ruby
tcp      0      0 127.0.0.1:18140      0.0.0.0:* LISTEN    7492/ruby
tcp      0      0 127.0.0.1:18141      0.0.0.0:* LISTEN    7521/ruby
tcp      0      0 127.0.0.1:18142      0.0.0.0:* LISTEN    7550/ruby
tcp      0      0 127.0.0.1:18143      0.0.0.0:* LISTEN    7579/ruby
tcp      0      0 0.0.0.0:3000         0.0.0.0:* LISTEN    5830/ruby
```

8) 在 Puppet 客户端运行 puppet 命令检测配置是否正确，方法如下：

```
$ sudo puppet agent --server puppet.domain.com --test --noop
... ..
notice: Finished catalog run in 3.04 seconds
```

Puppet Agent 并没有任务提示表示 Puppet Master 服务已经从 WEBRick 切换到了 Nginx。为了进一步验证配置的正确性，我们需要查看 Master 的日志与 Nginx 服务的日志。

在 Master 端查看 Puppet 服务的日志：

```
$ tail /var/log/message
(//agent.domain.com/Puppet) Finished catalog run in 3.04 seconds
```

在 Master 端查看 Nginx 服务的日志：

```
$ tail /var/log/nginx/access.log
10.210.213.219 - - [26/Jan/2013:21:51:06 +0800] "POST /production/catalog/
agent.domain.com HTTP/1.1" 200 6805 "-" "-" "-"
10.210.213.219 - - [26/Jan/2013:21:51:09 +0800] "GET /production/file_metadata/
modules/nginx/nginx_logrote.sh? HTTP/1.1" 200 304 "-" "-" "-"
10.210.213.219 - - [26/Jan/2013:21:51:10 +0800] "PUT /production/report/agent.
domain.com HTTP/1.1" 200 31 "-" "-" "-"
```

到此就配置好了使用 Mongrel 替代 WEBRick 的单台 Puppet Master 扩展。

15.2.2 Apache+Passenger 模式

Passenger 是一个用于将 Ruby 程序嵌入执行的 Apache 2.x 的一个模块，它可以让你运行 Rails，即 Rack 应用内的一个 Web 服务器。Rails 官方推荐的最佳模式为：Apache+Passenger。此模式拥有极其强大的功能，能够自动增减集群进程的数量、对后续的部署显得非常简单，为此 Puppetlabs 也推荐使用此配置。但在更多情况下我们都采用 Nginx+Passenger 模式。在使用 Passenger 扩展模式时不需要单独启动 Puppet Master 服务。

首先我们需要安装好 Apache 和 Passenger，然后配置 Apache 处理 Puppet Agent 的 SSL 验证请求，最后将 Apache 连接到 Puppet Master。在处理 SSL 验证请求时，Apache 会验证 Puppet Agent 的证书是否由 Puppet CA 签发，Apache 会先验证请求。如果授权通过，则调用

Master。与此同时，Apache 会提供给 Puppet Agent 一个证书用于验证服务端的真实证，再将 SSL 证书存放在适当的位置。打开 Passenger 模块并为 Puppet Master 服务创建一个虚拟主机来配置 Apache。

在理解此模式的工作流原理后，配置起来将非常容易。我们以 Centos 5、Puppet 2.7 为例详解讲解安装配置过程。

1) 安装 Apache，方法如下：

```
$ sudo yum install httpd httpd-devel mod_ssl ruby-devel rubygems
```

2) 安装 Passenger，方法如下：

```
$ sudo gem install rack passenger
Successfully installed rack-1.5.0
Successfully installed daemon_controller-1.1.1
Successfully installed passenger-3.0.19
3 gems installed
Installing ri documentation for rack-1.5.0...
Installing ri documentation for daemon_controller-1.1.1...
Installing ri documentation for passenger-3.0.19...
Installing RDoc documentation for rack-1.5.0...
Installing RDoc documentation for daemon_controller-1.1.1...
Installing RDoc documentation for passenger-3.0.19...

$ sudo passenger-install-apache2-module
```



提示

官方镜像经常出现因网络而引起莫名其妙的问題，可以考虑采用淘宝免费提供的镜像。

建议更换 Gem 镜像，再安装 Passenger，方法如下：

```
$ gem sources --remove https://rubygems.org/
$ gem sources --remove http://rubygems.org/
$ gem sources -a http://ruby.taobao.org/
$ gem sources -l
*** CURRENT SOURCES ***

http://ruby.taobao.org
# 请确保只有 ruby.taobao.org
$ gem install rails
```

运行 passenger-install-apache2-module 后，按回车键，默认会在大量输出提示后显示如何配置 Apache 加载 Passenger 模块。

本例的提示如下，在配置 Apache 虚拟主机时需要增加的 Passenger 模块配置文件，具体如下：

```
LoadModule passenger_module /usr/lib64/ruby/gems/1.8/gems/passenger-3.0.19/
ext/apache2/mod_passenger.so
PassengerRoot /usr/lib64/ruby/gems/1.8/gems/passenger-3.0.19
PassengerRuby /usr/bin/ruby
```

**提示**

每次安装时 Passenger 版本可能会发生变化，所以不要直接复制使用。

如果不知道 PassengerRoot 目录，可以使用如下命令查看：

