Puppet cookbook 中文翻译

译者	版本	完成日期	版权
sky	1.0 版	2012-02-29	未获得

请勿用于商业目的

联系方式：
技术网站：http://www.mysqlops.com
新浪微博：http://weibo.com/mysqlops
腾讯微博：http://t.qq.com/mysqlops

译者微博：http://weibo.com/u/1938768691

邮件地址：mysqlops@sina.com 或 mysqlops@gmail.com
知乎账号：mysqlops
湖畔账号：mysqlops

puppet 基础架构(第一章)

"Computers in the future may have as few as 1,000 vacuum tubes and weigh only 1.5 tons."

— Popular Mechanics, 1949

在这一章中,我们将学习如下内容:

- 使用版本控制
- 使用 rake 来部署应用
- 配置 puppet 的文件服务器
- 使用计划任务来运行 puppet
- 从 puppet 的 filebucket 检索文件
- 使用 puppet 自动签名
- 预先签署证书

- **使用 Passenger 来扩展架构**
- **创建分布式 puppet 架构**

介绍

在这一章中,甚至在整本书中,所讲述的是代表着 puppet 的最佳实践,得到了 puppet 社区的同意.所有的这些技巧都是让你在使用 puppet 过程中更加轻松,或者向您介绍到你从未接触过的功能.我不建议你使用标准的操作过程,因为在使用 puppet 过程中有些技巧可以用.但是标准的操作过程在某些紧急情况下是有用的.最后,还有些你可能想尝试的 puppet 技巧,但是只有用于或者适用于非常庞大的基础设施或者一些不寻常的情况.

puppet 基础设施

我的期望是,通过对这里的 puppet 的使用方法的阅读和思考,你可以了解并加深对 puppet 是如何工作的理解,以及思考如何使用 puppet 来帮助自己建立更好的基础设施.只有你自己根据自己的组织以及个人实际情况来决定特定的或者适当的方法.但我希望我收集的这些技巧将鼓励你尝试,并总结方法,当然最重要的是--有乐趣.

使用版本控制

曾经错误的删除过东西的有没有?本书中最重要的技巧是将你的 puppet 代码纳入版本控制系统中,像 git 或者 Subversion(就是我们常用的 svn).直接在 puppetmaster 上编辑你的代码是个坏主意,那是因为你准备好了修改代码,却还没有得到即时应用.Puppet 会自动检查代码文件变化.所以你有可能会发现你的客户端只是使用了一个半成品.这可是一个令人讨厌的结果!替代的是,使用版本控制系统(我推荐使用 git)将 puppetmaster 上的/etc/puppet 目录从仓库导出.这样做会有以下几个优点:

1. 你不用担心 puppet 会运行不完整的代码.
2. 你可以撤消并回滚代码到以前的任意版本.
3. 你可以使用分支来尝试使用新功能,并不会影响到主要的线上使用的版本.
4. 如果多个人需要修改代码,他们可以相互独立,修改自己的工作副本,然后合并分支应用改动.
5. 你可以使用日志来查看何时,何人改动了什么.

准备

你需要一个 Puppetmaster 主机并且代码已经存放到/etc/puppet 目录下,如果这些你还没有准备好,你可以参考 Puppet 文档如何安装 Puppet 以及建立自己的第一个代码文件. 把你的代码纳入版本控制,你可以导入你 Pupppetmaster 上的/etc/puppet 目录到版本控制系统中,并使用它作为工作副本.在本例中,我们将使用 Git.

如何做到这一点

1.把 Puppetmaster 上的/etc/puppet 目录纳入 Git 仓库:

```
root@cookbook:/etc/puppet# git init
Initialized empty Git repository in /etc/puppet/.git/
```

```
root@cookbook:/etc/puppet# git add *
root@cookbook:/etc/puppet# git commit -m "initial commit"
[master (root-commit) 26d668c] initial commit
 2 files changed, 104 insertions(+), 0 deletions(-)
create mode 100644 auth.conf
create mode 100644 puppet.conf
```

2. 克隆/etc/puppet/目录到 Git 的仓库.

```
root@cookbook:/etc/puppet# git clone --bare /etc/puppet /var/git/puppet.git
Initialized empty Git repository in /var/git/puppet.git
```

3. 添加 bare 仓库作为源导入到/etc/puppet 目录:

```
root@cookbook:/etc/puppet# git remote add -t master origin /var/git/puppet.git
```

4. 在工作目录建立一个独立的导出目录,因此可以避免 Puppetmaster 正在使用.

```
root@cookbook:/etc/puppet# cd
root@cookbook:~# git clone /var/git/puppet.git puppet-work
Initialized empty Git repository in /root/puppet-work/.git/
```

是如何工作的……

你创建了一个主要仓库,你可以在提交改动之前在工作目录的任何地方进行导出.Puppetmaster 可以从主仓库更新到最新版本,当你认为它可以满足需求并可以应用到生成环境.

还有更多……

现在你已经建立了版本控制,你可以按照下面的工作流程来修改你的 Puppet 代码.

- 在工作副本里应用改动.
- 提交更改,并把改动推送到原仓库.
- 更新 Puppetmaster 的工作副本.

下面是一个例子,演示了我们添加了新文件,提交并更新到 Puppetmaster 的工作副本.

```
root@cookbook:~# cd puppet-work
root@cookbook:~/puppet-work# mkdir manifests
root@cookbook:~/puppet-work# touch manifests/nodes.pp
root@cookbook:~/puppet-work# git add manifests/nodes.pp
root@cookbook:~/puppet-work# git commit -m "adding nodes.pp"
[master 5c7b94c] adding nodes.pp
0 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 manifests/nodes.pp
root@cookbook:~/puppet-work# git push
Counting objects: 5, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 365 bytes, done.
Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
```

```
To /var/git/puppet.git
26d668c..5c7b94c master -> master
root@cookbook:~/puppet-work# cd /etc/puppet
root@cookbook:/etc/puppet# git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From /var/git/puppet
26d668c..5c7b94c master
-> origin/master
Updating 26d668c..5c7b94c
Fast-forward
0 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 manifests/nodes.pp
```

你已经了解整个过程,你可以使用例如 Rake 的工具来自动化这一过程.

注:git 一般工作流程是先在本地工作目录修改,完成后提交到 git 本地仓库,再更新到 git master.
git 相关操作可见 git 指南:
https://sites.google.com/a/kingofat.com/wiki/git-tutorial

参见

- 用 Rake 来部署应用.
- 创建可扩展的 Puppet 架构
- 使用 commit 钩子
-

Blog 参见:
http://www.mysqlops.com/2011/09/27/puppet-%E8%BF%90%E7%BB%B4%E8%87%AA%E5%8A%A8%E5%8C%96%E4%B9%8Bpuppet%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6.html

使用 Rake 来部署应用

大家的生活都离不开键盘,我讨厌不必要的敲打.如果你是按照"把你的代码纳入版本控制"中所描述的工作流程一样.你可以添加一些自动化使这个过程更加容易一些.有许多工具可以帮助我们在远程机器上执行命令,包括我们常见的 Capistrano 和 Fabric,但是这个例子,我们却是使用 Rake.

准备

如果你还没有安装 Rake 的 gem,运行:

gem install rake

如何做到...

1.在你的工作副本的顶级工作目录创建一个名为 Rakefile 的文件,内容看起来是这样的:

```
PUPPETMASTER = 'cookbook'
SSH = 'ssh -t -A'
task :deploy do
sh "git push"
sh "#{SSH} #{PUPPETMASTER} 'cd /etc/puppet && sudo git pull'"
end
```

2.当你要在工作目录应用改动时,你只要简单运行:

 # rake deploy

Rake 将会很仔细的从 Git 仓库更新并刷新 Puppetmaster 的工作目录:

```
#git push
Counting objects: 4, done.
Delta compression using 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 452 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh://git@cookbook.bitfieldconsulting.com/var/git/cookbook
561e5a6..a8b8c76 master -> master
ssh -A -l root cookbook 'cd /etc/puppet && git pull'
From ssh://cookbook.bitfieldconsulting.com/var/git/cookbook
561e5a6..a8b8c76 master
-> origin/master
Updating 561e5a6..a8b8c76
Fast-forward
Rakefile |
6 ++++++
1 files changed, 6 insertions(+), 0 deletions(-)
create mode 100644 Rakefile
```

3.你也可以创建一个 Rake 任务,任务是在客户端运行 Puppet.

```
task :apply => [:deploy] do
client = ENV['CLIENT']
sh "#{SSH} #{client} 'sudo puppet agent --test'" do |ok,
status|
puts case status.exitstatus
```

```
when 0 then "Client is up to date."
when 1 then "Puppet couldn't compile the manifest."
when 2 then "Puppet made changes."
when 4 then "Puppet found errors."
end
end
end
```

4.如果你想测试客户端改动,运行:

```
#rake CLIENT=cookbook apply
```

你可以替换 cookbook 为你的客户端的主机名,或者设置 CLIENT 环境变量,
只要 Rake 知道要在哪台客户端机器上运行 Puppet.

```
info: Caching catalog for cookbook
info: Applying configuration version '1292865016'
info: Creating state file /var/lib/puppet/state/state.yaml
notice: Finished catalog run in 0.03 seconds
```

5.如果你仅仅想查看 Puppet 将会做些什么,而不是实际应用操作,你可以使用 --noop 标志:

```
task :noop => [:deploy] do
client = ENV['CLIENT']
sh "#{SSH} #{client} 'sudo puppet agent --test --noop'"
end
```

现在你可以运行:

```
#rake noop
```

可以预览改变.

是如何工作的...

一个 Rakefile 是有一系列的确定的任务组成,以关键字 task 来区别.而任务定义是一组步骤,在本例中依赖一系列的 shell 命令推送你的代码改动到主要的仓库,然后更新 Puppetmaster 的工作副本.任务可以互相链接,因些一个任务依赖其它任务.例如,在我们的 Rakefile 中,apply 的任务是连接到 deploy,因些,每当你运行 rake apply.Rake 会确保先完成 deploy 任务,然后再做 apply 任务.

还有更多...

你可以延伸这个 Rakefile 去实现更多的任务自动化.包括在更新 Puppet 代码前运行语法检查,甚至你可以在引导启动过程中使用 puppet 来初始化你的新机器.

另请参阅

- 将你的 puppet 代码纳入版本控制
- 基于 Git 创建一个分布式 Puppet 架构
- 使用 commit 钩子来检查你的代码

参考 blog:http://www.mysqlops.com/2011/11/28/rake-code.html

配置 puppet 文件服务器

在每个核电厂控制系统的某个地方,有个名称为"reactor.conf".部署配置文件是 puppet 最常见的用途之一.大多数常见的服务需要一些配置文件,你可以像这样使用 puppet 文件资源,将配置推送到客户端.

```
file { "/opt/nginx/conf.d/app_production.conf":
source => "puppet:///modules/app/app_production.conf",
}
```

source 参数是这样工作的:第一部分是在 puppet:///假设是一个挂载点,其余部份被视为一个文件路径.

puppet:///<mount point>/<path>

<<mount point>>最常见的值是模块,如上例所示.在这种情况下,Puppet 将会寻找文件在:

manifests/modules/app/files/app_production.conf

pupet 对待模块比较特别,将其视为一个挂载点:它希望接下来和路径组径是一个模块名,那么然后就会在模块目录下的 files 目录来寻找文件.因为其余的都被视为路径.但是,Puppet 可以创建自定义挂载点,可以有个别的访问控制设置,可以映射到 Puppetmaster 的不同位置.
在这章我们将演示如何创建和配置这些自定义的挂载点.

如何做到.....

1.在 PuppetMaster 的 fileserver.conf 添加新的一节,模块名放在[](方括号)中,
path 的值就是 puppet 将会寻找数据的目录路径.像这样:

```
[san]
path /mnt/san/mydata/puppet
```

2.在你的代码里,你可以指定 source 使用你的挂载点名称,像这样:

source => "puppet:///san/admin/users.htpasswd",

那么 Puppet 就会转换路径为:

/mnt/san/mydata/puppet/admin/users.htpasswd

像这样创建一个自定义挂载点有个很好的原因那是更加安全一些.比方说:你有个绝密的密码文件只应该部署到 web 服务器,其它机器不需要它.如果有人能够运行 puppet 并且能合法的访问 Puppetmaster,没有人能阻止他像这样执行下面的代码:

```
file { "/home/cracker/goodstuff/passwords.txt":
source => "puppet:///web/passwords.txt",
}
```

他们可以获取秘密数据,事实上,任何人可以导出 Puppet 仓库或者谁有 Puppetmaster 上的账户都能访问此文件.为了避免这个问题的方法之一是把秘密数据放在自定义挂载点并启用访问控制.

3.像这样在 fileserver.conf 里增加 allow 和 deny 参数来定义你的挂载点:

```
[secret]
/data/secret
allow web.example.com
deny *
```

它是如何工作的...

在本例中,只允许 web.example.com 来访问此文件.默认是拒绝所有的访问,因此 deny * 这一行不是绝对必要的,可以省略.但它确实是个好的习惯,这样看上去更清晰.然后 web 服务器可以像这样使用文件资源:

```
file { "/etc/passwords.txt":
source => "puppet:///secret/passwords.txt",
}
```

如果此代码是在 web.example.com 上执行,那么将会正常工作,如果是其它任何客户端,那么会执行失败.

还有更多...

你也可以指定一个 IP 地址来代替主机名,可以选择使用网段.CIDR(斜线)符号或者通配符,像这样:

```
allow 10.0.55.0/24
allow 192.168.0.*
```

参见

- 使用模块
- 分发目录树
- 使用多个文件来源

Blog 参见:
http://www.mysqlops.com/2011/10/06/puppet-%E8%BF%90%E7%BB%B4%E8%87%AA%E5%8A%A8%E5%8C%96%E4%B9%8B%E6%B7%B1%E5%85%A5%E7%90%86%E8%A7%A3fileserver.html

从计划任务运行 puppet

你的 Puppet 进程是 sleeping 状态吗?默认情况下,当你在客户端上运行 Puppet agent,Puppet agent 会成为一个守护进程(后台进程),每隔 30 分钟唤醒并检查 puppet 代码是否有任何改动更新,以及应用他们.如果你想要更多的控制 Puppet 运行,你可以使用 cron 代替人为的触发.

例如,如果你有很多的 Puppet 客户端,你可能会有意识的错开 Puppet 的运行时间,以减少 Puppetmaster 的压力.一个简单的方法是根据客户端的主机名做哈希表,并作为计划任务作业的分钟或者小时.

如何做...

1. 使用 Puppet 的 inline_template 功能, inline_template 允许你执行 Ruby 代码:

```
cron { "run-puppet":
command => "/usr/sbin/puppet agent --test >/dev/null 2>&1",
minute => inline_template("<%= hostname.hash % 60 %>"),
}
```

说明:hash 生成的数值可以是无限大,上例中只是生成 0-60,也就是限制了最大值为 60.

它是如何工作的...

因为每个主机名产生一个唯一的哈希值,每个客户端将会在过去的每小时中的不同分钟数运行 Puppet.这个散列技术是有用的随机任何的 cron 作业,提高了可性能,因为他们不会互相干扰。

还有更多...

本节没有其它的.

参见

- 随机分发 cron 作业以减少 load(压力)
- 使用嵌入式 Ruby 的 inline_template 来补充 Puppet 语言

从 puppet filebucket 检索与恢复文件

有铅笔就有橡皮擦，因为我们都会犯错，每当 puppet 检测到客户端文件变化时，它会把当前版本做个备份，我们可以看到这一过程，在 puppet 运行实例中，我们替换一个现有存在的文件，并且文件内容有变化，我们看示例：

```
root@cookbook:/etc/puppet# puppet agent --test
info: Caching catalog for cookbook
info: Applying configuration version '1293459139'
--- /etc/sudoers        2010-12-27 07:12:20.421896753 -0700
+++ /tmp/puppet-file20101227-1927-13hjvy6-0 2010-12-27
07:13:21.645702932 -0700

@@ -12,7 +12,7 @@
  # User alias specification
-User_Alias SYSOPS = john
+User_Alias SYSOPS = john,bob
info: FileBucket adding /etc/sudoers as {md5}c07d0aa2d43d58ea7b5c5307
f532a0b1
info: /Stage[main]/Admin::Sudoers/File[/etc/sudoers]: Filebucketed /
etc/sudoers to puppet with sum c07d0aa2d43d58ea7b5c5307f532a0b1
notice: /Stage[main]/Admin::Sudoers/File[/etc/sudoers]/content:
content changed '{md5}c07d0aa2d43d58ea7b5c5307f532a0b1' to '{md5}0d218
c16bd31206e312c885884fa947d'
notice: Finished catalog run in 0.45 seconds
```

我们感兴趣的是下面这行：

```
info: /Stage[main]/Admin::Sudoers/File[/etc/sudoers]: Filebucketed /
etc/sudoers to puppet with sum c07d0aa2d43d58ea7b5c5307f532a0b1
```

puppet 会根据文件内容创建一个 MD5 哈希,并使用它来创建一个 filebucket.filebucket 是基于哈希的前几个字符，filebucket 的作用是来保存 puppet 替换下来的任何文件的副本，它存放的默认位置是 **/var/lib/puppet/clientbucket:**

```
 root@cookbook:/etc/puppet# ls /var/lib/puppet/clientbucket/c/0/7/
d/0/a/a/2/c07d0aa2d43d58ea7b5c5307f532a0b1
contents    paths
```

你在 bucket 存放位置会看到两个文件：**content**s 和 **paths.contens** 即为原始文件内容，paths 即为原始文件的路径。

如果你知道文件内容的哈希值（像你看到的上面的例子）,可以很容易找到该文件，如果你不知道，在整个 filebucket 中创建以文件内容为表，并建立索引文件将非常有用。

如何去做...

1. 使用如下命令创建索引文件：

```
find /var/lib/puppet/clientbucket -name paths -execdir cat {} \;
-execdir pwd \; -execdir date -r {} +"%F %T" \; -exec echo \; > bucket.txt
```

2. 在索引文件中查找文件：

```
cat bucket.txt
/etc/sudoers
/var/lib/puppet/clientbucket/c/0/7/d/0/a/a/2/c07d0aa2d43d58ea7b5c5
307f532a0b1
2010-12-27 07:13:21
/etc/sudoers
/var/lib/puppet/clientbucket/1/0/9/0/e/2/8/a/1090e28a70ebaae872c2e
c78894f49eb
2010-12-27 07:12:20
```

如果你想恢复文件，并且知道他的 bucket path,那么只需要复制文件内容到原始文件，即可。

```
cp /var/lib/puppet/clientbucket/1/0/9/0/e/2/8/a/1090e28a70ebaae872
c2ec78894f49eb/contents      /etc/sudoers
```

是如何工作的...

刚那个脚本会创建一份完整的 filebucket 文件列表清单，显示原始文件的名称，以及 bucket 的路径，以及修改日期，(在上例中你学习到了如何恢复文件到以前版本)，一旦你知道 bucket 的路径，那么你就可以复制文件到正确的位置。

还有更多

你可以指定 puppet 在原始目录下创建备份文件，而不是在 filebucket.要想做到这一台，你只需要在代码中添加 backup 参数：

```
file { "/etc/sudoers":
      mode      => "440",
```

```
    source => "puppet:///modules/admin/sudoers",
    backup => ".bak",
}
```

现在如果 puppet 要替换上面的文件，会在原始路径(/etc)下面创建一个扩展名为.bak 的备份文件，如果你想 puppet 默认在替换文件前先备份这一策略，可以这样做：

```
 File {
    backup => ".bak",
}
```

要完全禁用备份，可以这样做：

```
 backup => false,
```

使用自动签名

在密码学中，跟生活一样，你必须小心你的签名，一般来说，每当 puppetmaster 新增一个客户端，你需要在客户端上生成一个证书请求，然后你要到 puppetmaster 给它 ssl 签名(puppetca -s hostname)。然而，你可以使用自动签名（autosign）跳过这一步骤。

如何做...

1.在 puppetmaster 上创建/etc/puppet/autosign.conf 文件，添加如下内容：

***.example.com　##请根据实际情况，修改为你客户机所对应的域名**

是如何工作的...

puppet 会检查所有的证书请求是否匹配 autosign.conf 中任何一行，客户端主机名匹配 *.example.com 的任何证书请求,puppetmaster 都会自动签名。

重要:这是个潜在的安全问题，因为 puppetmaster 信任任何客户端连接，只要主机名匹配，出于这个原因，不建议使用 autosigning,如果你真的要这样做用，确保 puppetmaster 受防火墙保护，只允许信任的客户端或者 ip 段连接，一个更安全的方法是使用 pre-signing.

Blog 见：http://www.mysqlops.com/2011/11/12/puppet-ssl-sign.html

Pre-signing certificates (预签名)

这里有一个刚才签名过的例子，一般来说，如果你要自动的加入大量客户端，最好在 puppetmaster 上预先给客户端签名，然后推送相应 ssl 证书到客户端，这是整个过程，你可以使用 puppetca --generate <hostname> 生成证书。

如何做...

1.给 clietn1.example.com 预先生成 ssl 证书。

puppetca --generate client1.example.com

puppet 会生成并签名一个证书名为 client1.example.com.pem

3. 复制三个文件到新的客户端：客户端证书的私钥，客户端证书和 CA 证书。这三个文件所在位置为：

/etc/puppet/ssl/private_keys/client1.example.com.pem
/etc/puppet/ssl/certs/client1.example.com.pem
/etc/puppet/ssl/certs/ca.pem

复制上述三个文件到客户端相应的目录下，puppet 会自行进行身份验证可以省略 ssl 证书请求这一步骤。

使用 passenger 来扩展 puppet

很难解雇一个员工因为他们工作得太久，从一开始到慢慢变老，puppetmaster 使用一个简单的 web 服务器名为 Webrick.来处理客户端连接，在少量服务器的时候是没有问题，但是你可能会发现客户端数量的增加(比如 50-100+),对于 puppetmaster 来说可能是性能瓶颈。

为了扩展 puppet 以应对数百台服务器的请示，一种方法是切换到更高性能的 web 服务器像 apache ,使用 Passenger(mod_rails)模块，puppet 在 Passenger 运行下，需要些必需的配置，因此，你需要安装 Apache 和 Passenger,并添加一个合适的虚拟主机。

下面的示例是在 Ubuntu 10.4 下。你可以在 puppetlabs 上按照说明找到 RedHat linux,CentOs 以及其它发行版本的上对应的做法。

参考文档地址： http://projects.puppetlabs.com/projects/1/wiki/Using_Passenger

准备工作：

如果你有 puppet 的源码包的话，那就是非常省事。

1. 举例来说：如果你想安装 puppet 2.6.4，下载 puppet 2.6.4 的源码包。

wget –c http://puppetlabs.com/downloads/puppet/puppet-2.6.4.tar.gz

如果你想用其它版本，你可以到 http://puppetlabs.com 下载适合你的版本。

2. 解压缩源码包：

tar xzf puppet-2.6.4.tar.gz

如何做...

1. 安装 apache 和 Passenger,以及相关依赖包。

apt-get install apache2 libapache2-mod-passenger rails librack-ruby libmysql-ruby
gem install rack

2. 创建 Passenger 必须需要的目录，用来查找 puppet 的配置。

mkdir -p /etc/puppet/rack
mkdir -p /etc/puppet/rack/public

这些目录权限要设置 755,必且属主为 root

chown root:root /etc/puppet/rack && chmod 755 /etc/puppet/rack
chown root:root /etc/puppet/rack/public && chmod /etc/puppet/rack/public

2. 创建 Passenger 启动 puppet 程序所需要的 config.ru 文件。你可以使用 puppet 自带的示例文件：

cp /tmp/puppet-2.6.4/ext/rack/files/config.ru /etc/puppet/rack/
chown puppet /etc/puppet/rack/config.ru

在 puppet 2.6.4 版本：会有下面的内容：

 # a config.ru, for use with every rack-compatible webserver.
 # SSL needs to be handled outside this, though.
 # if puppet is not in your RUBYLIB:
 # $:.unshift('/opt/puppet/lib')

```
$0 = "master"
# if you want debugging:
# ARGV << "--debug"
ARGV << "--rack"
require 'puppet/application/master'
# we're usually running inside a Rack::Builder.new {} block,
# therefore we need to call run *here*.
run Puppet::Application[:master].run
```

4.你现在需要在 apache 创建一个虚拟主机，修改相应的端口，使其监听 puppet 监听端口并能响应 puppet 请求,同样，你也可以使用 puppet 自带的示例文件：

```
cp /tmp/puppet-2.6.4/ext/rack/files/apache2.conf
   /etc/apache2/sites-available/puppetmasterd
#a2ensite puppetmasterd
```

文件部分内容看起来应该是这样的：

```
# you probably want to tune these settings
PassengerHighPerformance on
PassengerMaxPoolSize 12
PassengerPoolIdleTime 1500
# PassengerMaxRequests 1000
PassengerStatThrottleRate 120
RackAutoDetect Off
RailsAutoDetect Off
Listen 8140
<VirtualHost *:8140>
   SSLEngine on
   SSLProtocol -ALL +SSLv3 +TLSv1
   SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:-LOW:-SSLv2:-EXP
   SSLCertificateFile          /etc/puppet/ssl/certs/cookbook.
bitfieldconsulting.com.pem
   SSLCertificateKeyFile      /etc/puppet/ssl/private_keys/cookbook.
bitfieldconsulting.com.pem
   SSLCertificateChainFile /etc/puppet/ssl/ca/ca_crt.pem
   SSLCACertificateFile        /etc/puppet/ssl/ca/ca_crt.pem
   # If Apache complains about invalid signatures on the CRL, you
can try disabling
   # CRL checking by commenting the next line, but this is notrecommended.
   SSLCARevocationFile         /e
   SSLVerifyClient optional
   SSLVerifyDepth    1
   SSLOptions +StdEnvVars
```

```
    DocumentRoot /etc/puppet/r
    RackBaseURI /
    <Directory /etc/puppet/rac
      Options None
      AllowOverride None
      Order allow,deny
      allow from all
    </Directory>
</VirtualHost>
```

5. 修改上面的文件，请注意把 **SSLCertificateFile** 和 **SSLCertificateKeyFile** 修改为自己的证书（最简单的方法是运行 puppet 生成证书）。

6. 你还要在 apache 启用 Passenger 和 mod_ssl 模块

a2enmod passenger ssl

7. 在/etc/puppet/puppet.conf 文件里添加下面的内容：

ssl_client_header = SSL_CLIENT_S_DN
ssl_client_verify_header = SSL_CLIENT_VERIFY

8.停止正在运行的 puppetmaster。

8. 启动 apache 服务

/etc/init.d/apache2 restart

10.如果一切工作正常，你可以像平时一样运行 puppet:

```
# puppet agent --test
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1294145142'
notice: Finished catalog run in 0.25 seconds
```

是如何工作的...

