目录

中文 wiki：
官网文档：

第一章. puppet 简介

关于 puppet

puppet 是一种 Linux、Unix、windows 平台的集中配置管理系统，使用自有的 puppet 描述语言，可管理配置文件、用户、cron 任务、软件包、系统服务等。puppet 把这些系统实体称之为资源，puppet 的设计目标是简化对这些资源的管理以及妥善处理资源间的依赖关系。

puppet 采用 C/S 星状的结构，所有的客户端和一个或几个服务器交互。每个客户端周期的（默认半个小时）向服务器发送请求，获得其最新的配置信息，保证和该配置信息同步。每个 puppet 客户端每半小时(可以设置)连接一次服务器端，下载最新的配置文件,并且严格按照配置文件来配置服务器. 配置完成以后,puppet 客户端可以反馈给服务器端一个消息. 如果出错,也会给服务器端反馈一个消息.

puppet 的工作流程

1、 客户端 puppetd 调用 facter，facter 会探测出这台主机的一些变量如主机名、内存大小、IP 地址等。然后 puppetd 把这些信息发送到服务器端。

2、 服务器端的 puppetmaster 检测到客户端的主机名，然后会到 manifest 里面对应的 node 配置，然后对这段内容进行解析，facter 送过来的信息可以作为变量进行处理的，node 牵涉到的代码才解析，其它的代码不不解析，解析分几个过程：语法检查、然后会生成一个中间的伪代码，然后再把伪代码发给客户机。
3、 客户端接收到伪代码之后就会执行，客户端再把执行结果发送给服务器。
4、 服务器再把客户端的执行结果写入日志。

第二章.puppet 安装与验证

puppet 安装

这里先搭建一个简单的环境，一台 master，一台 client，然后就是 client 端需要从 master 端验证。

我们这里通过 epel 源来安装，当然也可以通过源码来安装。先安装 epel
rpm -ivh

puppet 需要 ruby 语言的支持，先安装 ruby
yum install ruby

安装 master 端，主要有两个 puppet-master、facter
yum install puppet-server –y 这里会自动关联安装 facter

安装 client 端，只要安装 puppet 就行了
yum install puppet

puppet 运行前的环境配置

先对两台 linux 做简单的配置，建议先关闭 iptables、selinux，puppet 需要 dns 的支持，我们这里直接使用 hosts 文件来解析，
master 的主机名为 puppet1.com，
sed -i 's/\(HOSTNAME=\).*/\1puppet1.com/g' /etc/sysconfig/network

client 的主机名为 puppet2.com，
sed -i 's/\(HOSTNAME=\).*/\1puppet1.com/g' /etc/sysconfig/network

修改两台 linux 的 hosts 文件添加两行：
211.211.211.140 puppet1.com
211.211.211.151 puppet2.com

接着配置 ssl 自动登录，方便后续维护
master 上先生成私钥和公钥:
[root@puppet-2 ~]# ssh-keygen -f ./aa/test
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): 这里可以设置密钥的密码
Enter same passphrase again:
Your identification has been saved in ./aa/test.这个是私钥
Your public key has been saved in ./aa/test.pub.这个是公钥
The key fingerprint is:
a2:78:a1:f1:2b:b2:d3:d9:db:38:39:a2:de:13:d8:b5 root@puppet2.com
The key's randomart image is:
……..

接着就是将公钥拷贝到 client 上，用命令 ssh-copy-id
[root@puppet-2 ~]# ssh-copy-id puppet2.com
The authenticity of host 'puppet2.com (211.211.211.151)' can't be established.
RSA key fingerprint is 20:26:0f:3d:76:fd:9e:5b:78:6b:1d:22:5d:64:9b:77.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'puppet2.com' (RSA) to the list of known hosts.
Now try logging into the machine, with "ssh 'puppet2.com'", and check in:

.ssh/authorized_keys 重点是这个自动应答文件

to make sure we haven't added extra keys that you weren't expecting.

当然也可以自己新建这个/root/.ssh/authorized_keys 文件，将 test.pub 公钥内容拷贝进去就行了

有时后会出现无法登录的情况 bad permissions: ignore key: /home/robin/.ssh/id_rsa

解决方法是：
chmod 755 ~/.ssh/
chmod 600 ~/.ssh/id_rsa ~/.ssh/id_rsa.pub
chmod 644 ~/.ssh/known_hosts

到这里就可以就可以 ssh puppet2.com 无密码登录了，当然也可以把私钥拷贝过去，实

现相互之间无密码登录。

client 端验证

先启动相关服务：

master 端：/etc/init.d/puppetmaster start

client 端不必启用什么服务，后续执行用 cron 来执行，执行验证申请
puppetd --test –server puppet1.com

master 端执行 puppetca –list 查看是否有 client 端的验证，接着运行验证
puppetca –s puppet2.com

此时 client 端就可以执行 puppetd --test –server puppet1.com 触发从 master 下发预先定义的配置，此命令会触发 master 解析 manifest 下的 site.pp 文件，看下一章。

第三章.puppet 结构和 manifest 语法

puppet 结构

树结构如下：
|-- puppet.conf #主配置配置文件
|-- fileserver.conf #文件服务器配置文件
|-- auth.conf #认证配置文件
|-- autosign.conf #自动验证配置文件
|-- tagmail.conf #邮件配置文件（将错误信息发送）
|-- manifests #文件存储目录(puppet 会先读取该目录的.PP 文件<site.pp>)
| --nodes
| puppetclient.pp
|-- site.pp #定义 puppet 相关的变量和默认配置。
|-- modules.pp #加载 class 类模块文件（include syslog）

```
|-- modules #定义模块
    | `-- syslog #以 syslog 为例
        |-- file
        |-- manifests
            |-- init.pp
        | `-- templates #模块配置目录
            |-- syslog.erb #erb 模板
```

先只要知道前面几个主要的用途就行，manifest 和 modules 是比较难的部分，需要熟悉 puppet 语言。

puppet manifest 语法

参考官方文档 learnning puppet.pdf 文档，里面讲解得很详细，不过是 English 的，
下载地址：http://docs.puppetlabs.com/puppetdocs-latest.tar.gz

未注明的情况下以下所有操作都在 master 端

首先要知道是 resources，
resources，官方定义是 Imagine a system's configuration as a collection of many independent atomic units; call them"resources."