```
$ passenger-config --root
```

3) 配置 Apache。

需要在 Puppet Master 创建 Rack 应用，创建一个目录用来存放 config.ru 配置文件，并创建一个虚拟主机配置文件。Rack 为 Web 服务器提供了用来和 Puppet 服务交换请求和响应的一些常用 API。Rack 适用于 Ruby 类的 HTTP 服务，可以用于多台服务器之间部署服务。默认我们采用软件维护者提供的 config.ru 配置文件。

创建 Rack 框架目录，并将 config.ru 文件复制至该目录，并赋予该文件 puppet 用户权限，通常 Puppet 以 puppet 用户启动，为此需要赋予“config.ru”以 puppet 用户权限。具体如下：

```
$ sudo mkdir -p /usr/share/puppet/rack/puppetmasterd
$ sudo mkdir /usr/share/puppet/rack/puppetmasterd/public /usr/share/puppet/
rack/puppetmasterd/tmp
$ sudo cp /usr/share/puppet/ext/rack/files/config.ru /usr/share/puppet/rack/
puppetmasterd/
$ sudo chown puppet /usr/share/puppet/rack/puppetmasterd/config.ru
```

创建 Apache 虚拟主机文件，具体如下：

```
$ vim /etc/httpd/conf.d/puppet.domain.com.conf
LoadModule passenger_module /usr/lib64/ruby/gems/1.8/gems/passenger-3.0.19/ext/
apache2/mod_passenger.so
PassengerRoot /usr/lib64/ruby/gems/1.8/gems/passenger-3.0.19
PassengerRuby /usr/bin/ruby
# And the passenger performance tuning settings:
PassengerHighPerformance On
PassengerUseGlobalQueue On
# Set this to about 1.5 times the number of CPU cores in your master:
PassengerMaxPoolSize 12
# Recycle master processes after they service 1000 requests
PassengerMaxRequests 1000
# Stop processes if they sit idle for 10 minutes
PassengerPoolIdleTime 600

Listen 8140
<VirtualHost *:8140>
    SSLEngine On

    # Only allow high security cryptography. Alter if needed for compatibility.
    SSLProtocol All -SSLv2
    SSLCipherSuite HIGH:!ADH:RC4+RSA:-MEDIUM:-LOW:-EXP
    SSLCertificateFile /var/lib/puppet/ssl/certs/puppet-server.example.com.pem
    SSLCertificateKeyFile /var/lib/puppet/ssl/private_keys/puppet-server.example.com.pem
```



```

SSLCertificateChainFile /var/lib/puppet/ssl/ca/ca.crt.pem
SSLCACertificateFile    /var/lib/puppet/ssl/ca/ca.crt.pem
SSLCARevocationFile    /var/lib/puppet/ssl/ca/ca.crl.pem
SSLVerifyClient         optional
SSLVerifyDepth          1
SSLOptions              +StdEnvVars +ExportCertData

# These request headers are used to pass the client certificate
# authentication information on to the puppet master process
RequestHeader set X-SSL-Subject %{SSL_CLIENT_S_DN}e
RequestHeader set X-Client-DN  %{SSL_CLIENT_S_DN}e
RequestHeader set X-Client-Verify %{SSL_CLIENT_VERIFY}e

RackAutoDetect On
DocumentRoot /usr/share/puppet/rack/puppetmasterd/public/
<Directory /usr/share/puppet/rack/puppetmasterd/>
    Options None
    AllowOverride None
    Order Allow,Deny
    Allow from All
</Directory>
</VirtualHost>

```

以上代码中定义了：

- ❑ 加载 Passenger 模块及与模块相关的文件定义。
- ❑ 虚拟主机默认监听 8140，在后续扩展时可以监控其他端口释放 8140。
- ❑ SSL 相关的配置，打开 SSL 并指定 Master 证书。需要确保此配置为 Puppet Master 的主机名或正式域名。
- ❑ RequestHeader 请求的 Header 信息。
- ❑ 如果所配置的 SSL 目录不一样，需要修改如下几行代码：

```

SSLCertificateChainFile /var/lib/puppet/ssl/certs/ca.pem
SSLCACertificateFile    /var/lib/puppet/ssl/certs/ca.pem
SSLCARevocationFile    /var/lib/puppet/ssl/crl.pem

```

4) 启动 Apache。

我们已经完成了 Apache 和 Passenger 的安装和配置。现在可以启动 Apache 服务来准备测试刚才的变更了，具体如下：

```

$ sudo /etc/init.d/httpd start
Starting httpd: [ OK ]

```

5) 确认端口及进程。

启动 Apache 服务后最好再确认一下端口与进程是否吻合，具体如下：

```

$ sudo netstat -ntlp |grep httpd
tcp        0      0 0.0.0.0:8140          0.0.0.0:*             LISTEN      16955/httpd

```

6) 在 Puppet 客户端运行 puppet 命令检测配置是否正确，具体如下：

```
$ sudo puppet agent --server puppet.domain.com --test -noop
... ..
notice: Finished catalog run in 3.04 seconds
```

Puppet Agent 并没有任务提示表示 Puppet Master 服务已经从 WEBRick 切换到了 Apache, 为了进一步验证配置的正确性, 我们需要查看 Master 的日志与 Apache 服务的日志。

在 Master 端查看相应的日志:

```
$ tail /var/log/message
(//agent.domain.com/Puppet) Finished catalog run in 3.04 seconds
```

在 Master 端查看 Apache 服务的日志:

```
$ tail /var/log/httpd/access_log
10.210.213.219 - - [26/Jan/2013:23:35:02 +0800] "POST /production/catalog/agent.
domain.com HTTP/1.1" 200 6805 "-" "-"
10.210.213.219 - - [26/Jan/2013:23:35:13 +0800] "GET /production/file_metadata/modules/
nginx/nginx_logrote.sh? HTTP/1.1" 200 304 "-" "-"
10.210.213.219 - - [26/Jan/2013:23:35:13 +0800] "PUT /production/report/agent.domain.com
HTTP/1.1" 200 31 "-" "-"
```

在 access_log 文件中我们看到, Puppet Agent 使用 URI “/production/catalog/agent.domain.com” 发出一个 HTTP POST 请求。请求状态码为 200, 表示成功。GET 请求为文件请求, PUT 请求表示 Puppet Agent 提交了一个报告。

目前的配置已经为 Puppet Master 添加了一个 Apache 服务和 Passenger 前端, 它们能极大地扩展 Master 能够管理的节点的数量。

通过 Passenger 相关的命令, 可以查看服务器相应的状态信息, 使用 status 可以看到当前的 App 为 puppetmasterd、PID 为 17098、进程数为 3、启动时间为 1 分 3 秒。

7) 查看 Passenger 状态, 方法如下:

```
$ sudopassenger-status
----- General information -----
max          = 12
count        = 1
active       = 0
inactive     = 1
Waiting on global queue: 0

----- Application groups -----
/usr/share/puppet/rack/puppetmasterd:
  App root: /usr/share/puppet/rack/puppetmasterd
  * PID: 17098   Sessions: 0   Processed: 3   Uptime: 1m 3s
```

也可以使用 passenger-memory-stats 命令查看内存相关的状态。

15.2.3 Nginx+Passenger 模式

Puppet 3.0 以上版本不再支持 Mongrel 模式, 读者肯定会问为什么不采用并发处理能力

更强、性能更优异的 Nginx 取代 Apache 呢？答案是可以的，Passenger 既然可以作为 Apache 的模块，同样也可以作为 Nginx 的模块。Passenger 文档也有对应的 Nginx 说明。

下面以 Centos 5 为例讲解配置步骤。

1) 安装 Passenger 和 Nginx，方法如下：

```
$ sudo yum -y install ruby-devel rubygems
$ sudo gem install rake rack passenger
# 运行 passenger-install-nginx-module 命令可以选择自动安装 Nginx 和 Passenger，
# 命令会给出两个选项
$ sudo passenger-install-nginx-module
1. Yes: download, compile and install Nginx for me. (recommended)
2. No: I want to customize my Nginx installation. (for advanced users)
# 在选择 1 的情况下，会自动下载 Nginx 并安装，选择 2 的情况下会提示
Please specify the directory: /usr/local/src/nginx-1.3.11 # 输入 nginx 源代码目录
Please specify a prefix directory [/opt/nginx]:          # 选择安装路径
sh ./configure --prefix='/opt/nginx' --with-http_ssl_module --with-http_gzip_static_module
--with-cc-opt='-Wno-error' --add-module='/usr/lib64/ruby/gems/1.8/gems/passenger
-3.0.19/ext/nginx'# 等待编译安装完成
```

使用以上命令编译安装完成后会提示 Nginx 加载 Passenger 模块的用法如下：

```
http {
    ...
    passenger_root /usr/lib64/ruby/gems/1.8/gems/passenger-3.0.19;
    passenger_ruby /usr/bin/ruby;
    ...
}
```

2) 手工编译安装 Nginx，并加载 Passenger 模块，具体如下：

```
$ cd /usr/local/src
$ wget http://nginx.org/download/nginx-1.3.11.tar.gz
$ tar xzf nginx-1.3.11.tar.gz
$ cd nginx-1.3.11
$ ./configure --user=daemon --group=daemon --prefix=/usr/local/nginx_passenger/ --with-
http_stub_status_module --with-http_ssl_module --with-http_sub_module --with-md5=/usr/
lib --with-sha1=/usr/lib --with-http_gzip_static_module--add-module='/usr/lib64/ruby/
gems/1.8/gems/passenger-3.0.19/ext/nginx'
#add-modules 添加 Passenger 模块
$ sudo make && make install
```

3) 创建 rack 目录并配置 Rack 应用，具体如下：

```
$ sudo mkdir -p /etc/puppet/rack/public
$ cp /usr/share/puppet/ext/rack/files/config.ru /etc/puppet/rack/
$ sudo chown -R puppet:puppet /etc/puppet/rack/
```

4) 配置 Nginx。

修改 nginx.conf，添加 Passenger 模块配置，本例添加的内容如下：

```
# Passenger needed for puppet
passenger_root /usr/lib64/ruby/gems/1.8/gems/passenger-3.0.19;
passenger_ruby /usr/bin/ruby;
passenger_max_pool_size 15;

include vhosts/*.conf;
```

passenger_root 目录可以使用 passenger-config--root 命令查看。

添加 Puppet 对应的 Nginx 虚拟主机, puppet.domain.com.conf 内容如下:

```
server {
    listen                8140 ssl;
    server_name           puppet puppet.domain.com;

    passenger_enabled     on;
    passenger_set_cgi_param HTTP_X_CLIENT_DN $ssl_client_s_dn;
    passenger_set_cgi_param HTTP_X_CLIENT_VERIFY $ssl_client_verify;

    access_log            /var/log/nginx/puppet_access.log;
    error_log             /var/log/nginx/puppet_error.log;

    root                  /etc/puppet/rack/public;

    ssl_certificate        /var/lib/puppet/ssl/certs/puppet.domain.com.pem;
    ssl_certificate_key    /var/lib/puppet/ssl/private_keys/puppet.domain.com.pem;
    ssl_crl                /var/lib/puppet/ssl/ca/ca_crl.pem;
    ssl_client_certificate /var/lib/puppet/ssl/certs/ca.pem;
    ssl_ciphers            SSLv2:-LOW:-EXPORT:RC4+RSA;
    ssl_prefer_server_ciphers on;
    ssl_verify_client      optional;
    ssl_verify_depth       1;
    ssl_session_cache      shared:SSL:128m;
    ssl_session_timeout    5m;
}
```

以上代码中定义了:

- Passenger 配置, 默认不需要修改。
- SSL 配置, 需要修改至自己的环境。

到此为止, 我们就实现了 Nginx+Passenger 模式的架构。在使用 Passenger 时, 不管前端是 Apache 还是 Nginx, passenger_max_pool_size 默认 10 个, 这是应用程序实例可以同时激活的最大数量。如果服务器性能足够好, 又想控制 Puppet 占用系统内存在 1GB 以内, 可以设置 passenger_max_pool_size 为 15。

15.3 架构扩展之多台 Puppet Master

现在已经将 WEBRick 替换成了 Web 服务器。然而, 有时可能需要提供比单台服务器

更多的处理能力，以解决单机单点故障。在这种情况下，除了纵向扩展外，还可以横向扩展 Puppet Master。横向扩展是使用多台服务器提供 Puppet Master 服务以组成一个集群来获得更多的处理能力。

要提供一个前端请求处理程序有多种方法和策略可供选择。我们选择使用 HTTP 负载均衡技术将客户端的请求直接导向后端服务器。每个 Puppet Master 都是单独配置。后端的服务器可以直接使用 Web 服务替换 WEBRick。

集群的架构扩展有着多种变通与组合方式。比较典型的组合方式有 5 种。下面我们就各种组合的架构流程图进行介绍。

1) 组合方式一：CA 与 Master 在同台服务器，利用 HTTP 负载均衡代理 Catalog。这种组合的架构流程如图 15-2 所示。

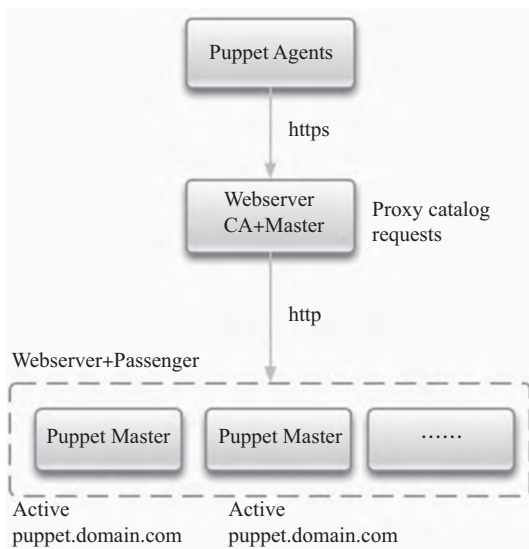


图 15-2 Master 集群扩展组合方式一（Master 与 CA 混和）

组合方式一的工作原理：

- ❑ 客户端证书由负载均衡器上的 CA 颁发，即通过代理转发 CA 请求至本地。
- ❑ 负载均衡通过代理 Catalog 请求至多台机器，以达到负载均衡的目的。
- ❑ 后端 Puppet Master 采用 Webserver 替代 WEBRick 以提升性能。
- ❑ manifests 代码通过 rsync 同步、GIT 或共享存储来实现。
- ❑ 此架构适用于单机房扩容，并且要求所有服务器主机名相同。

组合方式一的缺点：

- ❑ 负载均衡器存在单点故障，且扩展不方便。
- ❑ 不适合多机房跨运营商部署。