使用 apache 代替 puppet 内置的 web 服务器，内置的 web 服务器处理速度相当慢，-次只能处理一个请求，你使用高性能多线程的 apache 服务器，puppet 使用 Rack framework 作为嵌入式应用程序，效率更高，你应该会发现，你现在使用 apache 和 Passenger 配置能处理更多的客户端和更频繁的 puppet 请求，并且 puppetmaster 使用的内存比标准的 puppetmaster 守护进程占用的内存更少。

还有更多...

下面是一个 puppet 代码示例 ：可以实现上述功能（Ubuntu 系统）：

```
class puppet::passenger {
      package { [ "apache2-mpm-worker",
                  "libapache2-mod-passenger",
                  "librack-ruby",
                  "libmysql-ruby" ]:
            ensure => installed,
      }
      service { "apache2":
            enable    => true,
            ensure    => running,
            require => Package["apache2-mpm-worker"],
      }

     package { "rack":
            provider => gem,
            ensure      => installed,
      }
file { [ "/etc/puppet/rack",
                  "/etc/puppet/rack/public" ]:
            ensure => directory,
            mode      => "755",
      }
      file { "/etc/puppet/rack/config.ru":
            source => "puppet:///modules/puppet/config.ru",
            owner    => "puppet",
      }
      file { "/etc/apache2/sites-available/puppetmasterd":
            source => "puppet:///modules/puppet/puppetmasterd.conf",
      }
      file { "/etc/apache2/sites-enabled/puppetmasterd":
            ensure => symlink,
            target => "/etc/apache2/sites-available/puppetmasterd",
      }
      exec { "/usr/sbin/a2enmod ssl":
            creates => "/etc/apache2/mods-enabled/ssl.load",
      }
}
```

更多详细信息，或者如果你遇到问题，可以参考 Puppet-on-Passenger 文档：
http://projects.puppetlabs.com/projects/1/wiki/Using_Passenger

blog 参见：

http://www.mysqlops.com/2011/09/20/puppet-%E8%BF%90%E7%BB%B4%E8%87%AA%E5%8A%A8%E5%8C%96%E4%B9%8B%E4%BD%BF%E7%94%A8nginx%E8%B4%9F%E8%BD%BD%E5%9D%87%E8%A1%A1.html

创建一个分布式 puppet 架构

像 Mafia 系统一样，他们使用分布式架构并且运行的很好，最常见的使用方法是运行一个 puppetmaster 服务，这样 puppet 客户端就可以连接到 puppetmaster 并且能从 puppetmaster 上下载伪代码，虽然你也可以直接使用 puppet 客户端运行 puppet 代码，（你通常要使用 -v 开关开启详细输出模式，这样你就可以看到执行结果）

 # puppet -v manifest.pp
info: Applying configuration version '1294313350'

甚至你也可以直接在命令行运行代码片断：

 # puppet -e "file { '/tmp/test': ensure => present }"
notice: /Stage[main]//File[/tmp/test]/ensure: created

换句话说，如果你能整理合适的代码片断文件并能分发到客户端，你可以在 puppet 客户端直接运行它，而不需要一台中心的 puppetmaster 服务器，这将消除单台 puppetmaster 主服务器的性能瓶颈，也消除了单点故障，也避免了新增客户端的时候申请 ssl 证书以及证书的签署这些步骤。

有许多方法能实现将代码文件推送到客户端，但是 Git（或者其它版本控制系统如 Mercurial 或者 Subversion)能为你做大部分工作，你可以在本地编辑 puppet 代码副本，提交到 git 服务器，并将更新推送到中心仓库，并从 git 中心仓库自动分发代码到客户机。

准备工作

如果你的 puppet 代码还不在 git 版本控制下，请按照下列步骤：

将你的 puppet 的代码纳入版本控制.

如何做...

1. 请在客户端对 puppet 仓库做个全新的克隆。

 # git clone --bare ssh://git@repo.example.com/var/git/puppet

2. 使用下面命令复制仓库内容到/etc/puppet/目录

 # git archive --format=tar HEAD | (cd /etc/puppet && tar xf -)

3. 使用 puppet 命令执行 site.pp 文件：

 # puppet -v /etc/puppet/manifests/site.pp
info: Applying configuration version '1294313353'

4. 一旦完成上面的工作，下一步是配置自动推送到客户端，使用 git,你可以这样添加远程 web 服务器。
 # git remote add web ssh://git@web1.example.com/etc/puppet

4. 如果你有多个客户端，你可以添加多个远程 url:

 # git remote set-url --add webs ssh://git@web2.example.com/etc/puppet
 # git remote set-url --add webs ssh://git@web3.example.com/etc/puppet

或者，像这样编辑 git 的配置文件(.git/config):

[remote "web"]
url = ssh://git@web1.example.com/etc/puppet
url = ssh://git@web2.example.com/etc/puppet
url = ssh://git@web3.example.com/etc/puppet

5.现在你可以从 git 中心仓库推送更新到任意一台客户端，或者一组客户端
 # git push web

6.最后一步是客户端更新/etc/puppet 目录，一旦它接收到从 git 仓库的推送。
因此你可以使用 git 的 post-receive 钩子程序，在你全新的仓库目录下，创建 hooks/post-receive 文件，并赋于 0755 权限。文件内容如下：

#!/bin/sh
git archive --format=tar HEAD | (cd /etc/puppet && tar xf -)

是如何工作的：

替代连接 puppetmaster，下载编译好的伪代码，每个客户端都编译自己本地副本代码，每次你从 git 服务器获得更新（或者你从 git 仓库 checkout 下来），这对网络带宽有影响，如果客户端不需要联系上 puppetmaster 进行运行的话，那么，它也就消除了单点故障，

客户端可以从任何地方获得更新。

使用基于 git 的分布式的 puppet 架构为你提供了一个非常灵活的处理方式，你可以配置使用 ssh key 来启用访问控制和权限设定，以承载单台客户端或者一组客户端更多访问，数据库服务组，例如：可以只提供给需要访问的机器，但然也需要一定的额外的工作，对于大多数小规模部署这种方式是没有必要的，puppet 分布式架构提供了额外的灵活性，也提供了最苛刻权限控制的环境，

额外更多

如果你想每次只要中心仓库推送更新，puppet 立即更新并执行代码，你可以编辑 post-receive 脚本来完成，或者你采取其它的方法，或者，你也可以手动运行 puppet,或者是像上一章节中使用 cron 调度 puppet 运行，只要记得是运行 puppet,而不是 puppet 代理。

你可以在 Stephen Nelson-Smith 这篇文章里找到更多更详细的关于 puppet 架构的讨论，文章地址是：
http://bitfieldconsulting.com/scaling-puppet-with-distributed-version-control

监控报告以及调试

“发现问题的方法不止一个，然而，这并不意味着相关部分事情就是错误的，也可能在其它应用程序里找到很多。”
-错误信息

在本章，你可以学习以下内容：

- 生成报告
- 通过电邮发送包含特定标签的报告
- 创建图形报告
- 生成生成 html 文档
- 绘制依赖关系图
- 测试 puppet 代码
- 无损运行 puppet
- 编译错误检查
- 使用 commit 钩子
- 了解 puppet 报错信息
- 记录命令输出
- 记录调试信息
- 检查配置设置
- 使用标签

- 使用运行阶段
- 使用多环境

简介：
我们都曾经经历过坐在椅子上看着令人兴奋的新技术，赶紧回家尝试使用它，当然，一旦你开始试验它，你马上就会遇到问题，什么地方弄错了，它为什么不能正常工作，我怎么样才能看到究竟发生了什么，本章将帮助你回答上述问题，给你工具去解决常见的 puppet 的问题，我们还能看到生成有用的 puppet 报告,并如何使用 puppet 来监控和调试你的整个网络架构，

生成报告

世界上真正需要的是更多的爱和更少的文案工作

-Perarl Bailey

真正的第一个受害者是大型的架构，如果你管理许多机器，并使用 puppet 报告设备信息，从中可以看出一些很有价值的信息，以及设备实际上正在发生着什么.

如何做呢

1. 为了使用 puppet 报告，需要在客户端配置文件添加一行(/etc/puppet/puppet.conf):

enable = true

它是如何工作的

启用报告后，puppet 会生成一个报告并发送到 puppetmaster 上，其中包含像这样的数据如：
从 puppetmaster 获取配置的时间

puppet 的总运行的时间
在运行过程中日志消息输出
客户端所有的可用资源列表
puppet 是否应用每个资源
一个资源是否与代码同步

默认情况下，这些报告都保存在/var/lib/puppet/reports 目录下，当然你也可以使用 reportdir 参数来指定特定的目录来保存这些报告，你也可以创建自己的脚本去处理这些报告（报告都是标准的 YAML 格式的），或者使用一个工具，例如 puppet dashboard 来获得一个网络图形概览。

还有更多..

这两有两个或者以上的实用技巧来收集 puppet 报告

在命令行下启用报告：

如果你只是想要一个报告，或者你不想让所有客户端都发送报告，你可以手动切换到命令行，并添加--report 参数。

#puppet agent --test –report

你还可以看到 puppet 的运行统计概要信息，可以使用--summarize 参数。

```
# puppet agent --test --summarize
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1306169315'
notice: Finished catalog run in 0.58 seconds
Changes:
Events:
Resources:
                    Total: 7
Time:
     Config retrieval: 3.65
           Filebucket: 0.00
              Schedule: 0.00
```

记录 puppet 输出到系统日志记录

puppet 也可以发送日志信息到 puppetmaster 的系统日志记录，因此你也可以使用标准的 syslog 工具，为了实现这一点，你可以在 puppetmaster(puppet.conf)上添加下面选项:

```
[master]
reports=store,log
```

store 是默认的报告选项（输出报告到**/var/lib/puppet/reports**），log 选项是告诉 puppet 也将消息发送到 syslog 日志中去。

根据特定的标签发送日志信息到电子邮件

像大多数系统管理员一样，如果没有收到足够到的邮件，你会寻找一种方法生成邮件，另外一种 puppet 报告被称为 tagmail.它会根据你设定的 email 地址发送相应的日志信息到邮箱。

如何做

1.修改/etc/puppet/puppet.conf,在 report 选项里增加 tagmail 选项并以逗号分割。

[master]
reports = store,tagmail

2.修改/etc/puppet/tagmail 文件，增加 tags 标签，并配置相关的 e-mail 地址，
例如.下面示例 ，会将所有的日志信息发送到 john@bitfieldconsulting.com 邮箱。

all: john@bitfieldconsulting.com

2. 一旦客户端运行 puppet,你会收到像这样的一份邮件：

From: report@cookbook.bitfieldconsulting.com
Subject: Puppet Report for cookbook.bitfieldconsulting.com
To: john@bitfieldconsulting.com
Mon Jan 17 08:42:30 -0700 2011 //cookbook.bitfieldconsulting.com/
Puppet (info): Caching catalog for cookbook.bitfieldconsulting.com
Mon Jan 17 08:42:30 -0700 2011 //cookbook.bitfieldconsulting.com/
Puppet (info): Applying configuration version '1295278949'

如何做到：

puppet 会在 tagmail.conf 查找每一行，匹配 tag 标签并发送邮件到指定的邮箱，
特殊的标签会匹配所有的信息，错误标签匹配错误信息：
err: john@bitfieldconsulting.com

你可以定义很多规则在 tagmail.conf 文件里，puppet 会发送邮件到所有匹配的规则，
在下面的例子中，错误信息发送到一个电子邮件地址，而 web 服务器相关的信息发送到
另外一个电子邮件账户。

err: puppetmaster@example.com
webserver: webteam@example.com

额外更多：

tagmail 报告是一个非常有用的特性，你可能需要在实践中才能获得相关体会发，这里有些技巧：

什么是标签（tags）：
标签在本书的后面会更充分的解释，但为了使用这报告的目的，现在只要知道标签可以使用于节点（node）或者类（class）就足够了，（webserver 标签是匹配客户端执行 webserver 类）或者你直接像这样使用 tag 函数，添加一个标签：

```
class exim {
        tag("email")
        service { "exim4":
                ensure => running,
                enable => true,
        }
 }
```

指定多个标签，或者排除标签，你可以指定以逗号分隔的标记，在 tagmail.conf 文件里，也排除某些 tag 使用感叹号(!)的标签。！号表示非：

```
all, !webserver: puppetmaster@example.com
```

发送报告到多个电邮账户

你可以有规则将消息发送到多个电邮账户，用逗号分隔电子邮件地址：

```
 err: puppetmaster@example.com, sysadmin@example.com
```

创建图形化报告

让我们面对现实，老板更喜欢看直观图形化，puppet 可以使用 rrd 图形库产生合适的报告，为生成有度量的图形化，例如：每个客户端的运行时间等。

准备工作

你需要在系统上安装 rrd 工具以及 rrdtool 所需要的 ruby 链接库，例如: ubuntu,运行：

```
 # apt-get install rrdtool librrd-ruby
```

如何做

5. 在 puppet.conf 文件里添加 rrdgraph 添项：

reports = store,rrdgraph

如何工作的

每次运行，pupppet 会记录数据在客户端的目录（默认在/var/lib/puppet/rrd/<clientname>）.
它会创建 png 格式的图片，像事件，资源，以及传输时间，而你也可以使用第三方 rrd 工具，
来处理原始数据,原始数据是以.rrd 结尾的。

额外更多

如需要更详细的日志报告和图形，你可以使用 puppet dashboad.

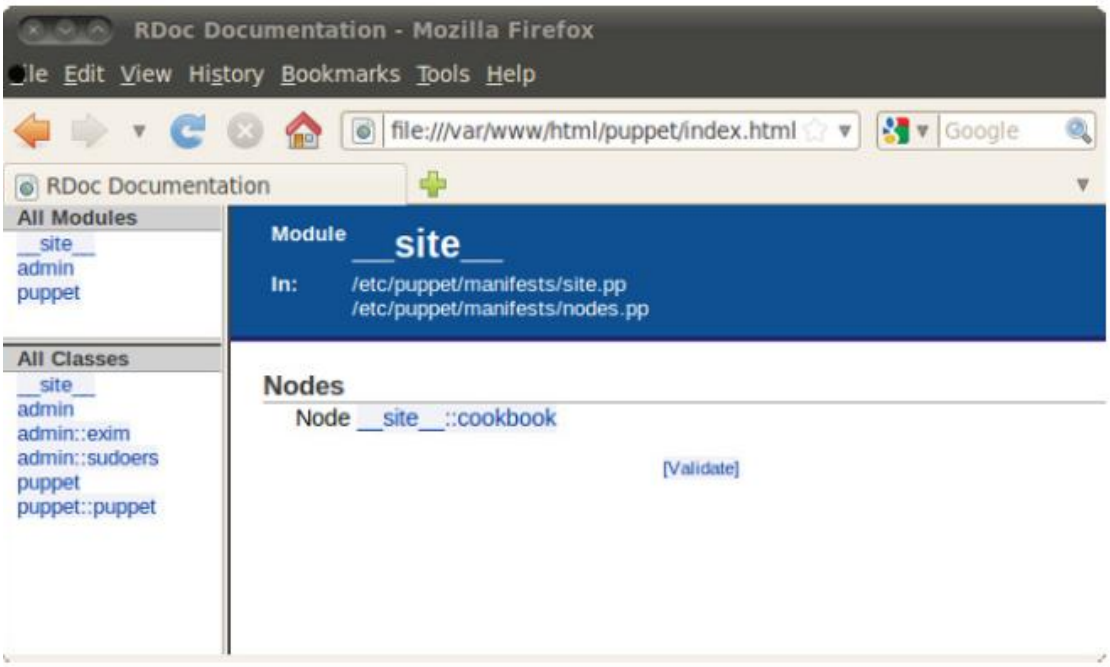
puppet dashboad 安装，可以参阅：
http://www.mysqlops.com/2011/10/28/puppet-dashboard.html

自动生成 html 文档

像大多数工程师一样，我从来没有阅读过手册，除非或者直到该应用十万火急的时候，
然而，由于你的代码杂乱而复杂，使用 puppet 自动化文档工具，例如:puppet doc ,
给每个节点和类创建 html 文档是非常有用.

1. 在你代码目录运行 puppet doc 命令：

puppet doc --all --outputdir=/var/www/html/puppet --mode rdoc --manifestdir=/etc/puppet/manifests/



是如何工作的

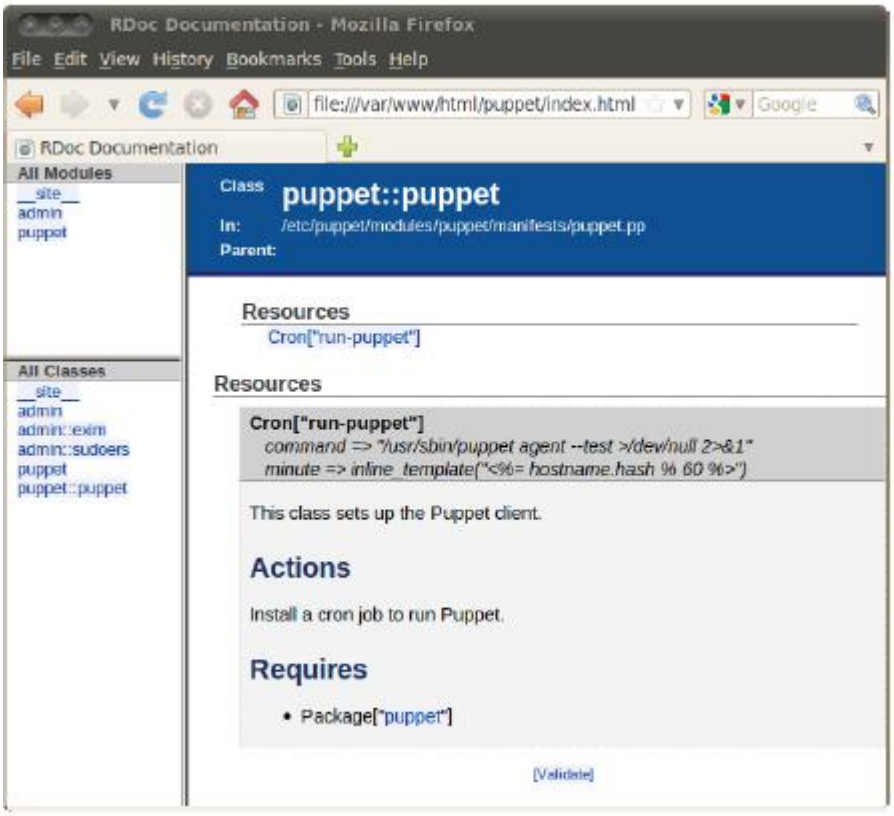
puppet doc 创建一个结构化 HTML 文档树在**/var/www/html/puppet** 类似 rdoc 生成的文档，rdoc 是流行的 ruby 文档生成器,这使得更容易理解不同部分代码之间的相互关系如何，你可以点击包含类的名称，并能看到它的定义，例 如：

还有更多

puppet doc 是标准的输出的，它会根据你的代码生成基本的文档，但是你可以在代码里，例用标准的 rdod 语法添加更多的注释，从而获得更多有用的信息，这里是添加一个类的文档并多处注释的例子。

```
class puppet {
    # This class sets up the Puppet client.
    # ==Actions
    # Install a cron job to run Puppet.
    #
    # ==Requires
    # * Package["puppet"]
    #
    cron { "run-puppet":
        command => "/usr/sbin/puppet agent --test >/dev/null 2>&1",
        minute    => inline_template("<%= hostname.hash.abs % 60 %>"),
    }
}
```

在生成的 HTML 文件里，你添加的注释在每个类的文档，像这样：



绘制依赖关系图

依赖使关系迅速变得复杂，并且很容易结束通知依赖，（其中 A 依赖 B，B 又依赖 A）这将引起 puppet 编译错误并停止工作，幸运的是，puppet 的图表选项可以很容易生成一个资源之间的依赖关系图，它可以帮助我们解决这些问题。

准备工作：

安装看图文件所需要的 graphviz 软件包：

 # apt-get install graphviz

创建/etc/puppet/modules/admin/manifests/ntp.pp 文件，并使用下面的代码包含一个通知依赖：

```
class admin::ntp {
        package { "ntp":
                ensure => installed,
                **require => File["/etc/ntp.conf"],**
        }
```

```
        service { "ntp":
                ensure    => running,
                require => Package["ntp"],
        }

        file { "/etc/ntp.conf":
                source    => "puppet:///modules/admin/ntp.conf",
                notify    => Service["ntp"],
                require => Package["ntp"],
        }
 }
```

2.复制已经存在的 ntp.conf 文件到 admin 模块 files 目录:

```
# cp /etc/ntp.conf /etc/puppet/modules/admin/files
```

3. 在客户端结点上执行这个类

```
 node cookbook {
    include admin::ntp
 }
```

4.客户端运行 puppet:

```
 # puppet agent --test
 info: Retrieving plugin
 info: Caching catalog for cookbook.bitfieldconsulting.com
 err: Could not apply complete catalog: Found dependency cycles in
the following relationships: File[/etc/ntp.conf] => Package[ntp],
Package[ntp] => File[/etc/ntp.conf], Package[ntp] => Service[ntp],
File[/etc/ntp.conf] => Service[ntp]; try using the '--graph'
option and open the '.dot' files in OmniGraffle or GraphViz
notice: Finished catalog run in 0.42 seconds
```

5. 查看图片文件是否已创建

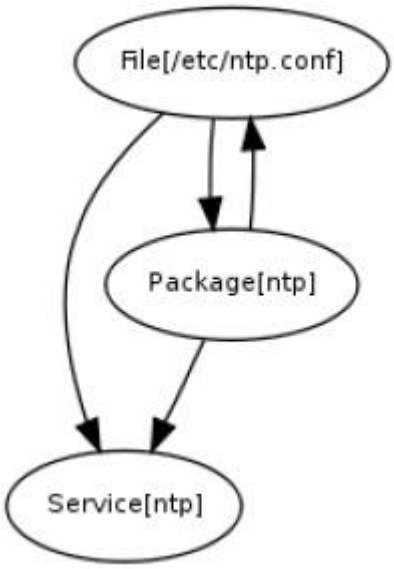
```
 # ls /var/lib/puppet/state/graphs/
 expanded_relationships.dot    relationships.dot    resources.dot
```

13.创建一个图形化的关系依赖图：

```
# dot -Tpng -o relationships.png /var/lib/puppet/state/graphs/relationships.dot
```

14. 查看图片

 # eog relationships.png



是如何工作的

当你使用 puppet --graph(或者在配置文件里启用图形化)，puppet 会生成三个 DOT 格式的三个文件（图形语言）
resource.dot 显示资源的类和次结构，但没有依赖关系。
releationships.dot 以箭头显示资源之间的依赖关系，如上图
expanded_relationships.dot 一个更详细的版本关系图
dot 工具（graphviz 软件包的一部分）可以将这些图像等转换为 png 格式进行查看。

在关系图中，每个资源在你的代码中显示为一个气球，用带箭头的线，连接他们表示依赖，在上面的例子中我们可以看到，在文件/etc/ntp.conf 与软件包 ntp 之间的依赖关系，
为了解决相互依赖关系的问题，所以你要做的就是删除依赖关系之一的线，因此可以打破循环。

还有更多

即使你不用去寻找 bug，资源和关系图都非常有用，如果你有一个非常复杂的网络类和资源，例如：研究资源图表可以让你看起来更简单，一目了然,同样，当依赖关系变得过于复杂，从阅读代码到理解，图表跟文档相比较是更有用的。

检查你的 puppet 代码

麻烦总是无孔不入，像标准的检查工具 nagios 不可能面面俱到能监控你要有的一切，如你

要衡量服务器负载和磁盘空间使用率两个指标，我更喜欢获得更多的服务器提供的应用以及服务信息。

例如：如果你运行的是一个 web 应用程序，你不能肯定在 web 服务器是监听 80 端口并能返回 HTTP 200 状态。也许它只是返回 Apache 的默认的 web 页面。

如果你的 web 应用程序是一个在线商店，例如：你可能希望检查以下几项：
请求 web 服务器，能否返回预期的页面（举例： “欢迎光临 FooStore”）？
用户能否正常登录（应用程序是否支持 session 会话）？
搜索某种产品能否返回搜索结果？
响应时间是否令人满意？
这种检查-专注于应用程序的行为，而不是对服务器本身业务的衡量，-能常被称为行为驱动检查。

每当修改代码的时候，开发人员经常使用行为驱动开发，以测试验证应用程序是否正常，你可以在整个项目周期里使用行为驱动来验证实现代码。

事实上，感谢有 cucumber-nagios 工具，你可以和开发人员使用同样的测试，Lindsay Holmwood 的封装是流行的 cucumber 测试框架，可以让你基于 nagios 运行 cucumber-based 测试，遵循标准的 nagios 的指标。

准备工作：

1.安装 cucumber-nagios,先需要安装相关依赖包，如你的系统是 Ubuntu 或者 Debian.
你可能需要从 gem 安装源中安装 RubyGems,cucumber-nagios 需要 ReubyGems1.3.6 或者更高版本。从 http://rubygems.org/pages/download 下载 ReubyGems 的源码包。

2.解压源码包并运行 ruby setup.rb 来构建并安装软件包。
3.接下来，你需要安装相关依赖包：
 # apt-get install ruby1.8-dev libxml2-dev
4. 最后才是安装 cucumber-nagios
 # gem install cucumber-nagios

如何做：

一旦 gem 包和相关依赖安装完成后，你可以开始写 cucumber 测试，要做到这一点，首先使用 cucumer-nagios 来帮助我们创建一个项目目录和所需要的一切。

```
# cucumber-nagios-gen project mytest
  Generating with project generator:
        [ADDED]    features/steps
        [ADDED]    features/support
        [ADDED]    .gitignore
```

[ADDED] .bzrignore
[ADDED] lib/generators/feature/%feature_name%.feature
[ADDED] Gemfile
[ADDED] bin/cucumber-nagios
[ADDED] lib/generators/feature/%feature_name%_steps.rb
[ADDED] README

新的 cucmber-nagios 项目录是放在/root/mytest 目录下,接下来安装所需要的 RubyGems.

bundle install

项目像 git 仓库一样已经初始化完成.

2. 有一个好主意是在项目目录里运行 bundle install 命令。像 cucumber-nagios 建议你的那样。cucumber-nagios 会在指定的目录下生成所有的依赖关系。

 # cd mytest
 # bundle install

3.我们开始写个测试示例，本例是用来检查 google 的主页。

 # cucumber-nagios-gen feature www.google.com home
Generating with feature generator:
[ADDED] features/www.google.com/home.feature
[ADDED] features/www.google.com/steps/home_steps.rb

3. 如果你编辑 home.feature 这个文件，你会发现 cucumber-nagios 会生成一个基本的测试。

Feature: www.google.com
It should be up

Scenario: Visiting home page
When I go to "http://www.google.com”
Then the request should succeed

5.在项目目录下运行下面命令：

 # cucumber --require features features/www.google.com/home.
feature
Feature: www.google.com
It should be up

Scenario: Visiting home page # features/www.google.com/home.feature:4
When I go to "http://www.google.com” # features/steps/http_steps.rb:11
Then the request should succeed # features/steps/http_steps.rb:64
1 scenario (1 passed)

2 steps (2 passed)
0m0.176s

6.假设这工作正常，（如果不正常，联系 Google）,为了使用带有 nagios 检查功能，你所要做的就是使用 cucumber-nagios 替代 cucumber。

 # bin/cucumber-nagios features/www.google.com/home.feature
CUCUMBER OK - Critical: 0, Warning: 0, 2 okay | passed=2;
failed=0; nosteps=0; total=2; time=0

它是如何工作：

任何脚本都可以作为 nagios 插件之一，它只是返回执行完成后退出的状态（0，为成功，1 为警告，2 为紧急）cucumber-nagios 使用 Cucumber 封装进行测试，还可以打印出有用信息，因为 nagios 通过警告或者 web 接口发送报告。

还有更多：

就其本身而言，你不用做任何事情都非常有用，但是，Cucumber 可以让你写出相当成熟与网站交互的脚本。
你可以填写表单字段，单击按扭搜索网页上匹配的字符串等等，你要监控你的 web 应用程序的任何功能或者任何服务，先弄清楚用户使用 web 浏览器习惯，然后根据这些用户行为，使用 Cucumber 创建自动监控脚本。

你可以从 http://cukes.info/网站上找到更多关于如何使用 cucumber 编写 cucumber-nagios 测试脚本。

做个试运行

“没有报警也没有惊喜。”
- Radiohead

我讨厌惊喜，有时自己写的 puppet 代码并没有达到自己的预期，或者别人在 puppet 检查后修改了你的代码，你并不知道，无论哪种方式也好，我们要精确知道 puppet 的执行结果.

如果更新了线上的某服务的配置文件，需要重启该服务，例如：这可能导致非计划性停机时间。此外，有时人为的手动编辑了配置文件被 puppet 同步更新时覆盖。为了避免这点问题，你可以使用 puppet 试运行模式，通常也被称为 noop 模式，即不执行或者无损运行.

如何去做：

1. 运行 puppet noop 模式：

```
# puppet agent --test --noop
info: Connecting to sqlite3 database: /var/lib/puppet/state/
clientconfigs.sqlite3
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1296492323'
--- /etc/exim4/exim4.conf      2011-01-17 08:13:34.349716342 -0700
+++ /tmp/puppet-file20110131-20189-127zyug-0       2011-01-31
09:45:27.792843709 -0700
@@ -1,4 +1,5 @@
  ##########
+# allow spammers to use our host as a relay
  ##########
notice: /Stage[main]/Admin::Exim/File[/etc/exim4/exim4.conf]/
content: is {md5}02798714adc9c7bf82bf18892199971a, should be {md5}
6f46256716c0937f3b6ffd6776ed059b (noop)
info: /Stage[main]/Admin::Exim/File[/etc/exim4/exim4.conf]:
Scheduling refresh of Service[exim4]
notice: /Stage[main]/Admin::Exim/Service[exim4]: Would have
triggered 'refresh' from 1 events
notice: Finished catalog run in 0.90 seconds
```

如何工作的

在 noop 模式下，puppet 会和通常运行的一样检查代码，只是不会对客户端产生实际影响。也就是说他将执行过程和结果输出，你可以和你所预期的进行对比，如果有什么不同，仔细检查代码或者机器的当前状态。

请注意，当 exim 的配置文件改变的时候，exim 服务会重新启动，这时候 puppet 会警告我们，这可能是或者不是我们当初的预期，但是事先知道这点非常有用，我制定了一个流程，当在生产线上服务器应用任何改变，首先使用 puppet noop 模式运行，验证发生的改变是否与我们预期一致。

还有更多...

你还可以使用试运行模式作为一个简单的审计工具，它会告诉你服务器 puppet 最近一次的应用改变。一些组组或者公司需要 puppet 应用所有的配置变化，这是实施一个变更的过程。使用 puppet 试运行模式可以检测到未授权的更改，也可以决定是否合并应用变化到 puppet 的代码，或者撤消。

另请参阅：

- 审计资源：
- 编辑错误检测：

通常情况下，在守护进程模式时，puppet 会忽略代码编译错误，只是从缓存中取最新的一个已知的工作版本，这种行为是由 usecacheonfailure 配置设置的，默认值为 true.

```
# puppet --genconfig |grep usecacheonfailure
  # usecacheonfailure = true
```

当你手动执行 puppet agent --test 应用变化时是值得注意的，不会发生这种情况，puppet 会拒绝做任何动作如果编译错误。那是因为-- test 开关是用于以下选项的所写：

```
# puppet agent --onetime --verbose --ignorecache --no-daemonize --no-usecacheonfailure
```

这是因为 usecacheonfailure 是在 puppet 作为守护进程的时候有效，有时候你不想看到代码错误输出通知，也就是说 puppet 静默的运行旧版本的代码，而不是每次都编译。

如何做

1. 如果你想改变这一行为，在 puppet.con 里添加下列值。

usecacheonfailure = false

是如何工作的

使用此选项设置，puppet 会立即停止错误代码编译并拒绝执行，直到代码正确。

理解 puppet 常见错误

puppet 的错误信息有时会造成混淆,而且没有包含关于如何解决这个问题的方法.通常我们看到错误信息第一反映就是去搜索网页关于错误信息解释以及获得解决问题的方法.这里是一些常见的令人头痛的错误以及尽可能明了的解释.

Failed to retrieve current state of resource: Could not retrieve information from source(s)

你为文件同步指定 source 参数,但是 puppet 却找不到该文件,请检查文件是否存在,以及 source 路径是否正确.

change from absent to file failed: Could not set ‘file on ensure: No such file or directory

通常引起这个报错是因为 puppet 尝试在某个目录写文件,但是该目录不存存.请检查该目录是否存在或者已经在 puppet 中定义了.另外请指定文件资源依赖于该目录(因为该目录必须先要创建)

undefined method `closed?' for nil:NilClass

这个错误消息可以解释为什么地方可能错了,这也意味可能是由多个原因引起的.但是你可以检查是什么资源或者类,或者说模块引起的.一个非常有用的方法,是使用--debug 开关来获得更多有用的信息:

 # puppet agent --test --debug

你可以检查下 git 日志信息,看看最近代码是否有所改动,这可能是另一种方式来确定是什么扰乱了 puppe 正常工作.

Could not parse for environment --- "--- production": Syntax error at end of file at line 1

这可能是由于命令行选项输错引起的.例如,如果你输入 puppet -verbose 而不是--verbose.这种错误一般是很难看到的.

Could not request certificate: Retrieved certificate does not match private key; please remove certificate from server and regenerate it with the current key

也许是节点的 sll 主机密钥发生了变化,或者 Puppet 的 SSL 目录被删除,或者你试图去请求证书,而 puppetmaster 上已经存在同名的节点.一般最简单的解决方法是在客户端上删除 puppet 的 ssl 目录(通常是在/etc/puppet/ssl)并在 puppetmaster 上使用 **puppet cert --clean** <nodename>删除原来的节点的 ssl 证书.然后再次运行 puppet,请求生成新的证书.

Duplicate definition: X is already defined in [file] at line
Y; cannot redefine at [file] line Y

在过去这曾经引起我的困惑.puppet 解析到资源重复定义,通常如果你有两个同名的资源,puppet 将会帮忙告诉你两者都定义了,但在这种情况下,它表示相同中的文件和行号,资源怎么可以重定定义吗?

答案就是 define.如果你使用 define 创建两个实例,你要在两个实例中包含所有的资源,并且他们需要不同的名称,例如:

```
define check_process() {
exec { "is-process-running?":
command => "/bin/ps ax |/bin/grep ${name} >/tmp/
pslist.${name}.txt",
}
}
check_process { "exim": }
check_process { "nagios": }
# puppet agent --test
info: Retrieving plugin
```

err: Could not retrieve catalog from remote server: Error 400 on
ERVER: Duplicate definition: Exec[is-process-running?] is already
defined in file /etc/puppet/manifests/nodes.pp at line 22; cannot
redefine at /etc/puppet/manifests/nodes.pp:22 on node cookbook.
bitfieldconsulting.com
warning: Not using cache on failed catalog
err: Could not retrieve catalog; skipping run

那是因为exec资源被名为 is-process-running.因为资源名相同,不管你传递什么参数到define. puppet 会拒绝创建两个实例.解决方法是为每个资源设置一个 title.

```
exec { "is-process-${name}-running?":
command => "/bin/ps ax |/bin/grep ${name} >/tmp/
pslist.${name}.txt",
}
```

记录命令输出

当你在客户端使用 exec 资源调用命令执行时，这时很难确认有没有执行成功。如果命令返回一个非零的状态，puppet 会返回如下错误信息：

```
 err: /Stage[main]//Node[cookbook]/Exec[this-will-fail]/returns:
change from notrun to 0 failed: /bin/ls file-that-doesnt-exist
returned 2 instead of one of [0] at /etc/puppet/manifests/nodes.pp:10
```

通常情况下，我们希望看到执行失败时的输出，而不仅仅是一个退出的状态码。你可以这样做使用 logoutput 参数.

如何做：

1. 定义一个 exec 资源，并像这样使用 logoutput 参数:

```
 exec { "this-will-fail":
       command      => "/bin/ls file-that-doesnt-exist",
       logoutput => on_failure,
 }
```

为什么能工作：

现在的话如果 command 执行失败，puppet 将会打印错误信息输出：

notice: /Stage[main]//Node[cookbook]/Exec[this-will-fail]/returns: /bin/ls: cannot access file-that-doesnt-exist: No such file or directory
err: /Stage[main]//Node[cookbook]/Exec[this-will-fail]/returns: change from notrun to 0 failed: /bin/ls file-that-doesnt-exist returned 2 instead of one of [0] at /etc/puppet/manifests/nodes.pp:11

额外更多：

你可以为 exec 资源设置一个默认值，执行命令失败打印错误输出。

```
Exec {
    logoutput => on_failure,
}
```

如果你想不管命令执行成功或者失败都输出错误信息，可以这样做：

```
logoutput => true,
```

记录调试信息

在调试问题的时候，如果能打印出代码中一定的信息，将非常有用。这是一个很好的工作方式，例如：如果一个变量没有定义或者定义了一个非法的值，知道一段特定的代码已运行，通常是有用的，（在调试的，设置断点）puppet 会通知资源，打印出调试信息。

怎么做...

1. 在你想要调试的代码里定义一个 notify 资源：

```
notify { "Got this far!": }
```

如何工作的...

当 puppet 编译到该资源的时候，puppet 将会打印出消息：

notice: Got this far!

还有更多：

如果你有勇敢的心，喜欢尝试，当然我也希望你是那样的一个人，你可能会发现自己使用调试信息，并清楚为什么代码不能工作。因此，如何获得更多的 Puppet 调试信息将非常有用。

打印出变量的值

你可以在消息中输出变量值

```
notify { "operatingsystem is $operatingsystem": }
 notice: operatingsystem is Ubuntu
```

打印出完整的资源路径

对于更高级的调试，你可能希望使用 withpath 的参数来查看类的 notify 资源被执行：

```
notify { "operatingsystem is $operatingsystem":
        withpath => true,
 }
```

现在你可以看到在通知信息前将给出资源的完整路径：

```
notice: /Stage[main]/Nagios::Target/Notify[operatingsystem is Ubuntu]/
message: operatingsystem is Ubuntu
```

在 puppetmaster 上记录日志信息

有时你想在 puppetmaster 上记录一条日志消息，而不会在客户端输出，你可以使用 notice 函数做到这一点。

```
 notice("I am running on node $fqdn")
```

这时你再运行 puppet，你在客户端上看不到任何输出，但对于 puppetmaster 来说会记录像这样的一条日志到系统日志：

```
Jan 31 11:51:38 cookbook puppet-master[22640]: (Scope(Node[cookbook])) I
am running on node cookbook.bitfieldconsulting.com
```

检查配置文件设置

你已经知道,puppet 的配置设置是保存在 puppet.conf 文件里，在该文件里没有提及的参数表示使用的是默认值，你怎么样才能显示所有的配置参数的值呢？不管它是否在 puppet.conf 文件里有明确设置？你可以使用 puppet 的 --genconfig 开关。

如何做：

1.运行

 # puppet –genconfig

这将输出每个配置参数以及参数的值（配置参数有许多）。但是，输出里包括解释每个参数说明非常有用，要找到你感兴趣的特定参数，你可以像这样使用 grep:

```
# puppet --genconfig |grep "reportdir ="
        reportdir = /var/lib/puppet/reports
```

blog 详见:http://www.mysqlops.com/2011/11/10/puppetd-conf.html

使用 tags（标签）

有时 puppet 的一个类需要知道另一类，或者，至少知道类是否存在。例如。puppet 管理防火墙的类可能需要知道该节点是否是一个 web 服务器。puppet 的 tagged 函数将会告诉你，在代码里检查类的名字或者资源标签是否存在。

1.为了帮助你查找出是要在特定的节点或者在所有节点上运行特点的类,那么所有
自动以节点名从父节点那里继承相应的属性.

```
node bitfield_server {
        include bitfield
 }

 node cookbook inherits bitfield_server {
        if tagged("cookbook") {
                notify { "this will succeed": }
        }
        if tagged("bitfield_server") {
                notify { "so will this": }
        }
 }
```

2.因此你可以告诉 puppet 在某个节点上执行一个特定类的，所有结点自动都会检查是否有 tagged 标签的类名包括其父类。

```
 include apache::port8000
```

```
if tagged("apache::port8000") {
        notify { "this will succeed": }
}
```

2. 如果你想给一个节点设置任意的 tag，可以使用标签（tag）函数。

```
tag("old-slow-server")
if tagged("old-slow-server") {
notify { "this will succeed": }
}
```

3. 如果你想对特定的资源设置一个标签，可以使用 tag 元参数。

```
file { "/etc/ssh/sshd_config":
source => "puppet:///modules/admin/sshd_config",
notify => Service["ssh"],
tag => "security",
}
```

5.你可以使用 tags（标签），可以确定应用 puppet 部分代码，如果你在 puppet 命令行下运行那些有明确的 tags（标签）的 pupet 类，或者特定资源，可以使用 --tags 选项，例如：如果你只想 Exit 更新配置，但是不想运行其它的代码。

```
＃puppet agent --test --tags exim
```

还有更多：

你可以使用（tags）标签来创建资源的集合，例如：如果一些服务依赖于许多片断文件。

```
class firewall::service {
service { “firewall”:
....
}
File <| tag == “firewall-snippet” |> ~> Service[“firewall”]
}
class myapp {
file { “/etc/firewall.d/myapp.conf”:
tag => “firewall-snippet”,
....
}
}
```

在这里，我们指定了任何标签为 firewall-snipped 资源更新了就必须通知防火墙服务，我们要做的就一切就是为特定的 app 或者服务添加一个防火墙配置并标签为 firewall=snippet,这样的话 puppet 会重新加载服务。虽然我们已经在每个资源片段添加了 notfiy=>Service["firewall"]，

如果我们定义防火墙服务随着代码文件不断更新而自动的变化。标签可以让我们把相应的代码进行逻辑封装在一个地，这样也使得将来的维护和重构更加容易。

使用运行阶段

一个常见的必要条件是应用某种资源之前必须安装其它依赖（例如：从仓库安装一个软件包）。或者之后运行其它的（例如：一当相关依赖全部安装好，就部署应用程序）puppet 的运行阶段允许你这样做。

```
class install_repos {
notify { "This will be done first": }
}
class deploy_app {
notify { "This will be done last": }
}
stage { "first": before => Stage["main"] }
stage { "last": require => Stage["main"] }
class { "install_repos": stage => "first" }
class { "deploy_app": stage => "last" }
```

2. 运行 puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303127505'
notice: This will be done first
notice: /Stage[first]/Beginning/Notify[This will be done first]/message:
defined 'message' as 'This will be done first'
notice: This will be done last
notice: /Stage[last]/End/Notify[This will be done last]/message: defined

'message' as 'This will be done last'
notice: Finished catalog run in 0.59 seconds
```

是如何工作的...

1.我们把想要做的事情分开，第一个事类和最后一个事类：

```
class install_repos {
notify { "This will be done first": }
}
```

class deploy_app {
notify { "This will be done last": }
}

2.接下来我们创建一个运行阶段名称为 first:
stage { "first": before => Stage["main"] }

before 参数指定了 first stage 阶段必须要在 main stage 之前完成所有的一切。也是默认的阶段。
3.接下来我们创建一个运行阶段，名称为 last:
stage { "last": require => Stage["main"] }
require 参数指定了 main stage 运行阶段必须要在 last 阶段完成所有一切。

3. 最后，我们执行两个类，分别为 install_repos 和 deploy_app,并指定他们分别应该是 first 和 last 两个运行阶段的一部分:

class { "install_repos": stage => "first" }
class { "deploy_app": stage => "last" }

请注意：我们使用的是 class 关键字，而不是 include.就像过支我们给类设置参数一样，你也可以认为，stage(阶段)可以看作是一个参数，参数通常可以随时传递到任何类中。
puppet 现在会按照如下阶段运行：
first
main
last

事实上，只要你喜欢你可以定义许多运行阶段，并为他们设立要运行的类，这可以极大的简化代码的复杂度，否则你就需要大量的分析资源之间的依赖性。如果你可以将资源分为 A 和 B 两组。A 资源组必须要在 B 资源之前运行完成，这是最好的候选方式，使用运行阶段，Gary Larizza 写了一篇非常有用的文章介绍如何使用运行阶段，并结合了实际的例子，地址为： http://glarizza.posterous.com/using-run-stages-with-puppet

使用多环境

你的代码运行环境是否友好，你是否要测试下 puppet 代码再应用到生产线上之前，你可以使用 puppet 的环境特性去做到,这可以让客户端根据所在的环境应用不同的 puppet 代码。
例如：你可以定义以下环境：
development
staging
production
你可以在 puppet.conf 文件里配置多环境，在下面的例子中，我们将添加了

一个 development(开发)环境，区别于其它的代码。

如何做到...

1. 在 puppet.conf 增加下面的行：

[development]
manifest = /etc/puppet/env/development/manifests/site.pp
modulepath = /etc/puppet/env/development/modules:/etc/puppet/modules

是如何工作的：

你可以自由添加你的代码环境到 puppet.conf 文件里，只要你在指向顶层的 site.pp 里设置的代码参数。在下面的例子中，我们将开发环境下的代码放在/etc/puppet/env/development 下，同样你需要设置开发环境的 modulepath 到你的模块目录下。
在上面的例子中，modulepath 已经包含了/etc/puppet/modules,如果是这样的话 puppet 在开发环境中找不到模块的话，它将会在默认环境中去找，这也就依味着你需要把你的模块复制到开发环境中去。

默认的环境为 production(生产环境)，所以如果你没有指定 puppet 的运行环境的话，puppet 将会使用默认环境。

还有更多...

如果你使用类似于 Git 的版本控制系统的话，你的环境可以是 Git 的分支，一旦你完成了测试和封装一个新的模块，你可以合并到 Git 的主分支，也就是生产线上所用的，你可以阅读更多关于如何使用环境在 R.I. Pienaar’s 的文章里，
http://www.devco.net/archives/2009/10/10/puppet_environments.php

你可以有好几种方式指定客户端所在的环境，你可以在运行 puppet 的时候，指定 --environment 开关：
 # puppet agent --test --environment=development
或者，你可以在客户端的 puppet.conf 文件里指定所属的环境。
[main]
environment=development
如果你使用的是外部节点分类的脚本（在本书的别处有描述），这也可以指定客户端所属的环境。

你也可以为每个环境指定不同的文件服务器，在 fileserver.conf，（可以在 puppetmaster 上查看配置片断）做到这一点，在 puppetmaster 的 puppet.conf 文件里设置 fileserverconfig 变量如下所示：

[development]

fileserverconfig = /etc/puppet/fileserver.conf.development
[production]
fileserverconfig = /etc/puppet/fileserver.conf.production

更多详情，可以在 puppetlabs 实验室里查看文档如何部署多环境：
http://projects.puppetlabs.com/projects/1/wiki/Using_Multiple_Environments

blog 详见:http://www.mysqlops.com/2011/11/15/puppet-environment.html

第三章 puppet 语言和规范

"Elegance is not a dispensable luxury but a factor that decides between success and failure."
— Edsger Dijkstra

在本章中我们将学习以下内容：
使用模块
使用标准的命名约定
使用公共的 puppet 规范
使用嵌入式的 ruby
写纯粹的 ruby 代码
编历多个项目
写出强健的条件语句
在 if 语句中使用正则表达式
使用选择器和条件语句
检查字符串中是否包含值
使用正则表达式替换

使用模块

最重要的一件事可以去做，那就是把 puppet 代码组织成模块，以更易于维护并使 puppet 的代码结构更加清晰。一个模块可以是一个简单的分组相关的东西：例如：web 服务器模块可以包括一个 web 服务器必需要的一切：Apache 的配置文件，虚拟主机模板，和必要的 puppet 代码去部署这些。

分离成模块，使得代码更容易重新使用和共享代码，也是最合乎逻辑的方式组织代码。在这个例子中，我们将创建一个模块用来管理 memcached.memcache 是一个内存缓存系统与 web 应用程序常用在一起。

怎么办呢...

1.明确你的模块路径，模块路径是在 puppet.conf 里设置，默认值是**/etc/puppet/modules**.如果你的代码已经使用的版本控制系统，像之前我建议你的方式去做，然后使用你的工作副本目录用来部署到/etc/puppet/moudules/，并替代原目录。

 # puppet --genconfig |grep modulepath
modulepath = /etc/puppet/modules:/usr/share/puppet/modules
 # cd /etc/puppet/modules

2. 创建 memcached 目录：

 # mkdir memcached

3. 在 memcached 目录下创 manifests 和 files 两个目录：

 # cd memcached
 # mkdir manifests files

4. 在 manifests 目录，创建 init.pp 文件，init.pp 文件内容如下：

import "*"

5. 在 manifests 目录，创建另外一个文件名称为 memcached.pp,其内容如下：

```
class memcached {
package { "memcached":
ensure => installed,
}
file { "/etc/memcached.conf":
source => "puppet:///modules/memcached/memcached.conf",
}
service { "memcached":
ensure => running,
enable => true,
require => [ Package["memcached"],
File["/etc/memcached.conf"] ],
}
}
```

6. 切换到 files 目录，创建 memcached.conf 文件，其内容如下：

-m 64

—p 11211
-u nobody
-l 127.0.0.1

7. 定义一个节点，并使用新的模块。

```
node cookbook {
include memcached
}
```

8.运行 puppet 测试新的配置

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1300361964'
notice: /Stage[main]/Memcached/Package[memcached]/ensure: ensure
changed 'purged' to 'present'
...
info: /Stage[main]/Memcached/File[/etc/memcached.conf]:
Filebucketed /etc/memcached.conf to puppet with sum a977521922a151
c959ac953712840803
notice: /Stage[main]/Memcached/File[/etc/memcached.conf]/content:
content changed '{md5}a977521922a151c959ac953712840803' to '{md5}f
5c0bb01a24a5b3b86926c7b067ea6ba'
notice: Finished catalog run in 20.68 seconds
# service memcached status
* memcached is running
```

是如何工作..

模块有特定的结构，这些目录不是都必须存在的，但是如果都存在，那么结构应该是像这样的：

```
MODULEPATH/
    MODULE_NAME/
        files/
        templates/
        manifests/
            init.pp
            ...
          README
```

puppet 会自动寻找并自动加载 init.pp 这个文件，这是模块在所有导入的类的时候必须的，

在我们的例子：

import "*"

memcache 类是定义在 memcached.pp 这个文件里，这将是由 init.pp 加载。现在的话，我们在结点上执行类：

include memcached

在 memached 类中，我们提到了 memcached.conf 文件：

```
file { "/etc/memcached.conf":
source => "puppet:///modules/memcached/memcached.conf",
}
```

正如我们在 puppet 的文件服务器中自定义挂载点里所看到的片断。 上面的 source 参数告诉 puppet 所要寻找的源文件路径。

```
MODULEPATH/
        memcached/
                files/
                    memcached.conf
```

还有更多

学习着去热爱模块，因为他们让你管理 puppet 的生活会轻松许多，模块并不复杂。然而，实践和经验会帮助我们判断何时应该组织划分为模块，以及如何更好的安排你的模块结构，这里有一些帮助你正确上路的技巧。

模板

模板作为模块的一部分，如果你需要使用模板，可以把模板放到 **MODULE_NAME/templates** 目录下，用法可以参考这样：

```
file { "/etc/memcached.conf":
content => template("memcached/memcached.conf"),
}
```

puppet 会在下面路径寻找文件：

```
MODULEPATH/
        memcached/
            templates/
                    memcached.conf
```

Facts,functions(函数)，types(类型)和 providers

模块可以包含自定义的 Facts,自定义函数，和自定义类型以及 providers.如需有关这些的详细信息，请参阅外部工具和 puppet 的生态系统(ecosystem)

第三方模块

你可以下载由其他人开发的模块，并可以在你的代码里使用模块，就像你自己写的模块一样，可以参考 puppet labs 网址：
http://projects.puppetlabs.com/projects/1/wiki/Puppet_Modules

使用标准的命名约定

给模块和类取个合适且简单明了的名称是非常有用的,尤其是别人要维护你的代码的时候,或者有其他人需要阅读并使用你的代码工作的时候.