2) 组合方式二：Master 与 CA 位于独立的服务器，利用 HTTP 负载均衡器代理转发 CA 和 Catalog 请求。这种组合的架构流程如图 15-3 所示。

组合方式二的工作原理：

- ❑ 此组合的工作原理和组合方式一类似，代理请求将 CA 和 Catalog 分别转发到不同后端。
- ❑ 用户与负载均衡器之间及负载均衡器与后端之间均为 HTTPS 请求。
- ❑ 解决 CA 与负载均衡器单点故障。

组合方式二的缺点为不适合多机房跨运营商部署。

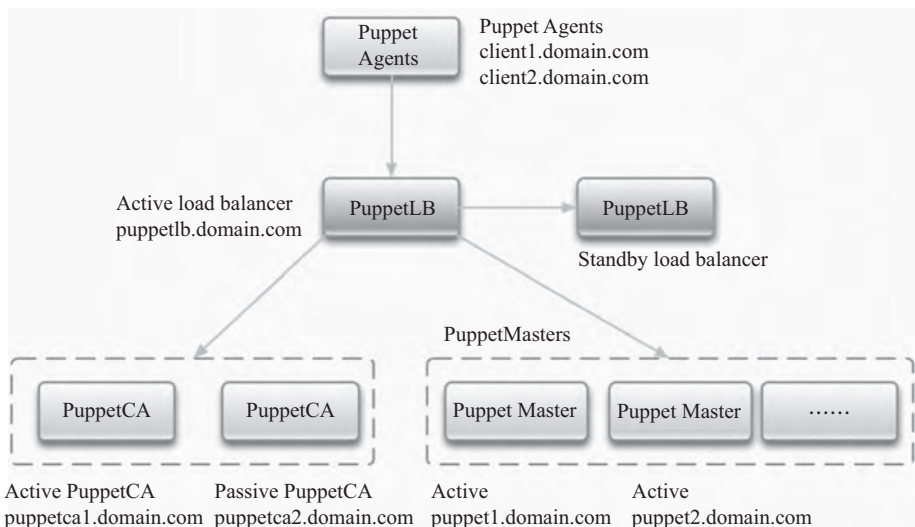


图 15-3 Master 集群扩展组合方式二 (Master 与 CA 独立)

3) 组合方式三: Master 与 CA 位于独立服务器, 利用 DNS 轮询或分区域解析至 Master, HTTP 负载均衡代理 CAfunctions。这种组合的架构流程如图 15-4 所示。

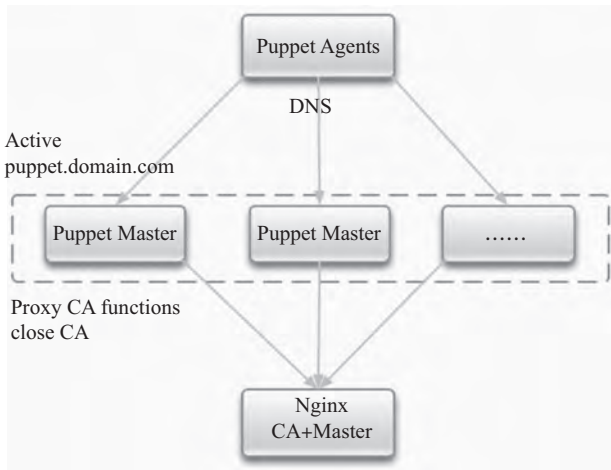


图 15-4 Master 集群扩展组合方式三 (Master 与 CA 独立, DNS 轮询)

组合方式三的工作原理:

- ❑ 通过 DNS 扩展达到多机房部署。
- ❑ Master 通过代理转发 CA 请求至 CA 服务器。

组合方式三的缺点为配置较复杂。

4) 组合方式四: CA 和 Master 位于独立服务器, 并指定 CA 服务器和 Master 服务器。这种组合的架构流程如图 15-5 所示。

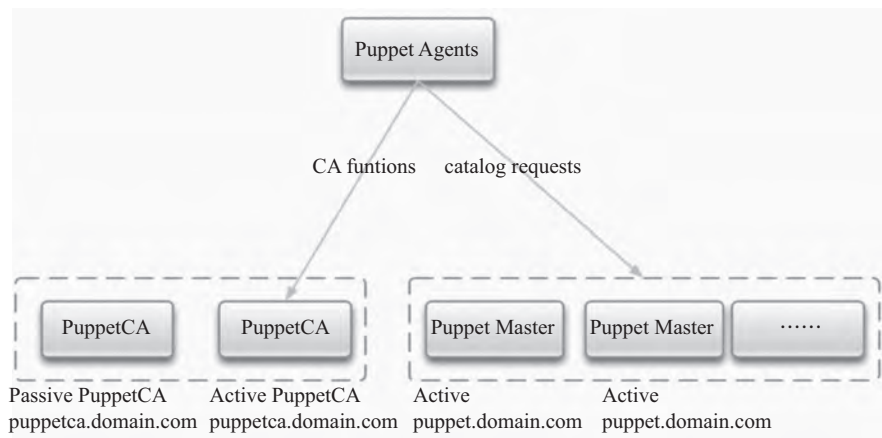


图 15-5 Master 集群扩展组合方式四（Master 与 CA 独立且要指定）

组合方式四的工作原理：

- ❑ 客户端通过配置 `ca_server` 指定 CA 服务器，以达到独立 CA 服务器的目的。
- ❑ CA 服务器可以部署在多个机房。
- ❑ Master 集群可以在同一机房配置负载均衡器，也可以使用 DNS 解析 Puppet Master 域名到不同机房的多台服务器，通过 DNS 实现负载均衡。

组合方式四的缺点：暂时没有发现。

5) 组合方式五：共享一个证书。共享一个证书文件需要将证书文件分发至客户端，因此笔者就不过多说明这种组合方式了。

特别说明：

- ❑ Master 是 CA 的域授权，在配置集群架构时，需要特别注意这一点，所有的 Master 都需要通过 CA 获得授权。
- ❑ 集群内 Master 主机名可以相同也可以不同，建议一个集群内的 Master 主机名相同；在增加 Master 或故障时需要安装好软件，并配置相同的证书文件。不需要在 CA 服务器上重新授权。
- ❑ 当集群内 Master 主机名不同时，增加一台 Master 服务器需要在 CA 服务器上重新授权。

下面将采取组合方式四并利用 Nginx 进行负载均衡，采用 Nginx+Passenger 模式替换 WEBRick 以详解 Puppet 集群架构配置过程。

主机的 IP 地址信息及用途如表 15-1 所示。

表 15-1 主机的 IP 地址信息及用途

主 机 名	IP 地址	用 途
puppetlb.domain.com	10.210.215.251	负载均衡服务器，转发请求
puppet.domain.com	10.210.213.217	Puppet 服务端，处理 catalog requests
puppetca.domain.com	10.210.213.219	Puppet 认证服务器，处理 CA funtions
agent.domain.com	10.210.215.254	Puppet 客户端

详细架构如图 15-6 所示。

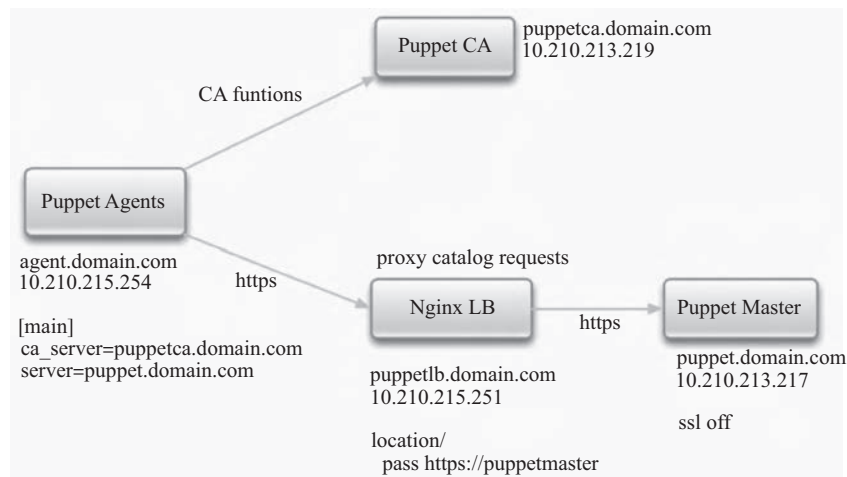


图 15-6 Master 集群扩展组合方式四的详细架构图

上述架构工作原理：

- ❑ 客户端配置 `ca_server` 和 `server` 指定认证和服务端服务器。
- ❑ CA 负责颁发所有客户端证书。
- ❑ 负载均衡 LB 与 Master 使用相同主机名的证书，证书由 CA 颁发。
- ❑ 如有多个 Master 时需要保持 manifests 代码同步，可采用 `rsync` 进行同步。

架构优点：

- ❑ CA 服务器可以任意扩展，只需要将证书复制至另一台服务器进行冷备。
- ❑ CA 服务器的死机对架构无大影响，只是新客户端将无法获取证书。
- ❑ 负载均衡 LB 可以纵向扩容至多组。
- ❑ Puppet Master 可以横向扩容至多台或多个集群。

15.3.1 配置前的准备

接下来进入配置阶段，整个配置过程不是很复杂。读者可以先通读一遍再进行操作。

1. 配置软件源

配置软件源的方法如下：