怎么办呢...

1. puppet 的模块名以他们所管理的软件或者服务名:例如:apache 或者 haproxy
2. 以模块的功能或者所提供的服务来命名类名,例如:apache::vhost 或者 rails::dependencies.
   译者说明 apache 就是所提供的服务,vhost 是功能,中间以::分格.
3.如果模块内提供禁止提供某服务 ,就可以命名为 disabled.例如:一个用于停止 apache 的类应该被命令为
apache::disabled.
   译者说明:比如 iptables 服务,有时需要开启,或者需要关闭,那么就分成两个类,iptables::disabled 和 iptables::enable.
4. 如果一个节点需要提供多种服务,请在节点定义定义后为每个服务执行所需要的类,或者导入模块.

```
node server014 inherits server {
include puppet::server
include mail::server
include repo::gem
include repo::apt
include zabbix
}
```

5.管理用户的模块应该命名为 user.
6.在用户模块里,声明你的虚拟用户类名为 user::virtual
7.在用户模块里,为特定的用户群体可以设置为子类,子类应该被命名为用户组.举例,
user::sysadmins 或者 user::contractors.
8.如果你需要覆盖特定节点上的一个类或者服务,可以使用继承,继承类的话前缀是子类的名称.举例,如果一个节点名称为 cartman 需要特定的 ssh 配置,你想覆盖 ssh 类,可以这样做:

class cartman_ssh inherits ssh {
[override config here]
}

9.当你需要运行 puppet 为不同的服务布署配置文件时,完整的配置文件名以服务开头,使用点为分隔符,
后面为文件功能.举例:
Apache 的初始化脚本:apache.init
Rails 的 snippet 的定时处理日志配置文件:rails.logrotate
mywizzoapp 的 Nginx 的虚拟主机配置文件:mywizzoapp.vhost.nginx
standalone 服务的 mysql 配置文件:standalone.mysql

10.举例来说,如果你要管理不同的 ruby 版本,命名类名是有影响的,例如,ruby 1.92 或者 ruby 1.86

还有更多

在 puppet 公共社区维护着最佳实践准则为你的 puppet 的提供基础保障,其中包括关于一些命名的提示:
http://projects.puppetlabs.com/projects/1/wiki/Puppet_Best_Practice

使用公共的 puppet 规范

如果有人需要阅读或者维护你的代码,再或者你如果想共享代码到社区,那么就尽可能的遵守已经存在的规范约定,与公共社区紧密合作那是个好主意.

如何去做:
1.资源名通常都需要用双引号引起来,示例:
package { "exim4":
不是下面这样
package { exim4:

像一些连字符和空格容易引起 puppet 的解析器混淆,安全的做法是,也是明智的做法是把资源名称用双引号引起来.
2. 通常把参数值使用双引号引起来,只要那些不是 puppet 保留的字符,示例:

name => "Nucky Thompson",
mode => "0700",
owner => "deploy",
这些例外

ensure => installed,
enable => true,
ensure => running,w
3.在字符中引用变量,通常都使用大括号括起来.示例:

source => "puppet:///modules/webserver/${brand}.conf",

否则的话,puppet 的解析器需要去猜测哪些字符是变量名的一部分,使用大括号,使变量变得更加清晰.

3. 通常在最后一行使用逗号结尾,即使它只有一个参数也应该使用逗号.

service { "memcached":
ensure => running;
enable => true,
}

很多时候你在编辑文件并添加了额外的参数后,在原来的最后一行忘记添加必须要的逗号.
说明:在使用过程中最后一行是可以没有逗号结束的.建议还是写上.

4. 当声明一个资源只有一个参数的时候,所有参数放在一行,结尾可以没有逗号. 当资源有多个参数的时候,每个参数占一行:

package { "rake":
ensure => installed,
provider => gem,
require => Package["rubygems"],
}

6.当声明资源是链接,像这样使用 ensure =>link:
file { "/etc/php5/cli/php.ini":
ensure => link,
target => "/etc/php.ini",
}
8. 为了使代码更加容易阅读,以最长的参数为对齐,像这样的:

file { "/var/www/${app}/shared/config/rvmrc":
owner => "deploy",
group => "deploy",
content => template("rails/rvmrc"),
require => File["/var/www/${app}/shared/config"],
}

每个资源的箭头(=>)应该对齐,而不是整个文件,否则非常难如果你剪切或者复制到另外一个

文件.

额外更多:
Puppet Labs 网站上有 puppet 社区维护着 puppet 规范指南文档,url 地址为:http://projects.puppetlabs.com/projects/puppet/wiki/Style_Guide

使用嵌入式 Ruby

使用嵌入式 Ruby 是模板的强大的方法以之一,使用嵌入式 Ruby 能帮助建立动态文件配置和遍历数组,例如,你不需要使用一个单独文件就可以直接在你的代码里使用嵌入式 ruby,调用 **inline_template** 函数.

怎么办呢?
1. 在你的 puppet 代码里贴上 Ruby 代码 inline_template.

```
cron { "nightly-job":
command => "/usr/local/bin/nightly-job",
hour => "0",
minute => inline_template("<%= hostname.hash.abs % 60 %>"),
}
```

为什么能工作:
inline_template 里面的所有参数都会被传递并执行,就好像之前使用的 ERB 模板一样.也就是,**在<%=和%>分隔符之间的所有的都以 Ruby 代码来执行**,其余的被视为字符串.

另请参阅:
- 使用 ERB 模板
- 使用遍历数组生成模板文件

写纯粹的 ruby 代码

puppet 有时被指责,需要自己写些 provider 或者自定义函数,实现自己的特殊需求,而不是像现有的通用语言 ruby,不是每个人都认为这是缺点,计算机科学家 Dennis Ritchie 说过.
"A language that doesn't have everything is actually easier to program in than some
that do."
不论你是什么意见,这种指责声音已经不在,puppet 是支持用 ruby 语言写代码,在生产线上这是相当实用的,尽管还是处在早期,你可以混合和匹配 ruby 以及 puppet 代码在 puppet 文件里,puppet 是基于文件扩展名的来识别语言:**.rb 结尾的为 ruby 文件,.pp 结尾的为 puppet 文件.**

使用领域特定语言(DSL, domain-specific language)写代码,使用 ruby 看起来非常像标准的 puppet 的语法,在本章示例中,我会告诉你如何把一个典型的 puppet
代码转换为 ruby,这是原始的 puppet 代码.

```
class admin::exim {
package { "exim4": ensure => installed }
service { "exim4":
ensure => running,
require => Package["exim4"],
}
file { "/etc/exim4/exim4.conf":
content => template("admin/exim4.conf"),
notify => Service["exim4"],
require => Package["exim4"],
}
}
```

怎么做...

1. 创建 /etc/puppet/modules/admin/manifests/exim.rb 文件,内容如下:

```
hostclass "admin::exim" do
package "exim4", :ensure => :installed
service "exim4",
:ensure => :running,
:require => "Package[exim4]"
file "/etc/exim4/exim4.conf",
:content => template(["admin/exim4.conf"]),
:notify => "Service[exim4]",
:require => "Package[exim4]"
end
```

2.在节点上执行这个类,并运行 puppet.

为什么能工作...

1. hostclass 关键词声明为一个类,和 puppet 里面的 class 一样.

```
hostclass "admin::exim" do
```

2.然后我们看上面的代码,有 do...end 这个块,这相对于 puppet 里的大概括的功能.3.在调用函数前我们先声明资源类型,示例 ,软件包或者服务.

```
package "exim4", :ensure => :installed
```

4.参数传递到函数中是以逗号分隔的列表,参数名必须要用双引号引起来,或者在参数前使用一个冒号使它们看起来是 Ruby Symbol.

译者说明,Symbol 是什么?在 Ruby 中 Symbol 表示“名字”，比如字符串的名字，标识符的名

字。创建一个 Symbol 对象的方法是在名字或者字符串前面加上冒号：
:ensure => :running,

再次像 puppet 内置的参数一样:installed 或者:runing 是 Ruby Symbols.
5.当我们需要分析资源之间的相互关系,例如: require,标识资源第一个字母大写并且资源名称写在方括号中.
:require => "Package["exim4"]"
6.使用 template 名称和圆括号,参数是在方括号中间,这样我们可以调用类似模板的功能.
:content => template(["admin/exim4.conf"]),

还有更多...

我很负责任的说,其实我不建义你使用 Ruby DSL 来写 puppet 代码,虽然实验很有趣,但真正要使用 Ruby,除非有令人兴服的理由,我会坚持一直到现在的标准的 puppet 语言,Ruby 的 DSL 或许将来会广泛使用,但是,我怀疑它,如果你坚持要使用,然后,这里有几个得心应手的提示.
:content => template(["admin/exim4.conf"]),

变量
你和平时一样使用 ruby 变量只要遵照 ruby 语法,你可以像这样使用 scope.lookupvar 来查看 puppet 变量.
notify (["I am running on node %s" % scope.lookupvar("fqdn")])
gives:
notice: I am running on node cookbook.bitfieldconsulting.com

在你的 puppet 代码里设置一个变量,使用 scope.setvar:
require 'time'
scope.setvar("now", Time.now)
notify (["At the third stroke, the time sponsored by Bitfield
Consulting will be: %s" % scope.lookupvar("now")])
gives:
notice: At the third stroke, the time sponsored by Bitfield Consulting
will be: Wed Mar 23 05:58:16 -0600 2011

文档:
你可以在 Puppet labs 实验室了解更多关于如何使用 Ruby DSL,包括更高级的主题,例如:虚拟资源和 collections.
http://projects.puppetlabs.com/projects/1/wiki/Ruby_Dsl

Ken Barber 提供了些语法例子,并且把 puppet 语法和 Ruby DSL 做了详细的结构比较,
url 为 https://github.com/bobsh/puppet-rubydsl-examples
最后,James Turnbul 曾经写了一篇 blog,演试了使用 Ruby 连接数据库的更优的方案.
http://www.puppetlabs.com/blog/using-ruby-in-the-puppet-ruby-dsl/

遍历数组

puppet 中数组是个强大的特性之一,不论你何时对列表中的字符串执行相同的操作,
数组可能会帮你忙.你可以创建一个数组,只需要其内容放在方括号中([]).
$lunch = ["eggs", "beans", "chips"]

如何做...

1. 增加以下内容到你的 puppet 代码里:

```
$packages = [ "ruby1.8-dev",
"ruby1.8",
"ri1.8",
"rdoc1.8",
"irb1.8",
"libreadline-ruby1.8",
"libruby1.8",
"libopenssl-ruby" ]
package { $packages: ensure => installed }
```

2.运行 puppet 即可看到每个软件包都会被安装.

是如何工作的...

Puppet 在遇到数组的时候,会把数组名作为一个资源,它会为每个数组的元素创建一个资源,在本例中,会为$packages 数组里的每个元素创建一个新的 package 资源,并使用参数(ensure=>installed.)这也是一个非常简洁的方式初始化很多类似的资源.

还有更多...

如果你听到哈希,会比数组更兴奋.

哈希

哈希和数组一样,但是它的每个元素都可以通过名称查找并且可以通过名字来存储的.

例如:

```
$interface = { name => 'eth0',
address => '192.168.0.1' }
notice("Interface ${interface[name]} has address
${interface[address]}")
Interface eth0 has address 192.168.0.1
```

你可以给哈希赋任意的值:字符串,函数调用,表达式,甚至其它哈希或数组。

使用 split 创建数组

你可以使用方括号来声明数组,示例:

```
define lunchprint() {
notify { "Lunch included $name": }
}
$lunch = [ "egg", "beans", "chips" ]
lunchprint { $items: }
Lunch included egg
Lunch included beans
Lunch included chips
```

但是 puppet 也可以从字符串创建数组,像这样使用 split 函数:

```
$menu = "egg beans chips"
$items = split($menu, ' ')
lunchprint { $items: }
Lunch included egg
Lunch included beans
Lunch included chips
```

请注意:**split 需要两个参数,第 1 个是需要 split 的字符串,第二个是 split 的分隔**符.在本例中,分隔符是空格.puppet 会把它当成字符串,一遇到空格,它会认为是字符串的开始和结束,如此下去.因此,上面会显示"egg beans chips",会被分割成三个项目.可以以任意字符作为分割符,也可以是字符串:

```
$menu = "egg and beans and chips"
$items = split($menu, ' and ')
```

也可以是正则表达式,举例:使用|来分割字符串:

```
$lunch = "egg:beans,chips"
$items = split($lunch, ':|,')
```

写出强健的条件语句

puppet 的 if 语句允许你根据变量的值或者表达式的值应用不同的代码,有了它,你就可以根据客户端 facter 的探测结果应用不同的资源或者参数值-例如:操作系统,内存大小,你还可以在执行类的动作的代码里设置变量. 例如,在数据中心 A 的节点需要使用不同的 DNS 服务器跟数数据中心 B 的服务相比.或者你需要在 Ubuntu 系统上执行一个类或者一组类,对于不同的操作系统来说.

1. 增加下面的代码的到 manifests

```
if $lsbdistid == "Ubuntu" {
```

```
notice("Running on Ubuntu")
} else {
notice("Non-Ubuntu system detected. Please upgrade to Ubuntu
immediately.")
}
```

Puppet 对待任何以关键词 if 开头的,后面的作为一个表达式并评估,如果表达式的结果为值, Puppet 会执行花括号中的代码 ,或者,你可以添加一个 else 分支,如果表达式的计算结果为 false,这段代码会被执行.

还有更多...

你可以在 puppet 中编写非常复杂的 if 条件语句,但是我建议你不要,通常最好是改变你的设计思路(例如,使用模板)比使用 if 语句要好.通常,例如:我在生产线上实际应用的一些例子,我惊讶的发现在我的成千上万行中的代码里我没有使用 if 语句,尽管如此,你的实际应用会有所不同,因此这里有些关于使用 if 的技巧.

elsif

你可以像这样的进一步添加条件语句,使用 elsif 关键词.

```
if $lsbdistid == "Ubuntu" {
notice("Running on Ubuntu")
elsif $lsbdistid == "Debian" {
notice("Close enough...")
} else {
notice("Non-Ubuntu system detected. Please upgrade to Ubuntu
immediately.")
}
```

比较

你可以使用==检查两个值是否相等,如下面的例子:

```
if $lsbdistid == "Ubuntu" {
...
}
```

或者你也可以使用!=检查两个值不相等:

```
if $lsbdistid != "CentOS" {
...
}
```

你还可以使用<和>比较两个数值:

```
if $uptime_days > 365 {
notice("Woohoo!")
}
```

如果要测试一值是否小于等于(或者大于等于)另外一值,使用<=或者>=:

```
if $lsbmajdistrelease <= 9 {
...
}
```

组合表达式

你可以把上述的简单的表达式结合成更复杂的逻辑表达式,使用 and(和),or(或者) 和 not(非):

```
if ($uptime_days > 365) and ($lsbdistid == "Ubuntu") {
...
}
if ($role == "webserver") and ( ($datacenter == "A") or ($datacenter
== "B") ) {
...
}
```

在 if 语句中使用正则表达式

你可以在 if 语句中测试另外一种表达式就是正则,正则表达式是一强大的字符串比较方式.

怎么办呢...

1. 添加以下内容到你的代码:

```
if $lsbdistdescription =~ /LTS/ {
notice("Looks like you are using a Long Term Support version
of Ubuntu.")
} else {
notice("You might want to upgrade to a Long Term Support
version of Ubuntu...")
}
```

这是如何工作的

puppet 把斜线之间的文本当作正则表达式,而斜线之间的就是要匹配的文本.如果 if 表达式为真,作为一个整体将匹配成功,大括号之间的第一代码将被执行。

如果你想不匹配文本,需要做些什么,可以使用!~ 而不是=~.

if $lsbdistdescription !~ /LTS/ {

还有更多...

Jamie Zawinski 曾经说过:

Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems.

正则表达式是非常强大的,但是很难理解和调试,一旦你发现自己使用如此复杂的正则表达式,而你又不清楚的知道它是做什么的,那么只能思考简化设计,使其变得更加易懂,然而,正则表达式的一个特别有用的功能是能够使用捕获模式.

捕获模式

你不仅可以使用正则表达式来匹配文本,你也可以使用捕获匹配的文本并把它存储在一个变量中.

译者说明,捕获模式,就是获得用户输入.

```
$input = "Puppet is better than manual configuration"
if $input =~ /(.*) is better than (.*)/ {
notice("You said '$0'. Looks like you're comparing $1 to $2!")
}
```

你输入'Puppet is better than manual configuration',看起来你是比较 Puppet 和人工配置!

变量$0 是存储的整个匹配的文本(假设整体匹配成功).如果你在括号里有正则表达式的的任何部分,,那样的话会创建一个组,所有匹配的组的都将被存储为变量,因此,在上面的例子.第一个匹配的单元将是$1,第二个为$2,

正则表达式的语法

puppet 使用的是 ruby 正则表达式的一个字集,如果你不是很熟悉正则表达式,这个地址将非常有用:

http://gnosis.cx/publish/programming/regular_expressions.html

使用 selectors 和 case 语句

虽然你可以使用 if 写出任何条件语句,puppet 支持额外的表单即 select 和 case 语句,以帮助你更容易地使用条件表达式.

怎么做:

1.添加下列内容到你的 puppet 代码:

```
$systemtype = $operatingsystem ? {
"Ubuntu" => "debianlike",
"Debian" => "debianlike",
```

```
"RedHat" => "redhatlike",
"Fedora" => "redhatlike",
"CentOS" => "redhatlike",
}

notify { "You have a ${systemtype} system": }
```

2.添加下列内容到你的 puppet 代码:

```
class debianlike {
notify { "Special manifest for Debian-like systems": }
}
class redhatlike {
notify { "Special manifest for RedHat-like systems": }
}
case $operatingsystem {
"Ubuntu",
"Debian": {
include debianlike
}
"RedHat",
"Fedora",
"CentOS": {
include redhatlike
}
}
```

它是如何工作的...

我们的示例演示了 selector 和 case 语句的用法,接下来让我们详细了解他们是如何工作的.

Selector

在第一个例子中,我们使用了一个 selector(the ? operator)会为$systemtype 选择一个值,而$systemtype 的值依赖于$operatingsystem 的值.这有点类似于 C 或者 Ruby 中的三元操作.
不同的是从两个选项中选择一个值,selector 你可以有多个值供选择,只要你想提供多个值.

Puppet 会比较我们提供的每一个可能,它就是$operatingsystem 的值:"Ubuntu","Debian"等等.
这些值可以是正则(字符串匹配部分,或者使用通配符,例如),但在我们例子中,我们只用文字字符串,一当找到一个匹配,selector 表达式会返回与之匹配的字符串的值作为值,如果$operatingsystem 的值是"Fedora",如上例子:selector 表达式会返回"redhatlike"字符串,
所以这将是分配给$systemtype 变量的值.

case 语句

不像 selectors, case 语句不返回值,case 语句是依赖于表达式的的不同值执行不同的代码.在我们的第二个例子,我们使用的是 case 语句去执行 debianlike 类,或者执行 redhatlike 类,这些都取决于$operatingsystem 的变量值.

再次说明,puppet 会根据$operatingsystem 的值去和潜在的列表比较,这些可以是正则表达式,字符串,或者像上例子中,以逗号分隔的列表字符串.一旦匹配,就会执行相关联的大概括号之间的代码 .因此,如果$operatingsystem 的值是"Ubuntu",那么的 debianlike 的类将会被执行.

还有更多...

一旦你熟悉使用 selectors 和 case 语句,你会发现下面的技巧非常有用.

正则表达式

与 if 语句一样,你也可以在 selectors 和 case 语句中使用正则表达式,你也可以捕获匹配单元,并可以使用$1,$2 来引用变量等.

```
case $lsbdistdescription {
/Ubuntu (.+)/: {
notify { "You have Ubuntu version $1": }
}
/CentOS (.+)/: {
notify { "You have CentOS version $1": }
}
}
```

Defaults
selectors 和 case 语句都可以让你指定一个默认值,这是在没有其它选项匹配的时候才使用:

```
$lunch = "Sausage and chips"
$lunchtype = $lunch ? {
/chips/ => "unhealthy",
/salad/ => "healthy",
default => "unknown",
}
notify { "Your lunch was ${lunchtype}": }
Your lunch was unhealthy
```

检查字符串是否包含特定的值

Puppet 2.6 引入了新的表达式,像这样使用关键词:in

"foo" in $bar
如果字符串 foo 是$bar 的子串,那么值为真,如果$bar 是数组,那么条件表达式为值,如果 foo 是数组的一个元素,或者 $bar 是一个 hash,如果 foo 是($bar)hash 的一个 key 值,那么表达式为值.

如何做...
1.添加以下内容到你的 puppet 代码:

```
if $operatingsystem in [ "Ubuntu", "Debian" ] {
notify { "Debian-type operating system detected": }
} elsif $operatingsystem in [ "RedHat", "Fedora", "SuSE", "CentOS"
] {
notify { "RedHat-type operating system detected": }
} else {
notify { "Some other operating system detected": }
}
```

2.运行 puppet:

```
# puppet agent --test
Debian-type operating system detected
```

它是如何工作的...

没有必要解释.

还有更多...

在表达式中不是只可以使用 if 语句或者其它条件语句,任何表达式都可以使用.因此,例如,你可以给变量分配一个值:

```
$debianlike = $operatingsystem in [ "Debian", "Ubuntu" ]
if $debianlike {
$ntpservice = "ntp"
} else {
$ntpservice = "ntpd"
}
```

使用正则表达式与替换

puppet 提供了 regsubst 函数来轻松处理文本,在字符串内搜索与替换,或者从字符串提取匹配模式,例如,通常我们需要处理从 facter 获得的变量,或者从外部程序获得的 facter.

在下面的例子,我们看到如何使用 regsubst 提取一个 IP 地址的前三个 8 位字节(网络的一部分,

假设是一个 C 类地址).

1.添加以下内容到你 puppet 代码里
$class_c = regsubst($ipaddress, "(.*)\..*", "\1.0")
notify { $ipaddress: }
notify { $class_c: }
2.运行 puppet
notice: 10.0.2.15
notice: 10.0.2.0

它是如何工作的...
regsubst 最少需要三个参数:源,匹配和替换,在我们的例子中,我们指定了$ipaddress 的值作为 source 参数,source 的值为 10.0.2.15
匹配部分为:
(.*)\..*

要替换的部分为:
\1.0

匹配为整个 IP 地址,捕获圆括号中的前三位 8 个字节,捕获的文本将在替换字符中使用\1(从源字符串捕获的文本),接下来的是字符串.0 来代替,计算结果为:
10.0.2.0
关于 regsubst 的介绍可以参阅官方文档:
http://docs.puppetlabs.com/references/stable/function.html

还有更多...
模式匹配可以是任何正则表达式,在 if 语句中可以同样使用相同的(Ruby)语法.

第四章　　编写更好的 puppet 代码

"An expert is someone who is one page ahead of you in the manual."
— David Knight

在本章中,我们将学习以下内容:
- 使用数组资源
- 使用 defines
- 使用依赖关系
- 使用节点继承(inheritance)
- 使用类继承和覆盖
- 类的参数传递
- 编写可重用的,跨平台的代码

- 获取环境变量
- 使用 generate 生成动态信息
- 导入 CSV 文件的数据
- 传递参数到 shell 命令

介绍

你的 pupept 代码是整个基础设施的生活的文档,保持它整洁和良好的结构是
个非常好的方式 ,使其更易于维护和理解.Puppet 给了您很多工具去做到这点,
工具包括有这些:

Arrays (数组)
defines (定义)
Dependencies (依赖)
Inheritance (继承)
Tags (标签)
Class parameters(类参数)
Run stage (运行阶段)

我们将看到如何使用所有上面的工具,甚至更多.当你通过阅读本章,试验着上面例子,看到这些功能可是否可以帮助你的 puppet 代码进行简化和改善,和自己的代码相比较.

使用数组资源

"If we wish to count lines of code, we should not regard them as lines produced but as lines spent."

— Edsger Dijkstra

puppet 将所有的都识为资源,你也可以处理数组资源,使用这种方法重构你的代码,使其更短,更清晰.

1.怎么办?

你有同类的几个实例,可以放到一个类中,例如,软件包 :

```
package { "sudo" : ensure => installed }
package { "unzip" : ensure => installed }
package { "locate" : ensure => installed }
package { "lsof" : ensure => installed }
package { "cron" : ensure => installed }
package { "rubygems" : ensure => installed }
```

2.把它们放在一起,做成组,并使用数组来取代单一的 package 资源.

```
package { [ "cron",
```

```
"locate",
"lsof",
"rubygems"
"screen",
"sudo"
"unzip" ]:
ensure => installed,
}
```

它是如何工作...

Puppet 的大部分资源都可以接受一个数组来代替单一的名称,并会为每个数组的元素创建一个实例,你提供每个资源的参数(例如,ensure=>installed)都会分配都新的资源实例.

另请参阅

遍历多个项目

使用定义(defines)

在前面的例子中,我们看到了使用数组减少相同的冗余代码.然而,这项技术是有限制的,那就是所有的参数必须相同.当你要设置一个资源,有一些共同的参数和有些不同的参数,你需要使用 define 把它们放在一起进行分组.

如何做...

1.添加下面内容到你的代码:

```
define tmpfile() {
file { "/tmp/$name":
content => "Hello, world",
}
}
tmpfile { ["a", "b", "c"]: }
```

2.运行 Puppet:

```
notice: /Stage[main]//Node[cookbook]/Tmpfile[a]/File[/tmp/a]/
ensure: defined content as '{md5}bc6e6f16b8a077ef5fbc8d59d0b931b9'
notice: /Stage[main]//Node[cookbook]/Tmpfile[b]/File[/tmp/b]/
ensure: defined content as '{md5}bc6e6f16b8a077ef5fbc8d59d0b931b9'
notice: /Stage[main]//Node[cookbook]/Tmpfile[c]/File[/tmp/c]/
ensure: defined content as '{md5}bc6e6f16b8a077ef5fbc8d59d0b931b9'
```

它是如何工作...

你可以认为 define 是千篇一律的,它描述一个模式,可以让 Puppet 来创建大量类似的资源.你可以在任何时候在代码里声明 tmpfile 实例,puppet 将会插入 tmpfile 定义的资源.

在我们的例子中,tmpfile(函数)包含了单个文件资源,内容为"Hello,world",文件完整路径为/tmp/${name}.如果你像这样声明一个实例名称为 foo:tmpfile { "foo": }

那样 puppet 在/tmp 下会创建一个文件,文件名为 foo.换句话说.define 中的${name}会被替换为任何实际的实例名称 ,也就是 puppet 所要创建的, 它几乎就像我们创建了一种新的资源:tmpfile,它有一个参数 ,就是它的名称.

就像我们经常使用的资源,我们没有必要只传递一个名称的参数:我们可以提供一个数组名称,puppe 将会创建大量的 fmpfiles,像上面的例子.

还有更多...

在下面的例子中,我们创建了一个函数(define),并只添加了一个参数,参数又是实例的名字.但是,我们也可以添加任何我们想要的参数,因此我们只要在函数中进行声明.

```
define tmpfile( $greeting ) {
file { "/tmp/$name":
content => $greeting,
}
}
```

在声明实例资源的时候,并传递相应的变量值.

```
tmpfile{ "foo": greeting => "Hello, world" }
```

你可以声明多个参数列表,**以逗号分割:**

```
define webapp( $domain, $path, $platform ) {
...
}
webapp { "mywizzoapp":
domain => "mywizzoapp.com",
path => "/var/www/apps/mywizzoapp",
platform => "Rails",
}
```

抽象出资源之间的共性,并保存在一个地方,你可以不需要重复自己劳动(Don't Repeat Yourself)是一个非常有用的技术,在上面的例子中,有可能有在其 webapp 里,有许多独立的资源:软件包,配置文件,源代码 checkout,虚拟主机等.除了我们提供的参数,他们都是 webapp 的

应用实例这样也可以引用模板,例如,给虚拟主机设置域名.

使用依赖关系

为确保事情按照正确的顺序执行,你可以在 puppet 中指定资源之间依赖关系,例如,你要启动 X 服务,需要先安装 X 软件包,因此,你就需要标记服务依赖于软件包.Puppet 会排序所有的依赖关系顺序,并满足所有的依赖.

在一些配置管理系统里,资源的执行顺序是根据你写的代码顺序-换句话说,顺序是比较隐式,这不是 Puppet 的问题的,如果要应用一个或者更多的资源,执行顺序是随机的,除非你明确的使用依赖关系,有些人喜欢隐式的话,因为你可以按照顺序写资源定义,那么就按顺序执行.

另一方面 ,在许多情况下,资源的顺序并不重要.有一个隐式的风格系统.你不能告诉在列出资源 A 的时候是否要依赖资源 B,或者说资源 A 恰好被列在首位,这使得重构更加困难,移动资源有可能会打破一些隐式的依赖关系.Puppet 会让你做一点额外的工作,那就是预先在资源前指定依赖,但是结果是代码更加清晰,更易于维护.让我们来看下面的一个例子.

怎么办呢...

1.创建一个新文件名为/etc/puppet/modules/admin/manifests/ntp.pp ,其内容如下:

```
class admin::ntp {
package { "ntp":
ensure => installed,
}
service { "ntp":
ensure => running,
require => Package["ntp"],
}
file { "/etc/ntp.conf":
source => "puppet:///modules/admin/ntp.conf",
notify => Service["ntp"],
require => Package["ntp"],
}
}
```

2.复制已经存在的 ntp.conf 文件到 Puppet 到 admin 模块下的 files 目录.

```
# cp /etc/ntp.conf /etc/puppet/modules/admin/files
```

3.在 nodes.pp 里添加要执行 admin::ntp 类的节点.

```
node cookbook {
include admin::ntp
}
```

4.现在删除已经存在的 ntp.conf 文件:
 # rm /etc/ntp.conf

5.运行 Puppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1302960655'
notice: /Stage[main]/Admin::Ntp/File[/etc/ntp.conf]/ensure:
defined content as '{md5}3386aaad98dd5e0b28428966dac9e1f5'
info: /Stage[main]/Admin::Ntp/File[/etc/ntp.conf]: Scheduling
refresh of Service[ntp]
notice: /Stage[main]/Admin::Ntp/Service[ntp]: Triggered 'refresh'
from 1 events
notice: Finished catalog run in 2.36 seconds

它是如何工作的...

这个例子演示了两种依赖:require 和 notify.在第一种情况下,要运行 NTP 服务,NTP 软件包必须要先安装:

```
service { "ntp":
ensure => running,
require => Package["ntp"],
}
```

在第二种情况下,设置了 NTP 配置文件需要通知 NTP 服务,换句话说,如果 NTP 配置文件有改动,Puppet 就重启 ntp 服务以应用新的配置.

```
file { "/etc/ntp.conf":
source => "puppet:///modules/admin/ntp.conf",
notify => Service["ntp"],
require => Package["ntp"],
}
```

这意味着,ntp 服务依赖于配置文件,以及软件包的安装,Puppet 可以应用三种资源,按照下面的顺序正确执行:
Package["ntp"] -> File["/etc/ntp.conf"] ~> Service["ntp"]

事实上,这是另一种方式来指定相同的依赖链,添加上面一行添加到你的 puppet 代码里,如上例,将有和使用(require 和 notify 参数)同样的效果(->是表示 require 而~>表示是 notify),但是我更喜欢使用 require 和 notify,因为依赖关系是资源定义的一部分,因此容易调试代码,接下来代码会做什么,不过,对于复杂的依赖关系链,你可能想使用->符号来代替.

还有更多...

你也可以指定某种资源依赖于某些类:
require => Class["my-apt-repo"]

你也可以指定依赖关系不只是资源和类,也可以是 collections:
Yumrepo <| |> -> Package <| provider == yum |>
这是一个功能强大的方式,说明:所有软件包需要通过 yum 来安装,yum 并且需要依赖 yumrepo 资源.

使用节点继承

比方说,你的服务器托管在三个不同的地方:WreckSpace,GoDodgy, 和 VerySlow.他们是不同的数据中心以及不同的地理位置(国内一般就是不同的 IDC),所以你需要为你的配置文件作出小的修改,以适应每个不同的数据中心的服务器.你有几中不同类型的服务器,他它们是随机分布在三个不同的数据中心.

一个实现方法是,在 Pupppet 节点里使用 definiton 来定义变量实现.

```
node webserver127 {
$provider = "VerySlow"
include admin::basics
include admin::ssh
include admin::ntp
include puppet::client
include backup::client
include webserver
}
node loadbalancer5 {
$provider = "WreckSpace"
include admin::basics
include admin::ssh
include admin::ntp
include puppet::client
include backup::client
include loadbalancer
}
```

正如你所看到的,结果是有大量的重复行,如果我们只是来定义一个"WreckSpace server",这比上面容易得多,例如,我们可以创建一个节点继承于另外一个节点,只需要确定执行什么类,负载均衡器或者 web 服务器.

怎么办呢:

1.为所有节点创建一个基类,这个基类是包含所有的节点都需要的操作.

```
node server {
include admin::basics
include admin::ssh
include admin::ntp
include puppet::client
include backup::client
}
```

2.为这个 server 节点创建三个不同的子类,并设置 provider 的变量.

```
node wreckspace_server inherits server {
$provider = "WreckSpace"
}
node gododgy_server inherits server {
$provider = "GoDodgy"
}
node veryslow_server inherits server {
$provider = "VerySlow"
}
```

3.假设我们现在要在 VerySlow 中创建一个新的 web 服务器,要做到这一点,只需要从 veryslow_server 继承:

```
node webserver904 inherits veryslow_server {
include webserver
}
```

是如何工作的...

当一个节点从另一个节点继承,他会应用父节点的所有配置,你也可以添加任何东西,这样使得这个节点特殊或者不同.

你可以配置一个节点继承于另外一个节点,而另外一个节点可以继承于其它节点等,你不可以继承多个节点,(不能多重继承)因此,你不能这样像下面这样,示例:

```
node movable_server inherits gododgy_server, veryslow_server,
wreckspace_server {
# This won't work
}
```

还有更多...

你可以像正常的节点定义,你也可以指定一组节点名称同样继承于相同的定义.

```
node webserver1, webserver2, webserver3 inherits wreckspace_server {
...
}
```

或者使用正则表达式匹配多个服务器:

```
node /webserver\d+.veryslow.com/ inherits veryslow_server {
...
}
```

另请参阅:

使用类继承与覆盖

使用类继承与覆盖

正如节点可以从父节点继承一样,节点继承节省了你很多东西(时间和代码),类的继承的出于相似的想法.

例如,假设你有一个 apache 的类,是用来管理 Apache web 服务器的,你想建立一个新的 Apache 主机,但配置文件略有不同-也许是 Apache 的监听端口不一样.你可以复制整个 apache 类,除了配置文件,另外,你可以以 apache 配置文件不同,创建两个新的类,其中每个类都执行基本的 class 类和不同的版本的 apache 配置文件.一个更简洁的做法是从 Apache 类继承,但是覆盖其配置文件.

准备...

1.为新的 apache 模块创建目录结构.

```
# mkdir /etc/puppet/modules/apache
# mkdir /etc/puppet/modules/apache/manifests
```

2.创建/etc/puppet/modules/apache/manifests/init.pp 文件,其内容如下:

```
import "*"
```

3.创建 /etc/puppet/modules/apache/manifests/apache.pp 文件,其内容如下:

```
class apache {
package { "apache2-mpm-worker": ensure => installed }
service { "apache2":
enable => true,
ensure => running,
require => Package["apache2-mpm-worker"],
}
file { "/etc/apache2/ports.conf":
source => "puppet:///modules/apache/port80.conf.apache",
notify => Service["apache2"],
}
```

}

4.安装 Apache 软件包,如果还不存在,那么就复制 ports.conf 放到 puppet fileserver 里:
 # apt-get install apache2-mpm-worker
 # cp /etc/apache2/ports.conf /etc/puppet/modules/apache/files/port80.conf.apache
5. 在节点上执行 apache 类:
node cookbook {
include apache
}
6. 运行 puppet 验证代码是否工作正常.

怎么办呢...
1.创建一个新的版本 port80.conf.apache 重命名为 port8000.conf.apache
注意下面的改动:
NameVirtualHost *:8000
Listen 8000
2.现在添加一个文件,文件名为 /etc/puppet/modules/apache/manifests/port8000.pp,
内容如下:
class apache::port8000 inherits apache {
File["/etc/apache2/ports.conf"] {
source => "puppet:///modules/apache/port8000.conf.apache",
}
3.修改你的节点,执行 apache:port8000 这个类而不是 apache 这个类:
node cookbook {
include apache::port8000
}
4.运行 puppet,检查验证是否发生变化:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1302970905'
--- /etc/apache2/ports.conf 2010-11-18 14:16:23.000000000 -0700
+++ /tmp/puppet-file20110416-6165-pzeivi-0
10:21:47.204294334 -0600
2011-04-16
@@ -5,8 +5,8 @@
 # Debian etch). See /usr/share/doc/apache2.2-common/NEWS.Debian.
gz and
 # README.Debian.gz
-NameVirtualHost *:80
-Listen 80
+NameVirtualHost *:8000
+Listen 8000
<IfModule mod_ssl.c>

```
# If you add NameVirtualHost *:443 here, you will also have
to change
info: FileBucket adding /etc/apache2/ports.conf as {md5}38b31d203
26f3640a8dfbe1ff5d1c4ad
info: /Stage[main]/Apache/File[/etc/apache2/ports.conf]:
Filebucketed /etc/apache2/ports.conf to puppet with sum 38b31d2032
6f3640a8dfbe1ff5d1c4ad
notice: /Stage[main]/Apache/File[/etc/apache2/ports.conf]/
content: content changed '{md5}38b31d20326f3640a8dfbe1ff5d1c4ad'
to '{md5}41d9d446f779c55f13c5fe5a7477d943'
info: /Stage[main]/Apache/File[/etc/apache2/ports.conf]:
Scheduling refresh of Service[apache2]
notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refresh'
from 1 events
notice: Finished catalog run in 4.85 seconds
```

它是如何工作...

让我们再看看新的类:

```
class apache::port8000 inherits apache {
File["/etc/apache2/ports.conf"] {
source => "puppet:///modules/apache/port8000.conf.apache",
}
}
```

你可以看到类的名称后面是继承于 apache,这样的话精确复制 apache 类,除了后面的变化.

这一行:

```
   File["/etc/apache2/ports.conf"] {
```

上面的意味着我们要覆盖父类的 source 参数值.如果我们复制整个 apache 类的定义,并改变源的值,那么结果将是完全一样的:

```
class apache {
package { "apache2-mpm-worker": ensure => installed }
service { "apache2":
enable => true,
ensure => running,
require => Package["apache2-mpm-worker"],
}
file { "/etc/ap
ache2/ports.conf":
**source => "puppet:///modules/apache/port8000.conf.apache",**
notify => Service["apache2"],
}
```

```
}
```

还有更多...

首先覆盖继承的类有些复杂,虽然实现起来简单,一旦你有想法,但是这个是一个创造性的方法,它删除了大量的重复,仅保留了不同的部分,可以使你的代码表现可具有可读性,这里有一些方法使用类的覆盖.

未定义参数

有时候你不想改变参数值,你只是想干脆删除它,要做到这一点,你可以使用值为 undef:

```
class apache::norestart inherits apache {
File["/etc/apache2/ports.conf"] {
notify => undef,
}
}
```

使用+>添加额外的值

同样,你可以取替已经设置好的值,你可能要添加更多的值在父类的定义中,+> 符号可以做到:

```
class apache::ssl inherits apache {
file { "/etc/ssl/certs/cookbook.pem":
source => "puppet:///modules/apache/cookbook.pem",
}
Service["apache2"] {
require +> File["/etc/ssl/certs/cookbook.pem"],
}
}
```

+>操作符在父类的定义中添加了个值(或者中概括([])中的数组值),
在上述情况下,这个最终结果相当于这样的:

```
service { "apache2":
enable => true,
ensure => running,
require => [ Package["apache2-mpm-worker"], File["/etc/ssl/certs/cookbook.pem"] ],
}
```

禁用资源

继承和覆盖最常见的用法是禁止服务或者其它资源:

```
class apache::disabled inherits apache {
Service["apache2"] {
enable => false,
```

ensure => stopped,
}
}

另请参阅:
使用节点继承
类的参数传递
使用标准命名约定

类的参数传递

"It should be noted that no ethically-trained software engineer would ever consent to write a DestroyBaghdad procedure. Basic professional ethics would instead require him to write a DestroyCity procedure, to which Baghdad could be given as a parameter."
— Nathaniel S Borenstein

传递参数到类通常非常有用,例如:你可能需要管理不同版本的 gem 软件包,而不是为每个不同的版本号写个单独的类,使用继承和覆盖,你可以把版本号作为参数传递到类.

如何去做...
1.在类的定义中声明一部分参数:
class eventmachine($version) {
package { "eventmachine":
provider => gem,
ensure => $version,
}
}

2.然后在一个节点里按照下面的语法执行一个类.
class { "eventmachine": version => "0.12.8" }

它是如何工作的...

类的定义:
class eventmachine($version) {

就像普通的类一样,不同的就是在指定类的需要一个参数:$version.在类中,我们定义了下软件包资源:
package { "eventmachine":
provider => gem,

ensure => $version,
}

这是一个 gem 软件包,我们的需求是安装指定$version 的软件包,当你在一个节点上执行类,而不是常见的语法:
include eventmachine

这里才是类的语句:
class { "eventmachine": version => "0.12.8" }

这是有同样的效果,但也为$version 参数设置了一个值.

还有更多...

你可以在类中指定多个参数:
class mysql($package, $socket, $port) {

同样使用下面方法应用他们:
class { "mysql":
package => "percona-sql-server-5.0",
socket => "/var/run/mysqld/mysqld.sock",
port=> "3306",
}

你也可以给一些参数设置默认的值:

class mysql($package, $socket, $port = "3306") {
部分或者全部:
class mysql(
package = "percona-sql-server-5.0",
socket = "/var/run/mysqld/mysqld.sock",
port = "3306") {

不同于 define 函数,只有存在于节点上类的在实例初始化化参数,那么,如果你有几种不同的实例资源,使用 define 取替.

编写可重用的,跨平台的代码

"The pessimist complains about the wind; the optimist expects it to change; the realist adjusts the sails."

— William Arthur Ward

每个系统管理员都有一个梦想,相同的基础设施,所有机器上运器上运行相同的操作系统,且版本一致,然而,正如在生活的其它方面,现实往往和理想不符.你可以负责一堆使用年限不同的机器,以及不同架构的服务器,不同操作系统,运行的内核版本也不一样,往往期分散在不同的数据中心以及不同的互联网服务供应商(ISP).这种情况像 CS 在系统管理员心中使用"SSH for 循环"信念,因为在所有服务器执行相同的命令,而有些服务器是不一样的,那么结果不可预测,甚至结果非常危险.我们一定要努力把旧服务器的使用起来,并尽可能的让它们服务于单一的集群平台,使其管理更简单,更省钱,更可靠.但是,直到我们发现 puppet,其应对不同的环境会稍微容易一些.

怎么办呢...

1.如果你不同的数据中心服务器需要配置略有不同网络,例如,使用节点继承技术来封装不同之处.

```
node wreckspace_server inherits server {
include admin::wreckspace_specific
}
```

2.如果你需要应用适应于用不同的操作系统发生版本的代码到服务器,他们的主要区别可能是软件包的名称或者服务名,以及配置文件存放位置.尝试捕捉这些差异到单个类,可以使用 selectors 来设置全局变量:

```
$ssh_service = $operatingsystem? {
/Ubuntu|Debian/ => "ssh",
default => "sshd",
}
```

那样的话,你不用担心在 puppet 代码中的任何差异,当你提交些东西,你可以放心的使用变量,它会根据具体环境正确的指向到相应值.

```
service { $ssh_service:
ensure => running,
}
```

3.我们经常需要配合不同的架构,这可能会影响共享库的路径,也可能需要不同版本的软件包,同样尝试在一个单一的架构类中设置所需要的全局变量:

```
$libdir = $architecture ? {
x86_64 => "/usr/lib64",
default => "/usr/lib",
}
```

那么,你可以在任何情况下使用架构依赖值为必须的,只要在你的代码,或者说模板里这样做:

```
; php.ini
[PHP]
; Directory in which the loadable extensions (modules) reside.
extension_dir = <%= libdir %>/php/modules
```

它是如何工作的...

这种方法的优点是(可称为自上而下),你只需要提供选项供系统选择,替代的,自下而上的方法是,将随处可见设置 selector 或者 case 语句:

```
service { $operatingsystem? {
/Ubuntu|Debian/ => "ssh",
default => "sshd" }:
ensure => running,
}
```

这不仅导致了大量的重复,又使代码更难读懂,而当你需要添加下个新的操作系统项到组合中去,你需要在整个代码里修改,而不是仅仅需要修改一处.如果你正在写一个通用的模块(例如 Pupppet Forge),你尽可能的使它可以跨平台使用,尽力就可以了,在不同的分支,平台或者架构里测试它,并添加合适的值,使它工作的更好.如果你正在使用一个公共模块,并修改它适应自己的环境,如果你认为他们可能会帮助到其他人,你可以考虑更新您的版本.

即使你不想发布的你的模块,牢记它必须能在生产环境中稳定运行一段时间,以适许多操作系统变化.一开始设计就要有此初衷,那么你的生活会变得容易或者你可以结束维护你的代码.

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."
— Dave Carhart

另请参阅:

- 使用节点继承
- 使用类继承与覆盖
- 使用公共模块

获取环境变量信息

通常在 puppet 代码里,你需要知道本地的机器的一些有关信息.Facter 是 puppet 工具,它提供的一个标准方式去获得环境变量信息,只要你想.

操作系统
内存大小
架构
处理器个数

要看到你系统上一份完整的环境变量列表:运行:
#facter

在命令行的你可以很方便的取得信息,真正的威力是你可以在 puppet 代码里来访问这些变量.

怎么办呢...

1.像使用其它变量一样,参考 facter 在你代码里写上变量名:
notify { "This is $operatingsystem version
$operatingsystemrelease, on $architecture architecture, kernel
version $kernelversion": }

当运行 puppet,会自动为当前节点填补合适的值:
notice: This is Ubuntu version 10.04, on i386 architecture, kernel
version 2.6.32

它是如何工作...

Facter 为 puppet 提供了一个抽象层和提供了一个标准方式使用代码,来获得客户端的环境变量,当你在代码里引出一个 fact 变量,puppet 会通过 Facter 获得当前的值,并将其值插入到代码里.

还有更多...

你也可以在模板里使用 facts,例如:你可能需要把结点名插入到一个文件,或者选择一个基于内存大小的应用程序配置来设置节点.当你在模板里使用 names,记住,他们不需要前面加$附号,那是因为是在 Ruby 代码里,不是 Puppet:

$KLogPath <%= case kernelversion when "2.6.31" then "/var/run/
rsyslog/kmsg" else "/proc/kmsg" end %>

另请参阅
创建自定义 facts

使用 generate 导入动态信息

尽管系统管理员喜欢从自己的办公室里使用成堆的旧的打印机,他们有时需要与其它部门进行信息交流与沟通.你如:你可能需要在你的代码里插入数据来自于外部的源,generate 功能非常有用并且能做到这点.

准备

1. 在 puppetmaster 创建/usr/local/bin/latest-puppet.rb 脚本,内容如下:

```
#!/usr/bin/ruby
require 'open-uri'
page = open("http://www.puppetlabs.com/misc/download-options/").
read
print page.match(/stable version is ([\d\.]*)/)[1]
```

怎么办呢...

1. 在你的代码里添加如下内容:

```
$latestversion = generate("/usr/local/bin/latest-puppet.rb")
notify { "The latest stable Puppet version is ${latestversion}.
You're using ${puppetversion}.": }
```

2.运行 puppet:

```
# puppet agent --test
notice: The latest stable Puppet version is 2.6.7. You're using 2.6.4.
```

它是如何工作的...

generate 函数是在 puppetmaster 上(不是 puppet 客户端)来运行指定的脚本或者程序并返回结果,在这种情况下,将返回最新的 Puppet 的版本号.

我不建议你在生产环境中运行此脚本,但是 Puppet 实验定有一种习惯重新编排自己的网站,但你有此想法,任何脚本可以做到,打印,提取,计算,例如:你可以使用 generate 在代码里,在数据库里查询结果.

这是值得记住的,就像在模板里调用使用嵌入式 Ruby,generate 只能运行在 puppetmaster 上,而不是仅仅运行着 Puppet 服务的节点.我曾经犯过一个错误在模板里调用/bin/hostname,结果让我大吃一惊,我所有的 puppet 节点名称均为 puppet.

当你要获得指定节点的更多信息,最好是做个自定义的 fact.

还有更多...

如果你需要在使用 generate 传递参数去执行,可以把参数作为额外的函数参数调用:

```
$latestpuppet = generate("/usr/local/bin/latest-version.rb","puppet")
$latestmc = generate("/usr/local/bin/latest-version.rb","mcollective")
```

Puppet 将会设法保护限制性的字符,禁止你使用 generate 恶意的 SEHLL 调用因此,shell 的管道符是禁止的,例如:最简单和最安全的事情的是把你逻辑写成一个脚本里,然后调用此脚本.

从 CSV 导入数据

当你需要在表中查找某些值,你可以使用冗长的 case 语句或者 selectors.但是一个整洁的方法是使用 extlookup 功能.在 puppetmaster 上可以查询外部的 CSV 文件,并返回匹配的数据.

合并所有的单一文件组合成一个大文件和移动 puppetmaster 上的文件,这样使 Puppetmaster 更易于维护,以及容易与其他人共享,一个开发团队只要知道他们的应用就可以管理,例如:通过部署部分的 release 的合适的 CSV 文件,puppet 只需要知道在哪里可以找到文件,extlookup 将会做这些.

准备

1.创建 /var/www/apps/common.csv 文件,内容如下:

```
path,/var/www/apps/%{name}
railsversion,3
domain,www.%{name}.com
```

2.创建/var/www/apps/myapp.csv 文件,内容如下:

```
railsversion,2
```

怎么办呢...