```
http://mirrors.ustc.edu.cn/fedora/epel/5/i386/epel-release-5-4.noarch.rpm
http://yum.puppetlabs.com/el/5Server/products/x86_64/puppetlabs-release-5-5.
noarch.rpm
```

2. 安装软件包

在部署应用时需要注意，在完成 Puppet 软件安装后，不要启用 Puppet 服务。

(1) Puppet Agent

在 Agent.domain.com 客户端配置主机名并安装 puppet 软件包，在命令行下运行如下代码：

```
$ sudo hostname agent.domain.com
$ sudo yum install puppet puppet-server
```

如果需要安装 Puppet 2.7 的版本，指定版本即可，具体如下：

```
puppet-2.7.20-1.el5 puppet-server-2.7.20-1.el5
```

(2) PuppetLB

Puppetlb.domain.com 负载均衡器只需要安装 Nginx，具体如下：

```
$ sudo hostname puppetlb.domain.com
$ sudo yum install nginx
```

(3) PuppetCA

Puppetca.domain.com 认证服务器配置主机名并安装 puppet-server 软件包，具体如下：

```
$ sudo hostname puppetca.domain.com
$ sudo yum install puppet puppet-server
```

如果需要安装 Puppet2.7 的版本，指定版本即可。

(4) Puppet Master

在 Puppet.domain.com 服务端配置主机名并安装 Nginx 和 puppet-server 软件包，具体如下：

```
$ sudo hostname agent.domain.com
$ sudo yum install puppet puppet-server
```

如果需要安装 puppet2.7 的版本，指定版本即可。

通过 passenger 配置并安装 nginx 至 /opt/nginx 目录，具体如下：

```
$ sudo yum -y install ruby-devel rubygems puppet puppet-server
$ sudo gem install rake rack passenger
$ sudo passenger-install-nginx-module
$ sudo mkdir -p /etc/puppet/rack/public
$ cp /usr/share/puppet/ext/rack/files/config.ru /etc/puppet/rack/
$ sudo chown -R puppet:puppet /etc/puppet/rack/
```

3. 设置主机名并解析

所有主机的 hostname 要能够解析服务器对应的 IP，可以通过配置 DNS 服务器解析主机名，也可以配置本地 hosts 文件“/etc/hosts”。

在所有服务器上编辑 /etc/hosts 文件并添加如下内容：