1.添加以下内容到你的代码:

```
$extlookup_datadir = "/var/www/apps/"
$extlookup_precedence = [ "%{name}", "common" ]
class app( $name ) {
$railsversion = extlookup("railsversion")
$path = extlookup("path")
$domain = extlookup("domain")
notify { "App data: Path ${path}, Rails version
${railsversion}, domain ${domain}": }
}
class { "app": name => "myapp" }
```

2.运行 puppet

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303129760'
notice: App data: Path /var/www/apps/myapp, Rails version 2,
domain www.myapp.com
notice: /Stage[main]/App/Notify[App data: Path /var/www/apps/
myapp, Rails version 2, domain www.myapp.com]/message: defined
```

'message' as 'App data: Path /var/www/apps/myapp, Rails version 2,
domain www.myapp.com'
notice: Finished catalog run in 0.58 seconds

它是如何工作的...

1.我们做的第一件事情就是定义$extlookup_datadir 变量,它告诉 extlookup 在什么目录下寻找数据文件.你通常会在 site.pp 里设置或者在任何其它地方定义全局变量.
$extlookup_datadir = "/var/www/apps/"

2.然后我们告诉 extlookup 哪里寻找数据文件,以及寻找的优先顺序:
$extlookup_precedence = ["%{name}", "common"]

这也可以是任意长度的数组,当我们使用 extlookup 查询时,Puppet 会尝试扫描整个文件直到查询到请求的值.文件名可以包含变量,在这个例子中,我们已经使用了%{name},因此我们期望能找到我们设置的$name.当我们调用 extlookup,Puppet 将会使用其值作为第一个文件名.

1.下一步,在 app 类里面,你们调用 extlookup 去得到一个变量值:
$railsversion = extlookup("railsversion")

现在看起来 extlookup 是机械的在 CSV 文件里读取数据.它会在$extlookup_datadir 目录(在本例中/var/www/apps)下寻找文件%{name}.csv(在本例中为 myqpp.csv).因此它从/var/www/apps/myapp.csv 文件查找包含 railsversion,2 匹配行.

我们已经找到所需要的值(2),因此 extlookup 返回它的值.

1.接下来的 extlookup 调用不是那么的幸运.
$path = extlookup("path")
再次,extlookup 首先在 myapp.csv 文件.但是它找不到匹配 path 的变量值,因此 extlookup 会优先移向$extlookup_precedence 文件去寻找,$extlooup_precedence 是 common.csv:

path,/var/www/apps/%{name}
railsversion,3
domain,www.%{name}.com

幸运的是,这样匹配了,因此 Puppet 会返回值/var/www/apps/%{name},在本例中值为/var/www/apps/myapp.

你可以看到,这样我们可以在 common.csv 文件里设置一个默认值,common.csv 文件里可以为每个应用程序选择覆盖其自己的 myapp.csv 文件.extlookup 将遍历$extlookup_precedence 文件直到查询到要求查询的值.由于 myapp.csv 在前面,其
的任何设置将优先覆盖 common.csv 里的设置.

还有更多...

你还可以在 extlookup 调用的时候使用默认值,如果在 CSV 文件里没有找到合适的值.
$path = extlookup("path", "/var/www/misc")

你也可以指定首先要查找 csv 文件,使用$extlookup_precedence 定义.
$path = extlookup("path", "/var/www/misc", "paths")

这样会在 paths.csv 文件里查找数据,如果没有找到,将会在$extlookup_precedence
如平常一样查找数据.

你的 CSV 文件中的值也可以引出变量,正像我们所做的那样:
domain,www.%{name}.com

你可以在整个过程中使用任意变量 ,包括 Facter 探测出来的变量.
domain,%{fqdn}

R.I. Pienaar 的文章"Complex data and Puppet" 是对 extlookup 一个很好的介绍.
地址为:http://www.devco.net/archives/2009/08/31/complex_data_and_puppet.php

Jordan Sissel 曾经写过关于使用 extlookup 配置你的整个基础设施:
http://sysadvent.blogspot.com/2010/12/day-12-scaling-operability-with-truth.html

另请参阅:

导入动态信息
创建自定义 facter 变量.

传递参数到 shell 命令

如果你需要在命令行下插入一个值,他们通常需要被引用,尤其是如果值里包含空格.shellquote 功能能将任意的参数,包括数组和把每个参数用双引号引起来,并返回以空格分隔的字符串,shellquote 可以传递参数到命令行.

在这个例子中,我们想建立一个 exec 资源,exec 资源是用来重命名一个文件,但两者源和目标的名称都包含空格,因此,他们需要在命令行下正确的引用.

怎么办呢...
1. 添加下面内容到你的代码:

```
$source = "Hello Jerry"
$target = "Hello... Newman"
$argstring = shellquote( $source, $target )
$command = "/bin/mv ${argstring}"
notify { $command: }
```

2. 运行 puppet:

```
notice: /bin/mv "Hello Jerry" "Hello... Newman"
```

它是如何工作的...

1. 首先我们定义了$source 和$target 两个变量,这两个变量是我们要在命令行使用的文件.

```
$source = "Hello Jerry"
$target = "Hello... Newman"
```

2.然后,我们调用 shellquote,并以 shellquote 为分隔符,来连接这些变量.如下:

```
$argstring = shellquote( $source, $target )
```

3.接下来我们把它们组合在一起,成为了最终的命令行:

```
$command = "/bin/mv ${argstring}"
```

4.其结果是:

```
/bin/mv "Hello Jerry" "Hello... Newman"
```

5.现在可以在命令下行运行 exec 资源.

如果我们不使用 shellquote 会发生什么?

```
$source = "Hello Jerry"
$target = "Hello... Newman"
$command = "/bin/mv ${source} ${target}"
notify { $command: }
notice: /bin/mv Hello Jerry Hello... Newman
```

这将无法工作,那是因为 mv 命令是要以空格分隔字符串参数,所以将其解释为,这要求三个文件 Hello,Jerry,和 Hello...直到目录名为 Newman,这可不是我们所希望看到的.

使用文件和软件包

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization."

— Gerald Weinberg

在本章中我们将学习包括以下内容:

- 快速编辑配置文件
- 使用 Augeas 自动编辑配置文件
- 使用依赖关系
- 使用 snippets 构建配置文件
- 使用 ERB 模板
- 在模板中使用循环数组
- 从第三方仓库安装软件包
- 建立 APT 软件包源
- 建立 gem 源
- 在源文件自动构建软件包
- 比较软件包版本

快速编辑配置文件

你知道吗 Pupppet 可以做洞眼手术?通常我们不想把整个配置文件交给 Puppet 管理,只希望添加些设置,特别指出的是该文件由别人管理,自己又不能覆盖它.一个简单的方法是添加该行如果该行不存在于原配置文件中,那是多么的棒.

你可以使用 exec 资源去完成那样的工作,这个例子,来源于 Puppet 实验室的维基.演示了如何使用 exec 资源在文本文件里追加行.

怎么办呢...

1.创建/etc/puppet/manifests/utils.pp 文件,内容如下:

```
define append_if_no_such_line($file, $line) {
exec { "/bin/echo '$line' >> '$file'":
unless => "/bin/grep -Fx '$line' '$file'"
}
}
```

2.在/etc/puppet/manifests/site.pp 添加这一行:

```
import "utils.pp"
```

3.现在添加这些代码:

```
append_if_no_such_line { "enable-ip-conntrack":
file => "/etc/modules",
line => "ip_conntrack",
}
```

4.运行 puppet:

```
# puppet agent --test
```

```
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303649606'
notice: /Stage[main]//Node[cookbook]/Append_if_no_such_
line[enable-ip-conntrack]/Exec[/bin/echo 'ip_conntrack' >> '/etc/
modules']/returns: executed successfully
notice: Finished catalog run in 1.22 seconds
```

它是如何工作的...

exec 资源会追加指定的文本$line 到指定的文件$file:

```
exec { "/bin/echo '$line' >> '$file'":
```

除非它已经存在该行:

```
unless => "/bin/grep -Fx '$line' '$file'"
```

append_if_no_such_line 资源现在可以在你的代码里使用.在本例中,我们已经使用它确保/etc/modules 文件里有包含这一行:(指定内核模块在开机时加载)ip_conntrack

还有更多...

你可以使用类似的定义在文本文件上执行其它的小改动.例如:这将让你在文件里使用搜索并替换匹配字符串:

```
define replace_matching_line( $match, $replace ) {
exec { "/usr/bin/ruby -i -p -e 'sub(%r{$match}, \"$replace\")'$name":
onlyif => "/bin/grep -E '$match' $name",
logoutput => on_failure,
}
}
replace_matching_line { "/etc/apache2/apache2.conf":
match => "LogLevel .*",
replace => "LogLevel debug",
}
```

另请参阅

使用 Augeas 自动修改配置文件

使用 Augeas 自动修改配置文件

当然,有这么多关于标准,标准是件伟大的事情,有时每个应用程序看起来都有自己巧妙不同的文件格式,并且写正则表达式去解析和修改文件,那是件烦人的业务.幸运的是,Augeas 在这里可以帮助我们.Augeas 是个工具,它旨在简化使用不同的配置文件格式,augeas 是通过树形

结构来配置修改相应的配置文件的.augeas 是 puppet 的标准资源。它可以使所需的配置变化更加智能和自动化.可以用来管理配置文件。但这种资源的使用需要有 augeas 及 ruby-augeas 的支持.

准备...

1.创建 /etc/puppet/modules/admin/manifests/augeas.pp 文件,内容如下:

```
class admin::augeas {
package { [ "augeas-lenses",
"augeas-tools",
"libaugeas0",
"libaugeas-ruby1.8" ]:
ensure => "present"
}
}
```

2. 在节点上执行类:

```
node cookbook {
include admin::augeas
}
```

3.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303657095'
notice: /Stage[main]/Admin::Augeas/Package[augeas-tools]/ensure:
ensure changed 'purged' to 'present'
notice: Finished catalog run in 21.96 seconds
```

怎么办呢...

1.创建/etc/puppet/modules/admin/manifests/ipforward.pp 文件,内容如下:

```
class admin::ipforward {
augeas { "enable-ip-forwarding":
context => "/files/etc/sysctl.conf",
changes => [
"set net.ipv4.ip_forward 1",
],
}
}
```

2. 在某个节点上执行这个类:

```
node cookbook {
```

include admin::augeas
include admin::ipforward
}

3.运行 Pupppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303729376'
notice: /Stage[main]/Admin::Ipforward/Augeas[enable-ip-forwarding]/returns: executed successfully
notice: Finished catalog run in 3.53 seconds

4.检查设置的值是否正确的已经应用:
 # sysctl -p |grep forward
net.ipv4.ip_forward = 1

它是如何工作...
1.我们声明了一个名为 augeas 资源名为 enable-ip-forwarding:
augeas { "enable-ip-forwarding":
2.我们指定了我们要在/etc/sysctl.conf 文件里要改变的内容:
context => "/files/etc/sysctl.conf",

3.传递一个数组参数,也就是我们要改变的值,(在本例中只有一个):
changes => [
"set net.ipv4.ip_forward 1",
],

通常 Augeas 采取这样的格式:
set <parameter> <value>

Augeas 使用了传输文件名为 lenses, 以便它能够在给定的配置文件中使用适当的格式应用这些设置.在本例中,该设置将被卧翻译成这样一行在/etc/sysctl.conf 中:
net.ipv4.ip_forward=1

还有更多...

我选择 /etc/sysctl.conf 作为例子,是因为它可以包含各种各样的内核设置,你可能想更改这些设置各种不同目的在不同的 Puppet 类中.你可能需要启用 IP 转发,如本例,为一个路由器的类.但你可能还需要为负载均衡类调整 net.core.somaxconn 的值.

这要意味着,Puppet 简单的修改 /etc/sysctl.conf 文件和分发它并不能工作,因为你有可能有版本冲突,这取决于你想要修改的设置依赖.Augeas 是正确的解决方案,因为你可以定义在不同的地方修改同一个文件,他们将不会发生冲突.Augeas 是个强大的工具,它与大多数标准的

Linux 配置文件兼容和你可以自己写属于或者专有配置格式.如果你要管理这些.使用 Puppet 和 Augeas 更多的信息,可以查看 puppet 实验室上的维基页面
http://projects.puppetlabs.com/projects/1/wiki/Puppet_Augeas

使用 snippets 来构建配置文件

你怎么一口吃掉一头大象?,有时候你有个情况,你要建立一个单一的配置文件从不同类别管理的的类中.例如:你可能有两个或者三个服务需要配置 rsync 模块.因此你不可能分配一个单的 rsyncd.conf.虽然你可以使用 Augeas,有一个简单的方法使用 exec 连接成一个单一文件.

怎么办呢...
1.创建/etc/puppet/modules/admin/manifests/rsyncdconf.pp 文件,内容如下:

```
class admin::rsyncdconf {
file { "/etc/rsyncd.d":
ensure => directory,
}
exec { "update-rsyncd.conf":
command    => "/bin/cat /etc/rsyncd.d/*.conf > /etc/rsyncd.conf",
refreshonly => true,
}
}
```

2.添加下面内容到你的代码:

```
class myapp::rsync {
include admin::rsyncdconf
file { "/etc/rsyncd.d/myapp.conf":
ensure => present,
source => "puppet:///modules/myapp/myapp.rsync",
require => File["/etc/rsyncd.d"],
notify => Exec["update-rsyncd.conf"],
}
}
include myapp::rsync
```

3.创建/etc/puppet/modules/myapp/files/myapp.rsync 文件,内容如下:

```
[myapp]
uid = myappuser
gid = myappuser
path = /opt/myapp/shared/data
comment = Data for myapp
list = no
```

read only = no
auth users = myappuser

4.运行 Puppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303731804'
notice: /Stage[main]/Admin::Rsyncdconf/File[/etc/rsyncd.d]/ensure:
created
notice: /Stage[main]/Myapp::Rsync/File[/etc/rsyncd.d/myapp.conf]/
ensure: defined content as '{md5}e1e57cf38bb88a7b4f2fd6eb1ea2823a'
info: /Stage[main]/Myapp::Rsync/File[/etc/rsyncd.d/myapp.conf]:
Scheduling refresh of Exec[update-rsyncd.conf]
notice: /Stage[main]/Admin::Rsyncdconf/Exec[update-rsyncd.conf]:
Triggered 'refresh' from 1 events
notice: Finished catalog run in 1.01 seconds

它是如何工作的...

admin::rsyncdconf 类创建一个存放 rsync 配置文件目录:
file { "/etc/rsyncd.d":
ensure => directory,
}

当你创建一个配置片断(snippet)(如例子中的 myapp::rsync),你所需要做的就是创建所需要的目录:
require => File["/etc/rsyncd.d"],

并通知 exec 更新主要配置文件:
notify => Exec["update-rsyncd.conf"],

这样只要配置片断更新有,exec 就会执行.
exec { "update-rsyncd.conf":
command => "/bin/cat /etc/rsyncd.d/*.conf > /etc/rsyncd.conf",
refreshonly => true,
}

将合并在/etc/rsyncd.d 下面的所有片断到 rsyncd.conf

还有更多...

当你有个像 rsync 服务,需要一个单独的配置文件,那是非常有用的模式,实际上,它可以让你的 Apache 的 conf.d 的功能或者 php-ini.d 目录.

另请参阅

使用 tags

使用模 ERB 模板

模板是一个非常智能的文本文件.你使用 Puppet 可以部署一个文本文件到任何地方,你可以使用模板来代替.在最简单的情况下,一个模板可以是一个静态文件.更有益的是,你可以使用 ERB(embedded Ruby)语法插入变量.

例如:

<%= name %>, this is a very large drink.

如果模板使用变量$name 的值为 Zaphod Beeblebrox,那么模板会计算为:

Zaphod Beeblebrox, this is a very large drink.

在不同的变量一个或者两个变量是非常有用,这是个简单的技术生成了大量的文件.例如:虚拟主机,在本例子中,我们将使用 ERB 模板来生成一个 Apache 虚拟主机定义.

准备..

1.如果你没有 apache 模块,请创建一个.

```
# mkdir /etc/puppet/modules/apache
# mkdir /etc/puppet/modules/apache/templates
# mkdir /etc/puppet/modules/apache/manifests
```

2.创建/etc/puppet/modules/apache/manifests/apache.pp 文件,内容如下:

```
class apache {
package { "apache2-mpm-worker": ensure => installed }
service { "apache2":
enable => true,
ensure => running,
require => Package["apache2-mpm-worker"],
}
}
```

怎么办呢...

1.创建/etc/puppet/modules/apache/manifests/site.pp 文件,内容如下:

```
class apache::site( $sitedomain ) {
include apache
file { "/etc/apache2/sites-available/${sitedomain}.conf":
```

```
content => template("apache/vhost.erb"),
require => Package["apache2-mpm-worker"],
}

file { "/etc/apache2/sites-enabled/${sitedomain}.conf":
ensure => symlink,
target => "/etc/apache2/sites-available/${sitedomain}.conf",
require => Package["apache2-mpm-worker"],
notify => Service["apache2"],
}
}
```

2.创建/etc/puppet/modules/apache/templates/vhost.erb 文件,内容如下:

```
<VirtualHost *:80>
ServerName <%= sitedomain %>
ServerAdmin admin@<%= sitedomain %>
DocumentRoot /var/www/<%= sitedomain %>
ErrorLog logs/<%= sitedomain %>-error_log
CustomLog logs/<%= sitedomain %>-access_log common
<Directory /var/www/<%= sitedomain %>>
Allow from all
Options +Includes +Indexes +FollowSymLinks
AllowOverride all
</Directory>
</VirtualHost>
<VirtualHost *:80>
ServerName www.<%= sitedomain %>
Redirect 301 / http://<%= sitedomain %>/
</VirtualHost>
```

3.添加上面到节点:

```
class { "apache::site": sitedomain => "keithlard.com" }
```

4.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304761207'
notice: /Stage[main]/Apache::Site/File[/etc/apache2/sites-
available/keithlard.com.conf]/ensure: defined content as '{md5}f2a
558c02beeaed4beb7da250821b663'
notice: /Stage[main]/Apache::Site/File[/etc/apache2/sites-enabled/
keithlard.com.conf]/ensure: created
```

info: /Stage[main]/Apache::Site/File[/etc/apache2/sites-enabled/
keithlard.com.conf]: Scheduling refresh of Service[apache2]
notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refresh'
from 1 events
notice: Finished catalog run in 8.06 seconds

它是如何工作的...

apache::site 类需要带个参数,参数名为 sitedomain,sitedomain 是你的虚拟主机,所要创建的域名:在本例中,keithlard.com.

然后我们使用 vhost.er 模板去生成一个合适的定义的 Apache 虚拟主机.无论在什么时候在模板里引用$sitedomain,例如:
ServerName <%= sitedomain %>

Puppet 会将取代替为相应的值:
ServerName keithlard.com

模板系统的优点是,你可以创建多个站点,都使用相同的虚拟主机模板.
class { "apache::site":sitedomain => "bitfieldconsulting.com" }
class { "apache::site": sitedomain => "cribbagecorner.com" }
class { "apache::site":sitedomain => "cornishceremonies.com" }
class { "apache::site":sitedomain => "mosquito-mojito.com" }

如果你想对所有网站做个小的配置文件修改(例子如,修改管理员电子邮件地址),你可以在模板里一次完成,Puppet 会更新所有相应的虚拟主机.

还有更多...
你上面例子中,我们在模板里仅仅使用了一个变量,但然你可以定义放多变量.这些也可以是 facts(factre 探测出来的变量):
ServerName <%= fqdn %>

或者是 Ruby 正则表达式:
MAILTO=<%= emails.join(',') %>

或者是你想写的什何 Ruby 代码:
ServerAdmin <%= sitedomain == 'coldcomfort.com' ? 'seth@coldcomfort.com' :
'flora@poste.com' %>

另请参阅
在模板里使用循环数组

在模板里使用循环数组

在前面的例子子中,我们可以看到使用 Ruby 能在模板里插入不同的值,这些 Ruby 结果是依赖于 Ruby 表达式的值.你还可以使用一个循环来生成,例如:数组中的元素:

怎么办呢...

1.添加以下内容到你的代码:

```
$ipaddresses = [ '192.168.0.1',
'158.43.128.1',
'10.0.75.207' ]
file { "/tmp/addresslist.txt":
content => template("admin/addresslist.erb")
}
```

2.创建/etc/puppet/modules/admin/templates/addresslist.erb 文件,内容如下:

```
<% ipaddresses.each do |ip| -%>
IP address <%= ip %> is present.
<% end -%>
```

3.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304766335'
notice: /Stage[main]//Node[cookbook]/File[/tmp/addresslist.txt]/
ensure: defined content as '{md5}7ad1264ebdae101bb5ea0afef474b3ed'
notice: Finished catalog run in 0.64 seconds
```

4.检查生成的文件内容:

```
# cat /tmp/addresslist.txt
IP address 192.168.0.1 is present.
IP address 158.43.128.1 is present.
IP address 10.0.75.207 is present.
```

它是如何工作的...

1.在模板的第一行,我们引用了一个 ipaddressess 数组,并调用 each 方法:

```
<% ipaddresses.each do |ip| -%>
```

2.在 Ruby 中,创建一个循环数组为每个元素执行一次,每次循环时,变量 ip 会被设置为当前元素的值.

3.在我们的例子中,ipaddresses 数组有三个元素,因此下面的行会被执行三次,一次每个元素:

```
<% ipaddresses.each do |ip| -%>
```

4.结果是输出如下三行:

IP address 192.168.0.1 is present.
IP address 158.43.128.1 is present.
IP address 10.0.75.207 is present.
5.最后一行结束循环.
<% end -%>
6.请注意第一和最后一行的 -%>符号,而不是我们以前所看到的%>,-的效果是阻止换行,否则的话将产生换行符,即在我们的文件里出现无用的空白行.

还有更多...

模板也可以循环散列或者数组或者哈希.
$interfaces = [{ name => 'eth0',
ip =>'192.168.0.1' },
{ name => 'eth1',
ip =>'158.43.128.1' },
{ name => 'eth2',
ip => '10.0.75.207' }]

<% interfaces.each do |interface| -%>
Interface <%= interface['name'] %> has the address <%= interface['ip'] %>.
<% end -%>
Interface eth0 has the address 192.168.0.1.
Interface eth1 has the address 158.43.128.1.
Interface eth2 has the address 10.0.75.207.

另请参阅:
使用 ERB 模板

从第三方仓库安装软件包

大多数情况下,你从主要发行版本仓库安装软件包,因些一个简单的 package 资源可以做到.
package { "exim4": ensure => installed }

但是,有时你需要的软件包只能在第三方仓库里才能找到(例如,Ubuntu 的 PPA),或者是由第三方提供的比主流发生版本较新版本的软件包.如果手动管理机器,你通常需要在安装软件包之前先添加仓库源到/etc/apt/sources.list.d(如果有必要,还要导入仓库的密钥).我们可以使用 Puppet 来自动完成这些过程.

怎么办呢...
1.添加以下内容到你的代码:

```
package { "python-software-properties": ensure => installed }
exec { "/usr/bin/add-apt-repository ppa:mathiaz/puppet-backports":
creates => "/etc/apt/sources.list.d/mathiaz-puppet-backports-lucid.list",
require => Package["python-software-properties"],
}
```

2.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304773240'
notice: /Stage[main]//Node[cookbook]/Exec[/usr/bin/add-apt-repository
ppa:mathiaz/puppet-backports]/returns: executed
successfully
notice: Finished catalog run in 5.97 seconds
```

它是如何工作的...

1.python-software-properties 提供了 add-apt-repository 命令,add-apt-repository 简化了添加额外的软件源的过程:

```
package { "python-software-properties": ensure => installed }
```

2.然后调用 exec 命令来添加所需要的配置:

```
exec { "/usr/bin/add-apt-repository ppa:mathiaz/puppet-backports":
```

3.因此,exec 不是每次都随 Puppet 运行,我们指定了命令生成的文件,因此,如果文件已经生成,那么 exec 将会跳过,并不执行 exec 资源.

```
creates => "/etc/apt/sources.list.d/mathiaz-puppet-backports-lucid.list",
```

你可能想在/etc/apt/sources.list.d 目录下,比较删除不必要的仓库源,像之前章节介绍的递归 file 资源.

另请参阅:

使用递归文件资源创建目录树

建立 Apt 软件仓库源

"We will control the horizontal. We will control the vertical."
- The Outer Limits

运行自己的软件包仓库有几个优点.你可以自己发布软件.你可以控制版本的上游或者把第三方软件包的版本放进仓库.你可以很快的找到你的服务器,可以避免因为网络慢或者不可靠的镜像站点去下载安装软件包.

即使你不需要创建自己的软件包,你可能需要下载所需的关键依赖包的版本,并保存在自己的仓库里,从而防止上游任何改变而引出的意外.(例如,你的发行版本镜像源到期而关闭).所以它很方指定 ensure=>laster 来更新到最新版本.

这也使得使用 Puppet 很容易自动更新这些软件包,你可能偶尔需要更新一个软件包(例如,当安全更新可用的时候),但是,如果你不控制仓库,如果你不想升级就面对危险.不升级会给你的系统带来麻烦.
自己建仓库两全其美,你可以使用 Puppet 自动更新软件包,但是,前提软件是要从仓库源更新,当新版本可用时,只需要复制到源去.你可以在软件包安装到线上环境前进行测试,然后再放到生产线上的源.

准备...

你需要'使用 ERB 模板'的章节中的 apache 模块,如果不存在请自己创建.在这个例子中,我的软件源名称为 packages.bitfieldconsulting.com,那是因为我想这样叫,当然你可以取一个不同名,那样的话你要替换例子中的整个源的名称.

怎么办呢...