```
10.210.215.251 puppetlb.domain.com
10.210.213.217 puppet.domain.com
10.210.215.254 puppetca.domain.com
10.210.213.219 agent.domain.com
```

当 Puppet Master 服务器比较多时，如果选择不同的主机名，需要在 CA 上获取授权，当然也可以采用相同主机名。笔者建议所有 Master 服务器采用相同主机名，使用不同主机名在配置时比较麻烦。

15.3.2 PuppetCA 认证服务器部署

PuppetCA 的唯一目的就是签署和撤销证书。当 PuppetCA 服务不可用时，新的客户端将不能获得证书，从而会影响使用，而已签发证书的客户端却不受影响。因此将 CA 进行独立对架构，这对容错性而言是非常有必要的。

在采用分布式架构时，所有的服务端即 Puppet Master 证书也由 CA 生成，CA 授权 Master 整个域权限，以便接收并处理请求。在生成证书之前所有服务器上的 Puppet 服务都没有启动过，因此 CA 服务器需要生成 CA 服务器和服务端的证书。

(1) 生成证书

使用 puppetca 命令生成 CA 服务器与服务端域名证书，生成 puppetca.domain.com 和 puppet.domain.com 两个域名的授权证书文件。

```
$ sudo puppetca --generate --dns_alt_names puppetca.domain.com:puppet puppetca.
domain.com
$ sudo puppetca --generate --dns_alt_names puppet.domain.com:puppet puppet.
domain.com
```

这里 CA 服务器在 /var/lib/puppet/ssl 目录中将会产生相应的私钥和公钥。

(2) 将公钥、私钥和根证书复制至 PuppetLB 和 Puppet Master

在 PuppetLB 和 Puppet Master 服务器由创建相应的目录，具体如下：

```
$ sudo mkdir -p /var/lib/puppet/ssl/{certs,ca, private_keys}
```

如果服务器之前启动过 puppet 服务，需要删除 ssl 目录，具体如下：

```
sudo rm -rf /var/lib/puppet/ssl
```

在复制证书时需要注意复制后的名称会有变化，具体如下：

- ❑ 根证书 ca.crt.pem 在 Puppet LB 和 Puppet Master 服务器上为 ca.pem；
- ❑ signed 证书同步到 certs；
- ❑ 私钥 puppet.domain.com.pem 维持不变。

Master 在使用 Nginx+Passenger 模式时还需要复制废除证书文件 ca_crl.pem，具体如下：

```
$ cd /var/lib/puppet/ssl

$ sudo scp ca/signed/puppet.domain.com.pem root@10.210.213.217:/var/lib/puppet/
ssl/certs/puppet.domain.com.pem
$ sudo scp ca/ca.crt.pem root@10.210.213.217:/var/lib/puppet/ssl/certs/ca.pem
$ sudo scp private_keys/puppet.domain.com.pem root@10.210.213.217:/var/lib/
puppet/ssl/private_keys/puppet.domain.com.pem
```

PuppetCA 主配置文件如下：

```
$ sudo vim /etc/puppet/puppet.conf
[master]
  confdir = /etc/puppet
  certname = puppetca.domain.com
  ca = true
```

通过 `/etc/init.d/puppetmaster start` 启动 Puppet 服务即完成了 Puppet CA 服务器部署。由于 CA 服务器只是处理证书请求，因此不存在性能瓶颈，不需要使用 HTTP 负载均衡扩展。

15.3.3 Puppet LB 负载均衡器部署

负载均衡器 `puppetlb.domain.com` 负责转发请求，为此只需要配置 Nginx，通过 location 处理 Catalog 相关的请求。配置 Nginx 的虚拟主机内容如下：

```
upstream puppet-production {
    server 10.210.213.217:8140;
}

server {
    listen                8140 ssl;
    server_name           puppet.domain.com;

    access_log            /var/log/nginx/puppet_access.log;
    error_log             /var/log/nginx/puppet_error.log;

    ssl_protocols SSLv3 TLSv1;
    ssl_ciphers ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:-LOW:-SSLv2:-EXP;
    proxy_set_header      X-SSL-Subject      $ssl_client_s_dn;
    proxy_set_header      X-Client-DN       $ssl_client_s_dn;
    proxy_set_header      X-Client-Verify   $ssl_client_verify;
    client_max_body_size 100m;
    client_body_buffer_size 1024k;
    proxy_buffer_size 100m;
    proxy_buffers 8 100m;
    proxy_busy_buffers_size 100m;
    proxy_temp_file_write_size 100m;
    proxy_read_timeout 500;

    ssl on;
    ssl_session_timeout 5m;
    ssl_certificate /var/lib/puppet/ssl/certs/puppet.domain.com.pem;
    ssl_certificate_key /var/lib/puppet/ssl/private_keys/puppet.domain.com.pem;
    ssl_client_certificate /var/lib/puppet/ssl/certs/ca.pem;
    ssl_crl /var/lib/puppet/ssl/ca/ca_crl.pem;
    ssl_verify_client optional;

    ssl_prefer_server_ciphers on;
    ssl_verify_depth 1;
    ssl_session_cache shared:SSL:128m;

    location / {
```

```

        proxy_redirect                off;
        proxy_pass https://puppet-production;
    }
}

```

以上代码的详解如下：

- upstream，定义负载均衡设备 IP，如果后端有多台 Puppet Master 服务器时还可以通过（down、weight、max_fails、fail_timeout、backup）设置不同后端设备的状态；
- proxy，将服务器上接收到的用户信息传到后端服务器；
- ssl，配置证书相关文件，这里的证书由 CA 服务器生成；
- location，代理转发所有请求，需要注意 proxy_pass 是转发 HTTPS 请求。

如果负载均衡服务器与 CA 服务器部署在同一台服务器上，Nginx 的配置需要通过代理转发 CA 请求至本地 puppet master。配置文件参考如下：