```
# mkdir /etc/puppet/modules/repo
# mkdir /etc/puppet/modules/repo/manifests
# mkdir /etc/puppet/modules/repo/files
```

2.创建/etc/puppet/modules/repo/manifests/init.pp 文件,内容如下:

```
import "*"
```

3.创建 /etc/puppet/modules/repo/manifests/bitfield-server.pp 文件,内容如下:

```
class repo::bitfield-server {
include apache
package { "reprepro": ensure => installed }
file { [ "/var/apt",
"/var/apt/conf" ]:
ensure => directory,
}
file { "/var/apt/conf/distributions":
source => "puppet:///modules/repo/distributions",
require => File["/var/apt/conf"],
}
file { "/etc/apache2/sites-available/apt-repo":
source => "puppet:///modules/repo/apt-repo.conf",
require => Package["apache2-mpm-worker"],
}
file { "/etc/apache2/sites-enabled/apt-repo":
ensure => symlink,
target => "/etc/apache2/sites-available/apt-repo",
```

require => File["/etc/apache2/sites-available/apt-repo"],
notify => Service["apache2"],
}
}

4.创建/etc/puppet/modules/repo/files/distributions 文件,内容如下:

Origin: Bitfield Consulting
Label: bitfield
Suite: stable
Codename: lucid
Architectures: amd64 i386
Components: main non-free contrib
Description: Custom and cached packages for Bitfield Consulting
45.Create the file /etc/puppet/modules/repo/files/apt-repo.conf
with the following contents:
<VirtualHost *:80>
DocumentRoot /var/apt
ServerName packages.bitfieldconsulting.com
ErrorLog /var/log/apache2/packages.bitfieldconsulting.com.
error.log
LogLevel warn
CustomLog /var/log/apache2/packages.bitfieldconsulting.com.
access.log combined
ServerSignature On
 # Allow directory listings so that people can browse the
repository from their browser too
<Directory "/var/apt">
Options Indexes FollowSymLinks MultiViews
DirectoryIndex index.html
AllowOverride Options
Order allow,deny
allow from all
</Directory>
 # Hide the conf/ directory for all repositories
<Directory "/var/apt/conf">
Order allow,deny
Deny from all
Satisfy all
</Directory>
 # Hide the db/ directory for all repositories
<Directory "/var/apt/db">
Order allow,deny
Deny from all
Satisfy all

</Directory>
</VirtualHost>

5. 在某个节点上添加下面的代码:
include repo::bitfield-server
6. 运行 Puppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304775601'
notice: /Stage[main]/Repo::Bitfield-server/File[/var/apt]/ensure:
created
notice: /Stage[main]/Repo::Bitfield-server/File[/var/apt/conf]/
ensure: created
notice: /Stage[main]/Repo::Bitfield-server/File[/var/apt/conf/
distributions]/ensure: defined content as '{md5}65dc791b876f53318a
35fcc42c770283'
notice: /Stage[main]/Repo::Bitfield-server/Package[reprepro]/
ensure: created
notice: /Stage[main]/Repo::Bitfield-server/File[/etc/apache2/
sites-enabled/apt-repo]/ensure: created
notice: /Stage[main]/Repo::Bitfield-server/File[/etc/apache2/
sites-available/apt-repo]/ensure: defined content as '{md5}2da4686
957e5acf49220047fe6f6e6e1'
info: /Stage[main]/Repo::Bitfield-server/File[/etc/apache2/sites-enabled/apt-repo]:
Scheduling refresh of Service[apache2]
notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refresh' from 1 events
notice: Finished catalog run in 16.32 seconds

它是如何工作的...

其实,你并不很需要 APT 仓库,它可以通过 HTPP 下载,所以你刚才需要一个虚拟主机,你可以把实际的 rpm 文件存放在你喜欢的位置,只要一个 conf/distributions 文件,这是提供仓库的 APT 信息.
1.bitfield-server 类的第一部分,是确保我们已经安装配置好 Apache:
class repo::bitfield-server {
include apache
2.reprepro 工具用于管理仓库本身,非常有用(例如,添加一个新的软件包):
package { "reprepro": ensure => installed }

3.我们创建一个仓库的根目录/var/apt,平时还有 conf/distributions 文件:
file { ["/var/apt",
"/var/apt/conf"]:
ensure => directory,

}
file { "/var/apt/conf/distributions":
source => "puppet:///modules/repo/distributions",
require => File["/var/apt/conf"],
}

4.类的其它部分是发布一个虚拟主机,并使它能够响应
packages.bitfieldconsulting.com 请求:
file { "/etc/apache2/sites-available/apt-repo":
source => "puppet:///modules/repo/apt-repo.conf",
require => Package["apache2-mpm-worker"],
}
file { "/etc/apache2/sites-enabled/apt-repo":
ensure => symlink,
target => "/etc/apache2/sites-available/apt-repo",
require => File["/etc/apache2/sites-available/apt-repo"],
notify => Service["apache2"],
}

还有更多
当然,一个好的仓库不能没有软件包,在本节中,我们将看到如何添加包,以及如何配置服务器从仓库下载软件包.

添加软件包

要添加一上软件包到仓库,先下载它然后使用 reprepro 添加它到仓库:
 # cd /tmp
 # wget
http://archive.ubuntu.com/ubuntu/pool/main/n/ntp/ntp_4.2.4p8+dfsg-1ubuntu2.1_i386.deb
 # cd /var/apt
 # reprepro includedeb lucid /tmp/ntp_4.2.4p8+dfsg-1ubuntu2.1_i386.deb
Exporting indices...

配置节点使用仓库
1.创建/etc/puppet/modules/repo/manifests/bitfield.pp 文件,内容如下(请根据实际替换 IP 为地址为你的仓库主机所在的 IP 地址):

class repo::bitfield {
host { "packages.bitfieldconsulting.com":
ip=> "10.0.2.15",
ensure => present,
target => "/etc/hosts",
}
file { "/etc/apt/sources.list.d/bitfield.list":

```
content => "deb http://packages.bitfieldconsulting.com/lucid main\n",
require => Host["packages.bitfieldconsulting.com"],
notify => Exec["bitfield-update"],
}
exec { "bitfield-update":
command => "/usr/bin/apt-get update",
require=> File["/etc/apt/sources.list.d/bitfield.list"],
refreshonly => true,
}
}
```

如果你有 DNS 服务器或者你可以控制你的 DNS 区域,你可以跳过主机条目.

2.应用这个类到节点:

```
node cookbook {
include repo::bitfield
}
```

3.测试 NTP 软件包是否能从仓库中的找到:

```
# apt-cache madison ntp
ntp | 1:4.2.4p8+dfsg-1ubuntu2.1 | http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main
Packages
ntp | 1:4.2.4p8+dfsg-1ubuntu2.1 | http://packages.bitfieldconsulting.com/ lucid/main ackages
ntp | 1:4.2.4p8+dfsg-1ubuntu2 | http://us.archive.ubuntu.com/ubuntu/ lucid/main Packages
ntp | 1:4.2.4p8+dfsg-1ubuntu2 | http://us.archive.ubuntu.com/ubuntu/ lucid/main Sources
ntp | 1:4.2.4p8+dfsg-1ubuntu2.1 | http://us.archive.ubuntu.com/ubuntu/ lucid-updates/main
Sources
```

3.签名你的软件包

供生产线使用,你应该给仓库和软件包设置 GPG 密钥;更多更关如何使用 GPG 密钥,
可以看 Sander Marechal 的文章,主题为建立和管理 APT 源非常有用.链接地址为
http://www.jejik.com/articles/2006/09/setting_up_and_managing_an_apt_repository_with_reprepro/

建立一个 gem 源

每个系统管理员都有一个梦想,没有不兼容的软件包和系统,如果你管理 Ruby 或者 Rails 应用程序,你需要使用 Rubygems 来处理.维护自己的 gem 源有很多优点,就像建立一个 APT 源一样:你可以控制软件包的版本和应用,如果你需要,你也可以用它来分发你的 gems.

怎么办呢....

1.创建/etc/puppet/modules/repo/manifests/gem-server.pp 文件,内容,如下:

```
class repo::gem-server {
include apache
file { "/etc/apache2/sites-available/gemrepo":
source => "puppet:///modules/repo/gemrepo.conf",
require => Package["apache2-mpm-worker"],
notify => Service["apache2"],
}
file { "/etc/apache2/sites-enabled/gemrepo":
ensure => symlink,
target => "/etc/apache2/sites-available/gemrepo",
require => File["/etc/apache2/sites-available/gemrepo"],
notify => Service["apache2"],
}
file { "/var/gemrepo":
ensure => directory,
}
}
```

2.创建/etc/puppet/modules/repo/files/gemrepo.conf,文件,内容如下:

```
<VirtualHost *:80>
ServerAdmin john@bitfieldconsulting.com
ServerName gems.bitfieldconsulting.com
ErrorLog logs/gems.bitfieldconsulting.com-error_log
CustomLog logs/gems.bitfieldconsulting.com-access_log common
Alias / /var/gemrepo/
<Location />
Options Indexes
</Location>
</VirtualHost>
```

3.添加下面代码到你的节点:

```
node cookbook {
include repo::gem-server
}
```

4.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304949279'
notice: /Stage[main]/Repo::Gem-server/File[/etc/apache2/sites-
available/gemrepo]/ensure: defined content as '{md5}ae1fd948098f14
503de02441d02a825d'
info: /Stage[main]/Repo::Gem-server/File[/etc/apache2/sites-
available/gemrepo]: Scheduling refresh of Service[apache2]
notice: /Stage[main]/Repo::Gem-server/File[/etc/apache2/sites-
```

enabled/gemrepo]/ensure: created
info: /Stage[main]/Repo::Gem-server/File[/etc/apache2/sites-
enabled/gemrepo]: Scheduling refresh of Service[apache2]
notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refresh'
from 2 events
notice: /Stage[main]/Repo::Gem-server/File[/var/gemrepo]/ensure:
created
notice: Finished catalog run in 6.52 seconds

它是如何工作的....
极其像 APT 源,并遵循相同的原则,我们定义了一个 gem 仓库的目录,并定义了一个虚拟主机,确保它能响应 gems.bitfieldconsulting.com 请求.

还有更多...
同样的,你如果把某些东西放到 gem 仓库也是非常有用的,下面会演示如何去做,以及如何配置节点能访问 gem 仓库.

添加 gems

增加一个新的 gems 到你的仓库非常简单.把 gem 放到 /var/gemrepo/gems ,并在 /var/gemrepo 目录运行 gem generate_index 命令.
 # gem generate_index

使用 gem 仓库

和 APT 仓库一样,确保节点知道 gems 源主机的主机名 bitfieldconsulting.com,,它通常使用 Puppet 来部署主机条目,或者配置 DNS.
然后,你可以在 Puppet 指定一个软件包,如下所示:
package { "json":
provider => "gem",
source => "http://gems.bitfieldconsulting.com",
}

从源码包自动构建包

压缩包会严重损害系统的稳定,当使用一个发行版或者第三方软件包,或者回滚你的软件包,通常都需要从源码包进行构建,有时有许多事要做.创建 Debain 软件包(或者其它各种格式的软件包)是一个漫长而且容易出错的过程,通常并不是总有时间或者一批要做.
如果你要从源码编译一个程序,Puppet 至少可以帮助你实现这个过程,一般程序是自动化的,否则你要手工来完成.
下载源码包

解压缩 tarball
编辑和构建程序
安装程序
在这个例子,我们将从源码包来构建 OpenSSL(虽然在生产环境,你应该使用分行版式的软件包,但是它的确是一个很有用的演示).

怎么呢...
1.添加以下内容到你的代码:
```
exec { "build-openssl":
cwd => "/root",
command => "/usr/bin/wget ftp://ftp.openssl.org/source/
openssl-0.9.8p.tar.gz && /bin/tar xvzf openssl-0.9.8p.tar.gz && cd
openssl-0.9.8p && ./Configure linux-generic32 && make install",
creates => "/usr/local/ssl/bin/openssl",
logoutput => on_failure,
timeout => 0,
}
```
2.运行 Puppet:(可能需要一段时间)
```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304954159'
notice: /Stage[main]//Node[cookbook]/Exec[build-openssl]/returns:
executed successfully
notice: Finished catalog run in 554.00 seconds
```

它是如何工作的.....

exec 命令是由 5 个阶段,由&&分隔操作.这也意味着任何子命令执行失败,整个命令将停止并失败.它是一个非常有用的构造,只要你确保每个子命令都能成功执行,按顺序再执行下一个命令.
1.第 1 个步骤是下载源码包:
```
/usr/bin/wget ftp://ftp.openssl.org/source/openssl-0.9.8p.tar.gz
```
2.第 2 步骤是解压缩源码包:
```
/bin/tar xvzf openssl-0.9.8p.tar.gz
```
3.第 3 步骤是改变工作目录进入解压后的源码包的目录:
```
cd openssl-0.9.8p
```
4.第 4 步骤是运行 configure 脚本(这通常是由你根据实际情况自定义选项或者默认)
```
./configure linux-generic32
```
5.最后一步是构建和安装软件:
```
make install
```
6.因此这是一个漫长的过程,不想每次运行 puppet 它都运行,我们指定了一个条件,条件为文件是否存在(即文件是否已经被创建):
```
creates => "/usr/local/ssl/bin/openssl",
```

如果你出于何种原因需要重建,可以删除此文件.
1.并不是每次编译都会出问题,我们指定了 logoutput 参数,logoutput 会记录,我们的构建过编译和构建过程.
logoutput => on_failure,
3. 最后,因为编译可能需要一段时间,我们设置了超时参数为 0.
4. (默认情况下,Puppet 执行 exec 命令超过 5 分钟后即超时)
timeout => 0,

还有更多...

如果你从源代码构建不少的软件包,可以转换上述方式为 define,那是非常值得的.因此你可以使用或多或者少的相同的代码来构建每个包.

比较软件包版本

软件包版本号是件奇怪的事情,它们看起来像十进制数字,但他们没有往往一个版本号形式像 2.6.4,例如,如果你需要比较两个软件包的版本号,你不能做一个简单的字符串比较:2.6.4 会认为比 2.6.12 要大.比较数字是不能完成工作,因为但不是一个有效的数字.Puppet 的 versioncmp 功能就派上用场了,如果你传递两个参数像版本号,它会对它们比较,并返回一个值.
versioncmp(A, B)

如果 A 和 B 相等返回 0
如果 A 版本号高于 B 会返回大于 1 的数字.
如果 A 版本号低于 B 会返回小于 0 的数字.

怎么办呢...

1.添加以下内容到你的代码:
$app_version = "1.2.2"
$min_version = "1.2.10"
if versioncmp($app_version, $min_version) >= 0 {
notify { "Version OK": }
} else {
notify { "Upgrade needed": }
}

2.运行 Puppet:
notice: Upgrade needed
3.我们现在改$app_version 的值:
$app_version = "1.2.14"

4.再次运行 Puppet:
notice: Version OK

它是如何工作的...
我们指定的 minimum 可接受的最低版本($min_version)是 1.2.10.因此,在第一个例子,我们是想比较$app_version,其值为1.2.2.一个简单的字符串数字比较(例如,在 Ruby 中,)会给出错误的结果,但是 versioncmp 返回正确的结果,1.2.2 小于 1.2.10 并通知我们需要进行版本更新.

在第二个例子中,$app_version 的值现在是 1.2.14. 那么 versioncmp 正确的确认$app_version 大于$min_version.因此我们取得 Version OK 信息 .

第六章　　用户和其它资源

"How good the design is doesn't matter near as much as whether the design is getting better or worse. If it is getting better, day by day, I can live with it forever. If it is getting worse, I will die."
— Kent Beck

在本章中,我们将学习:
- 使用虚拟资源
- 使用虚拟资源管理用户
- 管理用户的 SSH 访问
- 管理用户的自定义文件
- 有效的分发 cron 任务
- 当文件更新时运行命令
- 使用主机资源
- 使用多个文件来源
- 使用文件资源递归来分发目录树
- 清理旧文件
- 使用 schedules 资源
- 审计资源
- 临时禁用资源
- 管理时区

使用虚拟资源

什么是虚拟资源,为什么我们需要使用虚拟资源?让我们来看下一个典型解决方案,虚拟资源是有用的.

你负责两个应用程序,FaceSquare 和 Twitstagram.两者都是由 apache 运行的 web 应用程序.FaceSquarer 的定义(define)看起来可能像这样的:

```
class app::facesquare {
package { "apache2-mpm-worker": ensure => installed }
...
}
```

Twitstagram 的定义(define)看起来可能像这样的:

```
class app::twitstagram {
package { "apache2-mpm-worker": ensure => installed }
...
}
```

我们需要整合两个应用程序到单一的服务器上,看起来一切都很好.

```
node micawber {
include app::facesquare
include app::twitstagram
}
```

现在 Puppet 会编译错误,因为你试图定义两个相同名称的 apache2-mpm-worker 资源 .

```
err: Could not retrieve catalog from remote server: Error 400 on SERVER:
Duplicate definition: Package[apache2-mpm-worker] is already defined in
file /etc/puppet/modules/app/manifests/facesquare.pp at line 2; cannot
redefine at /etc/puppet/modules/app/manifests/twitstagram.pp:2 on node
cookbook.bitfieldconsulting.com
```

你可以从一个类中删除重得定义的软件包,如果你的服务器没有安装 Apache,而你又执行 app 类,那么会执行失败.

你可以解决这个问题,是将自己的 Apache 类添加上,并执行下 Apache 类,Puppet 不在乎你多次执行同一个类.但是这样有个缺点,每个都必须有它自己的类避免潜在的冲突.

虚拟资源可以解救.虚拟资源就像是个正常的资源,特别的是它**以@字符开始:**

```
@package { "apache2-mpm-worker": ensure => installed }
```

你可以认为它像一个'FYI'(仅供参考)的资源.我只是告诉你关于这个资源,其实我不希望你做任何事情.Puppet 会阅读并记住虚拟资源的定义,但不会实际创建资源真到你使用 realize 功能,告诉它创建该资源.

```
realize( Package["apache2-mpm-worker"] )
```

你可以在你想要的资源里多次调用 realize,并不会冲突.因此,虚拟资源常见的使用场景为,当你有好几个不同的类但都需要相同的某个资源,而他们可能是需要在同一个节点上共存.

怎么办呢...

1.创建一个新的 app 模块

```
# mkdir -p /etc/puppet/modules/app/manifests
```

2.创建/etc/puppet/modules/app/manifests/init.pp 文件,内容如下:

```
import "*"
```

3.创建/etc/puppet/modules/app/manifests/facesquare.pp 文件,内容如下:

```
class app::facesquare {
realize( Package["apache2-mpm-worker"] )
}
```

4.创建 /etc/puppet/modules/app/manifests/twitstagram.pp 文件,内容如下:

```
class app::twitstagram {
realize( Package["apache2-mpm-worker"] )
}
```

5.创建/etc/puppet/modules/admin/manifests/virtual-packages.pp,内容如下:

```
class admin::virtual-packages {
@package { "apache2-mpm-worker": ensure => installed }
}
```

6.在节点上添加以下执行类:

```
node cookbook {
include admin::virtual-packages
include app::facesquare
include app::twitstagram
}
```

7.运行 Puppet

它是如何工作.....

admin::virtual-packages 类在一个地方,你定义软件包为虚拟资源,所有节点都可以执行这个类,你也可以添加所有的虚拟软件包放到这点.他们都不会被执行,直到你调用 realize,才会安装到节点.

```
class admin::virtual-packages {
@package { "apache2-mpm-worker": ensure => installed }
}
```

需要 Apache 软件包的类可以调用 realize 实现这个虚拟资源.

```
class app::twitstagram {
realize( Package["apache2-mpm-worker"] )
}
```

Puppet 知道,因为你的资源被设置为虚拟,你打算多个地方引用相同的软件包,并不是有意的创建具有的两个相同名称的两个资源.因此,Puppet 会正确的执行.

还有更多.....

为了实现虚拟资源,你也可以使用 collection(收集)的用法.

```
Package <| title = "apache2-mpm-worker" |>
```

使用这种语法的好处是,你不仅限于资源的名称,也可以使用 tag,例如:

```
Package <| tag = "security" |>
```

或者,你也可指定所有实例的资源类型型,留下查询部分空白:
Package <| |>
另请参阅:
使用虚拟资源来管理用户

使用虚拟资源来管理用户

虚拟资源可以派上用场来管理用户,那其是一个好的例子,假设你需要下面的设置:你有三个用户,John, Graham,和 Steven.管理大量的机器,你需要简化管理,你有两种用户定义的类:开发和系统管理员.所有机器都需要执行系统管理员,但是只有部分机器开发人员需要能够访问.

```
node server {
include user::sysadmins
}
node webserver inherits server {
include user::developers
}
```

john 是系统管理员,而 Steven 是个开发人员,但是 Graham 即是系统管理员,又是开发.因此他需要在两个组里.我最终会导致重复定义用户 Graham 而使 webserver 上报冲突.

为了避免这种情况,Puppet 社区通过了将所有用户标记为虚拟,并定义在一个 user::virtual 类里,每台机器如果需要只需要使用 realize 执行 user::virtual 类.

怎么办呢...
1.创建一个用户模块:
 # mkdir -p /etc/puppet/modules/user/manifests
2.创建 /etc/puppet/modules/user/manifests/init.pp 文件,内容如下:

```
import "*"
```

3.创建/etc/puppet/modules/app/manifests/virtual.pp 文件,内容如下:

```
class user::virtual {
@user { "john": }
@user { "graham": }
@user { "steven": }
}
```

4.创建 /etc/puppet/modules/app/manifests/developers.pp 内容如下:

```
class user::developers {
realize( User["graham"],
User["steven"] )
}
```

5.创建/etc/puppet/modules/app/manifests/sysadmins.pp ,内容如下:

```
class user::sysadmins {
```

realize(User["john"],
User["graham"])
}
6.添加以下内容到节点:
include user::virtual
include user::sysadmins
include user::developers
7.运行 Puppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305554239'
notice: /Stage[main]/User::Virtual/User[john]/ensure: created
notice: /Stage[main]/User::Virtual/User[steven]/ensure: created
notice: /Stage[main]/User::Virtual/User[graham]/ensure: created
notice: Finished catalog run in 2.36 seconds

它是如何工作的....
每个节点执行 user::virtual 类,所有服务器都继承这个基本类.这个类将定义你的组织或者站点的所有用户.这些用户也应该包括那些已经存在,但是只是为了运行应用程序或者某种服务(例如,apache 或者 git).然后你可以根据用户进行分组(不是 UNIX 系统里的组,但是有可能是不同的团队或者工作的角色)-开发者和系统管理员就是一个典型的例子.才分一个组分一个类,可以调用 realize 并执行类实现用户.
class user::sysadmins {
realize(User["john"],
User["graham"])
}

然后你可以在任何需要的地方执行这个类,而不必担心由于同一个用户有多处定义而造成的冲突.

另请参阅
使用虚拟资源
管理自定义用户文件

管理用户 ssh 访问

账户和口令保护 SSH 密钥是常用于接受访问控制的最佳实践,Puppet 易于管理这些是因为内置了 ssh_authorized_key 类型. 结合虚拟用户,在上一节所述中,你可以创建一个 define ,define 包括用户和 ssh_authorized_key.这也将是在添加自定义文件和其它用户的资源非常有用.

怎么办呢....
1.更改 user::virtual 类,在上节中创建并管理虚拟资源,按照下面步骤:

```
class user::virtual {
define ssh_user( $key ) {
user { $name:
ensure => present,
managehome => true,

}
ssh_authorized_key { "${name}_key":
key => $key,
type => "ssh-rsa",
user => $name,
}
}
@ssh_user { "phil":
key =>”aC1yc2EAAAABIwAAAIEA3ATqENg+GWACa2BzeqTdGnJhNoBer8x6pfWkzNzeM8Zx7/
2Tf2pl7kHdbsiTXEUawqzXZQtZzt/j3Oya+PZjcRpWNRzprSmd2UxEEPTqDw9LqY5S
2B8og/NyzWaIYPsKoatcgC7VgYHplcTbzEhGu8BsoEVBGYu3IRy5RkAcZik=",
}
}
```

2.在结点上执行下面内容

```
realize( User::Virtual::Ssh_user["phil"] )
```

3.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305561740'
notice: /Stage[main]/User::Virtual/User::Virtual::Ssh_user[phil]/
User[phil]/ensure: created
notice: /Stage[main]/User::Virtual/User::Virtual::Ssh_user[phil]/
Ssh_authorized_key[phil_key]/ensure: created
notice: Finished catalog run in 1.04 seconds
```

是如何工作的...

我们已经创建了一个新定义名称为 ssh_user,其中 ssh_user 包括用户资源本身还包括分配 ssh_authorized_key.

```
define ssh_user( $key ) {
user { $name:
ensure => present,
managehome => true,
}
ssh_authorized_key { "${name}_key":
key => $key,
```

```
type => "ssh-rsa",
user => $name,
}
}
```

然后,接下我们创建一个 phil 用户的虚拟实例 ssh_user.

```
@ssh_user { "phil":
key =>"
B3NzaC1yc2EAAAABIwAAAIEA3ATqENg+GWACa2BzeqTdGnJhNoBer8x6pfWkzNzeM8Zx7/
2Tf2pl7kHdbsiTXEUawqzXZQtZzt/j3Oya+PZjcRpWNRzprSmd2UxEEPTqDw9LqY5S2B8o
g/NyzWaIYPsKoatcgC7VgYHplcTbzEhGu8BsoEVBGYu3IRy5RkAcZik=",
}
```

回想一下,因为资源是虚拟的,Puppet 会注意到这一点,只到调用 realize 否则不会实际创建什么.最后,我们在节点上添加以下内容:

```
realize( User::Virtual::Ssh_user["phil"] )
```

这实际上会创建用户和 authorized_keys 文件,authorized_keys 文件是用户的公钥.

还有更多...

正如我们在前面章节所看到了,组织用户划分为组是个好主意,像这样修改这个类:

```
class user::sysadmins {
search User::Virtual
realize( Ssh_user["john"],
Ssh_user["graham"] )
}
```

search User::Virtual 只是用避免杂乱,它允许你直接引用 Ssh_user 而不需要每次都添加 User::Virtual::前缀.

如果你得到一个像这样的错误:

```
err: /Stage[main]/User::Virtual/User::Virtual::Ssh_user[graham]/Ssh_
authorized_key[graham_key]: Could not evaluate: No such file or directory
- /home/graham/.ssh
```

这可能是因为之前创建个 graham 用户,但没有使用 Puppet 来管理家目录;在这种情况下,Puppet 不会自动的为 authorized_keys 文件创建.ssh 目录.运行:

```
# userdel graham
```

再次运行 Puppet 来解决这个问题.

管理用户自定义文件

用户像猫,常常觉得有必要去纪念他们的领地,不像猫.用户往往自定义他们的 shell 环境,终端颜色,别名,等等.通常在自己的家目录,有很多以.开头的文件:例如

.bash_profile.

你可以通过修改 user::virttual::ssh_user 类管理.开头的任何文件,.开头的文件可以放进 puppet 仓库.

怎么办呢...

1.像这样修改 user::virtual 类:

```
class user::virtual {
define user_dotfile( $username ) {
file { "/home/${username}/.${name}":
source => "puppet:///modules/user/${username}-
${name}",
owner => $username,
group => $username,
}
}
define ssh_user( $key, $dotfile = false ) {
user { $name:
ensure => present,
managehome => true,
}
ssh_authorized_key { "${name}_key":
key => $key,
type => "ssh-rsa",
user => $name,
}
if $dotfile {
user_dotfile { $dotfile:
username => $name,
}
}
}
@ssh_user { "john":
key=>"aC1yc2EAAAABIwAAAIEA3ATqENg+GWACa2BzeqTdGnJhNoBer8x6pfWkzNzeM8Zx7/
2Tf2pl7kHdbsiTXEUawqzXZQtZzt/j3Oya+PZjcRpWNRzprSmd2UxEEPTqDw9LqY5S
2B8og/NyzWaIYPsKoatcgC7VgYHplcTbzEhGu8BsoEVBGYu3IRy5RkAcZik=",
dotfile => [ "bashrc", "bash_profile" ],
}
}
```

2.创建/etc/puppet/modules/user/files/john-bashrc 文件,内容如下:

```
export PATH=$PATH:/var/lib/gems/1.8/bin
```

3.创建/etc/puppet/modules/user/files/john-bash_profile 文件,内容如下:

. ~/.bashrc

4.运行 Puppet.

它是如何工作的...
我们添加了一个新的定义了,名称为 user_dotfile.如果用户希望有.文件,那么 user_dotfile 将会被调用一次.