```

location /production/certificate/{
    proxy_pass http://local'ip:8141;
    types{}
    default_typeapplication/x-raw;
}
location /production/certificate_request/{
    proxy_passhttp://local'ip:8141;
    types{}
    default_typeapplication/x-raw;
}
location /production/certificate_revocation_list/{
    proxy_passhttp://local'ip:8141;
    types{}
    default_typeapplication/x-raw;
}
location/{
    proxy_storeoff;
    proxy_passhttp://puppet-production;
}

```

注意，修改 local'ip 为生产环境中的本地服务器 IP。

这时 proxy_pass 只需要配置 HTTP 代理即可，后端所有 Puppet Master 服务器也不需要配置 SSL 等。配置参考如下：

```

server{
    listen local'ip:8140;
    root /etc/puppet/rack/public;
    passenger_enabled on;
    passenger_use_global_queue on;
}

```

15.3.4 Puppet Master 服务器部署

Puppet Master 服务器部署时需要在主配置文件 puppet.conf 中添加客户端 ssl header 配置

选项，以便能获取到客户端的请求信息。同时还需要配置关闭 ca 请求。

1) Puppet 主配置文件 puppet.conf，具体如下：

```
[master]
  certname = puppet.domain.com
  ca = false
  ssl_client_verify_header = HTTP_X_CLIENT_VERIFY
  ssl_client_header = HTTP_X_CLIENT_DN
```

2) 增加 Nginx 虚拟主机，具体如下：

```
server {
    listen                8140 ssl;
    server_name           puppet.domain.com;

    passenger_enabled     on;
    passenger_use_global_queue on;
    passenger_set_cgi_param HTTP_X_CLIENT_DN $ssl_client_s_dn;
    passenger_set_cgi_param HTTP_X_CLIENT_VERIFY $ssl_client_verify;
    proxy_buffer_size 4000k;
    proxy_buffering on;
    proxy_buffers 32 1280k;
    proxy_busy_buffers_size 17680k;
    client_max_body_size 10m;
    client_body_buffer_size 4096k;

    access_log            /var/log/nginx/puppet_access.log;
    error_log              /var/log/nginx/puppet_error.log;

    root                  /etc/puppet/rack/public;

    ssl                   off;
    ssl_session_timeout   5m;
    ssl_certificate        /var/lib/puppet/ssl/certs/puppet.domain.com.pem;
    ssl_certificate_key    /var/lib/puppet/ssl/private_keys/puppet.domain.com.pem;
    ssl_client_certificate /var/lib/puppet/ssl/certs/ca.pem;
    ssl_crl                /var/lib/puppet/ssl/ca/ca_crl.pem;
    ssl_verify_client     optional;

    ssl_ciphers            SSLv2:-LOW:-EXPORT:RC4+RSA;
    ssl_prefer_server_ciphers on;
    ssl_verify_depth       1;
    ssl_session_cache      shared:SSL:128m;

    # File sections
    location /production/file_content/files/ {
        types { }
        default_type application/x-raw;
        alias /etc/puppet/files/;
    }
}
```

配置完成后启动 Nginx，具体如下：

```
/etc/init.d/nginx start
```

15.3.5 Puppet 客户端配置

在 Puppet 客户端需要指定 CA 服务器与 Puppet Master 请求的域名。证书域名为 puppetca.domain.com，在获取证书与授权认证时需要通过该域名发起请求。Master 的域名是 puppet.domain.com，此域名通过 CA 服务器授权并将证书同步至负载均衡器与后端 Puppet Master 上，负载均衡器通过转发请求至不同的 Puppet Master 处理。

1) Puppet 主配置文件 puppet.conf 的内容如下：

```
[agent]
  masterport = 8140
  environment = production
  server = puppet.domain.com
  ca_server = puppetca.domain.com
```

2) 执行如下 puppet 命令：

```
$ puppet agent --test --server puppet.domain.com
```

15.3.6 验证架构

为验证请求可以分别在负载均衡器与后端 Puppet Master 上通过抓包查看请求过程，分别在 PuppetLB 和 Puppet Master 上运行命令如下：

```
tcpdump -s 1024 -l -A port 8140 -i eth0 -vvvv
```

如果要想验证 Puppet Agent 认证过程，可以先取消客户端的认证，重新发起认证请求即可。

15.4 架构扩展之利用 Git 构建分布式的 Puppet

使用 Puppet 最常见的做法就是运行一个 Puppet Master 服务器，当 Master 压力过大时，就需要采用 15.3 节介绍的方式进行扩展。Puppet 客户端每次都从 Puppet Master 获取配置清单（通常是编译好的 Catalog），然后在本地进行编译，通常这一过程是客户端通过 puppet agent 命令完成的。

Puppet 同时还提供 puppet apply 命令替代 puppet agent 的功能。可以利用 puppet apply 直接对配置清单进行操作。通常会使用 -v 参数开启冗余输出模式，这样就可以看到详细的执行过程。

定义一个 command.pp 文件内容为：

```
$ more command.pp
exec { 'test_logoutoutput':
  command => "/bin/ls command.pp",
}
```