```
@ssh_user { "john":
Key => ...
dotfile => [ "bashrc", "bash_profile" ],
```

你可以提供单一的.开头的文件,或者像上面以数组列表的方式.对于每个.开头文件.user_dotfile 会在 modules/user/files 目录下寻找相应的源文件. 例如:
.barchrc 文件,Puppet 将会以这样的方式去 modules/user/files/john-bashrc
找到后复制到节点 /home/john/.bashrc

另请参阅:
使用虚拟资源来管理用户

有效分发 cron 任务

当你有许多服务器,需要执行相同的 cron 作业,不在同一时间集中运行他们通常是个好主意.如果作业是用来访问一个普通服务器,那么会给该服务器带来太多压力,即使他们不想,那么所有服务器都在同一时刻等待,这可能会影响它提供其它服务的能力.Puppet 的 inline_template 功能允许我们使用的 Ruby 的逻辑来根据主机名来设置不同的运行时间.

怎么办呢...
1.添加以下内容到节点:

```
define cron_random( $command, $hour ) {
cron { $name:
command => $command,
minute => inline_template("<%= (hostname+name).hash.abs % 60 %>"),
hour => $hour,
ensure => "present",
}
}
cron_random { "hello-world":
command => "/bin/echo 'Hello world'",
hour => 2,
}
cron_random { "hello-world-2":
```

command => "/bin/echo 'Hello world'",
hour => 1,
}

2.运行 Puppet:
 # puppet agent --test

info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305713506'
notice: /Stage[main]//Node[cookbook]/Cron_random[hello-world]/
Cron[hello-world]/ensure: created
notice: /Stage[main]//Node[cookbook]/Cron_random[hello-world-2]/
Cron[hello-world-2]/ensure: created
notice: Finished catalog run in 1.07 seconds

它是如何工作的

我们为每个 cron 作业选择了一个随机分钟数,也就是说,不是真正的随机(或者修改 Puppet 的每次运行时间),但或多或少保证为每个不同的每个主机上的 cron 作业.我们可以通过使用 Ruby 的哈希方法,计算任何数值对象,在这种情况下,一个字符串该值每次是相同的,看起来那个值是是随机的,它不会改变 Puppet 再次运行.hash 会生成一个大的整数,我们希望是从 0-59 之间,所以,我们使用了 Ruby(%)(模数)来运算并限制在此范围内的结果.虽然只有 60 个值,但是 hash 函数的设计目的是统一的生产尽可能的输出,所以,应该是很少的碰撞和分钟值应分布均匀.

我们想要在不同的机器上,根据不同的值来区分作业,因此我们使用主机名,以主机名来计算哈希值.但是,但是,我们也希望在同一台机器上,不同的作业,hash 值也是不同的.所以我们结合了 name 变量.这将是 cron 的作业名称(例如 Lhello-world).

还有更多.....

在这个例子中,我们只随机分钟的 cron 作业,并提供了小时的一部分定义.如果你有时需要指定一周中的某一天,你也可以把它添加到 cron_random,你可以设置参数,以及指定默认值:
define cron_random($command, $hour, $weekday = "*") {

如果你也想随机小时(例子如:在一天的任何埋单运行,需要均匀分配 24 小时),你
可以修改 cron_random 如下所示:
hour => inline_template("<%= (hostname+name).hash.abs % 24 %>"),
另请参阅:
从 cron 运行 Puppet

当一个文件更新时运行 puppet

这是一个非常普遍的模式,当一个指定文件更新时,Puppet 采取一些动作.例如,在 rsync 配置文件添加了版片段,每个片段文件都调用 exec 来管理,当主配置文件(rsync.conf)有改动时.通常 puppet 每次运行时都会执行 exec 资源,除非你指定了以下某个参数:

creats
onlyif
unless
refreshonly => true

refreshonly 参数意味着 exec 如果它收到从另一个资源的通知就运行.(例如,文件)即可以这样理解,通知 n 次,就执行 n 次.

准备...

1.安装 nginx 软件包
 # apt-get install nginx

如何去做...
1.创建一个 nginx 模块,目录结构如下:
 # mkdir /etc/puppet/modules/nginx
 # mkdir /etc/puppet/modules/nginx/files
 # mkdir /etc/puppet/modules/nginx/manifests
2.创建/etc/puppet/modules/nginx/manifests/init.pp 文件,内容如下:
import "*"
3.创建/etc/puppet/modules/nginx/manifests/nginx.pp 文件,内容如下:

```
class nginx {
package { "nginx": ensure => installed }
service { "nginx":
enable => true,
ensure => running,
}
exec { "reload nginx":
command
=> "/usr/sbin/service nginx reload",
Require => Package["nginx"],
refreshonly => true,
}
file { "/etc/nginx/nginx.conf":
source => "puppet:///modules/nginx/nginx.conf",
notify => Exec["reload nginx"],
}
```

}

4.复制 nginx.conf 到新模块:

cp /etc/nginx/nginx.conf /etc/puppet/modules/nginx/files

5.添加以下内容到节点

include nginx

6.从复制的 nginx.conf 文件做个测试,改变文件内容.

 # echo \# >>/etc/puppet/modules/nginx/files/nginx.conf

7.运行 Puppet:

 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303745502'
--- /etc/nginx/nginx.conf
2010-02-15 00:16:47.000000000 -0700
+++ /tmp/puppet-file20110425-31239-158xcst-0
09:39:49.586322042 -0600
2011-04-25
@@ -48,3 +48,4 @@
#
#
proxy
on;
}
 # }
+#
info: FileBucket adding /etc/nginx/nginx.conf as {md5}7bf139588b5e
cd5956f986c9c1442d44
info: /Stage[main]/Nginx/File[/etc/nginx/nginx.conf]: Filebucketed
/etc/nginx/nginx.conf to puppet with sum 7bf139588b5ecd5956f986c9c
1442d44
notice: /Stage[main]/Nginx/File[/etc/nginx/nginx.conf]/content:
content changed '{md5}7bf139588b5ecd5956f986c9c1442d44' to '{md5}d
28d08925174c3f6917a78797c4cd3cc'
info: /Stage[main]/Nginx/File[/etc/nginx/nginx.conf]: Scheduling
refresh of Exec[reload nginx]
notice: /Stage[main]/Nginx/Exec[reload nginx]: Triggered 'refresh'
from 1 events
notice: Finished catalog run in 1.69 seconds

它是如何工作的...

对于大多数服务,你可以简单的定义了一个服务资源并从 config 文件获得通知.这将导致 Puppet 会重启该服务以应用配置文件变化.但是,Nginx 有时不能正确的重新启动,尤其是使用 puppet 来重新启动 nginx,所以我从网站上做了个补救措施,使用 puppet 来运行

/etc/init.d/nginx reload 来替代 restart,这里的
它是如何工作的.

exec 资源的 refreshonly 参数设置为 true:

```
exec { "reload nginx":
command => "/usr/sbin/service nginx reload",
require => Package["nginx"],
refreshonly => true,
}
```

因此只要收到 notfify,exec 资源就运行.

配置文件资源提供必要的 notify,如果配置文件有改动:

```
file { "/etc/nginx/nginx.conf":
source => "puppet:///modules/nginx/nginx.conf",
notify => Exec["reload nginx"],
}
```

每当 Puppet 需要更新这个文件,它将运行 exec,调用/usr/sbin/service nginx reload 来应用变化.

还有更多...
你可以使用相似的模式,每个资源更新,任何地方需要触发动作.
包括以下可能的用途:
触发服务重新加载
运行语法检查,然后重新启动服务
串联配置片段
运行测试
exec 链

当单个文件更新时,如果你有几个命令需要去执行,它需要在所有命令前都需要提供 subscribe 参数指向该文件.而不是文件的 notfiy 命令.但是效果是一样的.

使用 host 资源

经常机器需要移动,尤其是云基础架构,因此一个特定的机器 IP 经常会变化.正因为如此,这显然是一个坏主意在配置文件里使用硬件编码.如果一台机器需要访问另一台:例如 : 一个应用服务器访问数据库服务器,它最好是使用主机名而不是一个 IP 地址.但是如何映射主机名到 IP 地址?这通常是使用 dns.但是一个小的组织可能不会有一个 DNS 服务器,但是以些大型组织可能需要时间去同步 dns 变化,此外,dns 信息同步到其它机器需要,因此,以确保快速一致的 IP 地址更新是使用 Puppet 来管理本地的/etc/hosts 文件.

1.添加以下内容到代码:

```
host { "www.bitfieldconsulting.com":
ip => "109.74.195.241",
target => "/etc/hosts",
ensure => present,
}
```

2.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305716418'
notice: /Stage[main]//Node[cookbook]/Host[www.bitfieldconsulting.com]/ensure: created
info: FileBucket adding /etc/hosts as {md5}977bf5811de978b7f0413019e77b4abe
notice: Finished catalog run in 0.21 seconds
```

这是如何工作...

没有解释的必要。

还有更多...

把您的主机资源组织成类能有所帮助.例如,你可以把所有 DB 服务器组织成一个类,类名为 admin::allhosts.在所有 web 服务器上执行这个类.

有些服务器可能需要在多个类中定义(例如,一个数据库服务器也许也是一个资源库服务器),虚拟资源可以解决这个问题.例如,你可以定义所有主机在一个虚拟类中:

```
class admin::allhosts {
@host { "db1.bitfieldconsulting.com":
...
}
}
```

然后在类中使用 realize 实现各主机需要的 hosts

```
class admin::dbhosts {
realize( Host["db1.bitfieldconsulting.com"] )
}
class admin::repohosts {
realize( Host["db1.bitfieldconsulting.com"] )
}
```

使用多个 file sources

puppet 的一个实用功能你是可以指定多个源文件.puppet 会在列表中寻找他们.如果第一个没有找到,就查找下一条,依此类推.你可以指定一个默认的源文件,如果特定的文件没有找到,

甚至一系列的源文件替代品.

怎么办呢....
1.添加这个类到代码:
class mysql::app-config($app) {
file { "/etc/my.cnf":
source => ["puppet:///modules/admin/${app}.my.cnf",
"puppet:///modules/admin/generic.my.cnf",],
}
}

2.创建/etc/puppet/modules/admin/files/minutespace.my.cnf 文件,内容如下:
 # MinuteSpace config file
3.创建/etc/puppet/modules/admin/files/generic.my.cnf 文件,内容如下:
 # Generic config file
4.添加以下内容到结点:
class { "mysql::app-config": app => "minutespace" }
5.运行 Puppet
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305897071'
notice: /Stage[main]/Mysql::App-config/File[/etc/my.cnf]/ensure:
defined content as '{md5}24f04b960f4d33c70449fbc4d9f708b6'
notice: Finished catalog run in 0.35 seconds
6.检查 puppet 已部署的应用程序特定的配置文件:
 # cat /etc/my.cnf
 # MinuteSpace config file
7.现在我们修改节点定义:
class { "mysql::app-config": app => "shreddit" }
8.再次运行 Puppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305897864'
--- /etc/my.cnf 2011-05-20 13:17:56.006239489 +0000
+++ /tmp/puppet-file20110520-15575-1icobgs-0
13:24:25.030296062 +0000
2011-05-20
@@ -1 +1 @@
-# MinuteSpace config file
+# Generic config file
info: FileBucket adding /etc/my.cnf as {md5}24f04b960f4d33c70449fbc4d9f708b6
info: /Stage[main]/Mysql::App-config/File[/etc/my.cnf]:

Filebucketed /etc/my.cnf to puppet with sum 24f04b960f4d33c70449fb
c4d9f708b6
notice: /Stage[main]/Mysql::App-config/File[/etc/my.cnf]/content:
content changed '{md5}24f04b960f4d33c70449fbc4d9f708b6' to '{md5}b
3a6e744c3ab78dfb20e46ff55f6c33c'
notice: Finished catalog run in 0.93 seconds

是如何工作的...

我们定义了/etc/my.cnf 有两个源:
file { "/etc/my.cnf":
source => ["puppet:///modules/admin/${app}.my.cnf",
"puppet:///modules/admin/generic.my.cnf",],
}

$app 的值可以是任意的,但它会传递到内中,因此在第一个例子中,我们传递了一个 minutespace 值.
class { "mysql::app-config": app => "minutespace" }
因此 Puppet 将首先寻找 modules/admin/files/minutespace.my.cnf. 此文件存在因此将使用它,到目前为止,一切正常.

接下来,我们改变 app 的值为 shredddit.Puppet 会寻找 modules/admin/files/shreddit.my.cnf. 此文件不存在,因此 Puppet 会尝试查找列表中的其它 source:modules/admin/files/generic.my.cnf. 此文件存在,将会被部署.

还有更多...
你可以使用这一招在任何地方,如果你有文件资源.例如.一些节点可能需要定制的配置文件,不是其它的,所以你可以类似的去做:
file { "/etc/stuff.cfg":
source => ["puppet:///modules/stuff/${hostname}.cfg",
"puppet:///modules/stuff/generic.cfg"],
}

接下来你就把正常配置放进 generic.cfg.如果机器 cartman 需要一个特殊的配置,你可以把它的配置文件放进 cartman.cfg 中,这将优先使用 generic 文件,因为它是在数组的前面.

使用文件递归分发目录树

"To understand recursion, you must first understand recursion."
— Saying

当你使用 Puppet 分发文件,而所有文件同是在同一个目录下,那你可以考虑使用递归来替代.

如果你在目录中使用 recurse 参数 ,Puppet 会复制目录,以及目录下的子目录里面所有文件到其它节点.

怎么办呢...

1.在 Puppet 仓库中创建一个合适的目录树:

```
# mkdir /etc/puppet/modules/admin/files/tree
# mkdir /etc/puppet/modules/admin/files/tree/a
# mkdir /etc/puppet/modules/admin/files/tree/b
# mkdir /etc/puppet/modules/admin/files/tree/c
# mkdir /etc/puppet/modules/admin/files/tree/a/1
```

2.添加以下内容到代码:

```
file { "/tmp/tree":
source => "puppet:///modules/admin/tree",
recurse => true,
}
```

3.运行 Puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304768523'
notice: /Stage[main]//Node[cookbook]/File[/tmp/tree]/ensure:
created
notice: /File[/tmp/tree/a]/ensure: created
notice: /File[/tmp/tree/a/1]/ensure: created
notice: /File[/tmp/tree/b]/ensure: created
notice: /File[/tmp/tree/c]/ensure: created
notice: Finished catalog run in 1.25 seconds
```

它是如何工作的...

如果某个文件资源设置了 rescurse 参数,它是一个目录,Puppet 不仅部署目录本身,而是其所有内容(包括子目录及其目录下的内容).这是很好的方法,如果你要把目录树部署到节点,或者快速的创建大量的单一的文件资源.

还有更多...

有时你要部署文件到现有目录的,但是要删除原目录下所有不是 Puppet 管理的文件,例如:在 Ubuntu 中/etc/apt/sources.list.d 目录,你可能需要确保所有文件都需要有 puppet 来管理.purge 参数将会帮助你.在 Puppet 定义该目录为资源 :

```
file { "/etc/apt/sources.list.d":
ensure => directory,
recurse => true,
purge => true,
```

}

recurse 和 purge 相结合,将删除 /etc/apt/sources.list.d 目录下所有不是 puppet 部署的文件以及子目录.然后,你使用一个单独的资源位置,可以部署自己的文件.

```
file { "/etc/apt/sources.list.d/bitfield.list":
content => "deb http://packages.bitfieldconsulting.com/ lucidmain\n",
}
```

如果子目录中有你不想删除文件,只需要定义子目录为 Puppet 资源,将其单独区分开来.

```
file { "/etc/exim4/conf.d/acl":
ensure => directory,
}
```

清理过期文件

Puppet 的 tidy 资源可以帮助你清理过期的或者无用的文件,来减少磁盘的使用情况.例如,如果你启用了 Puppet 报告,如之前一节中所描述的那样产生报告.你可以想定期删除过期文件.

如何去做 ...

1.添加以下内容到代码:

```
tidy { "/var/lib/puppet/reports":
age => "1w",
recurse => true,
}
```

2.运行 puppet

```
# puppet agent -–test
info: Retrieving plugin info: Caching catalog for cookbook.bitfieldconsulting.com
notice: /Stage[main]//Node[cookbook]/Tidy[/var/lib/puppet/reports]: Tidying
File[/var/lib/puppet/reports/cookbook.
bitfieldconsulting.com/201102241546.yaml]
notice: /Stage[main]//Node[cookbook]/Tidy[/var/lib/puppet/
reports]: Tidying File[/var/lib/puppet/reports/cookbook.
bitfieldconsulting.com/20110214727.yaml]
...
info: Applying configuration version '1306149187'
notice: /File[/var/lib/puppet/reports/cookbook.bitfieldconsulting.
com/201102241546.yaml]/ensure: removed
notice: /File[/var/lib/puppet/reports/cookbook.bitfieldconsulting.
com/201102141727.yaml]/ensure: removed ...
notice: Finished catalog run in 1.48 seconds
```

它是如何工作的...

puppet 会搜索指定路径的下所有匹配时间的参数:在这种情况下:1w(一个星期).你还可以搜索子目录(recurse => true)

任何标准匹配文件就会被删除.

还有更多...
你可以使用一个指定的时间单位来指定文件的时间为秒,分钟,小时,天,或者说周
像这样:
60s
180m
24h
30d
4w
你还可以指定文件大于你给定大小的文件应该被删除,像这样的:
size => "100m",
删除大于 100M 字节以上的文件.对于 kilobytes,可以使用 k,对于 bytes,可以使用 b.
请注意,如果你指定了 size 和 age 两个参数,它们是作为独立的处理标准.例如:如果
你指定:
age => "1d",
size => "512k",
Puppet 将会删除所有文件大于 512Kb 的,不论 age,接下来删除所有文件大于 1 天的,无论大小.

blog 可参见 http://www.mysqlops.com/2012/02/09/puppet-clean-file.html

在资源中使用 schedules

使用 schedule 资源,你可以控制其它资源得到应用.例如,内置的每天调度不是你所期望的:如果你指定了一个像这样的资源:

```
exec { "/usr/bin/apt-get update":
schedule => daily,
}
```

这将是每天都会应用一次.

稍微棘手的事情有关的时间表是,它并不能保证资源将每天被应用 一次,这只是一个限制,资源不会被卧应用于每天超过一次.或者是适用于所有的资源该资源将取决于何时运行 puppet 和 Puppet 是否正在运行.

正因为如此,schedule 是最好的用来限制其它资源:例如,你可以想确保 apt-get 的更新每小时运行不超过一次.或者说你维护的工作不是安排在白天工作时间内.为此,你需要创建自己的

schedule 资源.

怎么办呢...

1.添加以下内容到你的代码:

```
schedule { "not-in-office-hours":
period => daily,
range => [ "17:00-23:59", "00:00-09:00" ],
repeat    => 1,
}
exec { "/bin/echo Doing maintenance!":
schedule => "not-in-office-hours",
}
```

2.运行 Puppet.

它是如何工作...

我们已经创建了资源,名称为 not-in-office-hours 指定了重复时间表期间,每天下午 5 点后,上午 9 时或者之前所允许的时间范围:

```
period => daily,
range => [ "17:00-23:59", "00:00-09:00" ],
```

我们还指出了,在一个时段内该资源可以应用的执行次数为 1.

```
repeat =>1,
```

现在我们在 exec 资源中应用 schedule.

```
period => daily,
range => [ "17:00-23:59", "00:00-09:00" ],
```

如果没有指定 schedule 参数,这其中的资源将随 puppet 每次运行而运行.现在 Puppet 将检查 not-in-office-hours 资源: 时间是在允许范围内 在这一时间,该资源是否已经运行的时间超出了运行的次数

例如,让我们来看看发生什么,如果 Puppet 每隔 1 小时运行,

4pm:它超出了允许的时间范围内,因此不会做任何事情
5pm:在允许的时间范围内,在这个期间,该资源尚未运行,Puppet 将会应用该资源.
6pm:这是在允许的时间范围内,该资源已经运行了一次,所以它达到其最大的重得计数,puppet 将不会做任何事情,依此类推,直到第二天.

还有更多...

你可以增加 repeat 的参数,如果你想,例如,运行一个作业在一小时内不超过 6 次.

```
period => hourly,
repeat => 6,
```

请记住,这并不能保证一个小时该作业运行了 6 次,它只是规定了上限的限制,puppet 运行多久或者其它发生什么情况都是没有关系的.如果这个小时该作业已经运行了 6 次,将不会再运行.如果 Puppet 每天只运行一次,该作业将只会运行一次,因此,schedule 是最好的用于确保事情不会发生在特定的时间(或者不超过一个给定的频率).

审记资源

我曾经诊断过一个服务器,为什么 ping 没有响应,连接不上 ssh 或者控制台,最后解决了,当我打电话到位于服务器的位置,他们告诉我,在之前来过两名不明身份的男子,将服务器装进卡车并从前门出去了,这里的信息是,它知道谁对你的服务器做了些什么.

试运行模式,可以使用--noop 开关,是一种简单的方法来审记在 Puppet 控制下的任何机器变化,然而,Puppet 也有个专门的审记功能,它可以报告资源或者指定的属性变化.

怎么办呢...

1.使用 audit 参数定义一个资源.

```
file { "/etc/passwd":
audit => [ owner, mode ],
}
```

它是如何工作的...

audit 的 metaparameter(metaparameter 是个适用于任何资源,而不是特定类型) 告诉 Puppet 要记录和监视资源的某些方面,该值可以是你要审记的参数列表,在这种情况下,当 puppet 运行时,puppet 将会记录/etc/passwd 文件的属主和权限.如果其中任何有改变,例如,如果运行:

```
# chmod 666 /etc/passwd
```

在下次运行 Puppet 时,puppet 会应用变化,并会记录下改变.

```
notice: /Stage[main]//Node[cookbook]/File[/etc/passwd]/mode: audit
change: previously recorded value 644 has been changed to 666
```

还有更多...

此功能非常有用来审记大型网络中的机器所做的任何改变,无论是恶意的或者是意外,你还可以使用 tagmail 报告功能自动发送电子邮件,以报告审记变更和通知.这也是非常有用的可以保持对那些不是由 puppt 管理的文件监视.例如,在生产服务器上的应用程序代码.你可以在这里阅读更多关于审记功能:

http://www.puppetlabs.com/blog/all-about-auditing-with-puppet/

如果你只是想审计资源的一切,可以使用 all:

```
file { "/etc/passwd":
audit => all,
}
```

另请参阅:

使用试运行模式,以避免意外
通过电子邮件发送日志中包含特定的 tags

临时禁用资源

有时你想临时禁用一个资源,以便它不会干扰其他工作.例如,你可以要调整服务器上的配置文件设置直到到你想有确节的设置之前,在 puppet 检查之前,你想在此期间 Puppet 覆盖旧版本,所以你可以在资源里设置 noop metaparameter 参数.
noop=>true,

怎么办呢...

1.添加以下内容到代码:
file { "/tmp/test.cfg":
content => "Hello, world!\n",
noop => true,
}
2.运行 puppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1306159566'
notice: /Stage[main]//Node[cookbook]/File[/tmp/test.cfg]/ensure:
is absent, should be file (noop)
notice: Finished catalog run in 0.53 seconds
3.现在移除 noop 参数:
file { "/tmp/test.cfg":
content => "Hello, world!\n",
}
4.再次运行 puppet:
 # puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1306159705'
notice: /Stage[main]//Node[cookbook]/File[/tmp/test.cfg]/ensure:
defined content as '{md5}746308829575e17c3331bbcb00c0898b'
notice: Finished catalog run in 0.52 seconds

它是如何工作的...
我们在第一次运行 puppet,noop 参数被设置为 true,所以这个特定的资源,如果你运行 puppet --noop 标志一样.Puppet 会指出该资源将不会得到应用.在第二种情况下,noop 被删除了,该资源将像正常资源一样应用.

管理时区

"I try to take one day at a time, but sometimes several days attack at once."
- Ashleigh Brilliant

迟早,你会遇到一个奇怪的问题,你最终会追查到服务器有不同的时区,这是明智的,以避免此类的问题,确保所有的服务器都使用相同的时区,不论其地理位置(GMT 是合乎逻辑的选择)除非服务器是由太阳能供电的,我认为没有任何理由不重视时区设置.

怎么办呢...

1.创建/etc/puppet/modules/admin/manifests/gmt.pp 文件,内容如下:

```
class admin::gmt {
file { "/etc/localtime":
ensure => "/usr/share/zoneinfo/GMT",
}
}
```

2.增加下面内容到节点:

```
include admin::gmt
```

3.运行 puppet:

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304955158'
info: FileBucket adding /etc/localtime as {md5}02b73b0cf0d96e2f75c
ae56b178bf58e
info: /Stage[main]/Admin::Gmt/File[/etc/localtime]: Filebucketed
/etc/localtime to puppet with sum 02b73b0cf0d96e2f75cae56b178bf58e
notice: /Stage[main]/Admin::Gmt/File[/etc/localtime]/ensure:
ensure changed 'file' to 'link'
notice: Finished catalog run in 1.94 seconds
```

它是如何工作...

没有解释的必要。

还有更多...

如果你想使用不同的,在/usr/share/zoneinfo 下面选择相应的文件,例如,US/Eastern.