包含类型可以是：

- A user account
- A specific file?
- A directory of files?
- A software package
- A running service
- A scheduled cron job
- An invocation of a shell command, when certain conditions are met.
- ....

一个 resource 的简单举例：

```
user { 'dave':
    ensure => present,
    uid => '507',
    gid => 'admin',
    shell => '/bin/zsh',
    home => '/home/dave',
    managehome => true,
}
```

将其保存到/etc/puppet/manifest/目录中，以.pp 后缀，可以通过命令解析执行：
puppet apply user.pp

这里执行会创建一个 uid 为 507，用户组为 admin，登录 shell 为/bin/zsh，home 目录为/home/dave 的 dave 用户

通过此举例可以了解，此 resources 申明了四个部分：Type、Title、Attributes、Values，主要是 Type 类型有很多，官方维护了一个 the type reference.可以去去官网查看，一个 resource type 有哪些可用参数可以通过命令 puppet describe <type>进行详细的查看

比如：[root@puppet-1 manifests]# puppet describe user
user
Manage users. This type is mostly built to manage system
users, so it is lacking some features useful for managing normal
users.

This resource type uses the prescribed native tools for creating
groups and generally uses POSIX APIs for retrieving information
about them. It does not directly modify `/etc/passwd` or anything.

- **ensure**
  The basic state that the object should be in. Valid values are
  `present`, `absent`, `role`.
- **gid**
  The user's primary group. Can be specified numerically or by name.
  Note that users on Windows systems do not have a primary group; manage
  groups , with the `groups` attribute instead.

- **groups**
  The groups to which the user belongs. The primary group should
  not be listed, and groups should be identified by name rather than by
  GID. Multiple groups should be specified as an array.

- **home**
  The home directory of the user. The directory must be created
  separately and is not currently checked for existence.

.....省略

这里此命令相当于 linux 系统中的 man 命令

第四章.常见的 resource type 介绍

常见的 resource type 有 user、file、service、package、exec

user 资源类型

主要用于新建、删除用户等操作

```
[root@puppet-1 manifests]# puppet resource user root
user { 'root':
  ensure           => 'present',
  comment          => 'root',
  gid              => '0',
  home             => '/root',
                password                                                    =>
'$6$KE/GESu3qR.XDlSf$2xpD4yPPMLVC8J.p2hjXr2FKR9WImtVlm6sCyXlMTMqx
dDLCQ2/V12/Jbd105EYilinZSgJBMWkGXbIcdNAVx.',
  password_max_age => '99999',
  password_min_age => '0',
  shell            => '/bin/bash',
  uid              => '0',
}
```

user 资源都比较好理解，主要是 ensure 参数有三个可用值：`present`创建，`absent`删除，`role`不知道.可以自个测试一下，详细参数可以参照 puppet describe user

file 资源类型

file 类型主要是用于创建、删除、修改文件及权限等操作

```
[root@puppet-1 manifests]# puppet resource file /etc/rc.local
file { '/etc/rc.local':
  ensure   => 'link',
  ctime    => 'Mon Nov 18 09:22:16 -0500 2013',
  group    => '0',
  mode     => '777',
  mtime    => 'Mon Nov 18 09:22:16 -0500 2013',
```

```
  owner   => '0',
  selrange => 's0',
  selrole  => 'object_r',
  seltype  => 'etc_t',
  seluser  => 'system_u',
  target   => 'rc.d/rc.local',
  type     => 'link',
}
```

这里参数比较多，详细参数的意思可以参照 puppet describe file 命令，这里列举一下常用的参数

content创建文件的内容

The desired contents of a file, as a string. This attribute is mutually exclusive with `source` and `target`.

可以是一个提前定义的一个数组或者是一个文件

- **ctime**

A read-only state to check the file ctime.

- **ensure**

Whether to create files that don't currently exist.
Possible values are *absent*, *present*, *file*, and *directory*.
Specifying `present` will match any form of file existence, and
if the file is missing will create an empty file. Specifying
`absent` will delete the file (or directory, if `recurse => true`).

- **group**属于的用户组

Which group should own the file. Argument can be either a group
name or a group ID.

- **links**创建链接文件

How to handle links during file actions. During file copying,
`follow` will copy the target file instead of the link, `manage`
will copy the link itself, and `ignore` will just pass it by.
When not copying, `manage` and `ignore` behave equivalently
(because you cannot really ignore links entirely during local
recursion), and `follow` will manage the file to which the link points.

Valid values are `follow`, `manage`.

- **mode**权限，可以用 421 来定义

The desired permissions mode for the file, in symbolic or numeric
notation. Puppet uses traditional Unix permission schemes and translates
them to equivalent permissions for systems which represent permissions
differently, including Windows.

- **mtime**

A read-only state to check the file mtime.

owner属于哪个用户

The user to whom the file should belong. Argument can be a user name or

a user ID.
path (*namevar*)路径
The path to the file to manage. Must be fully qualified.
- **source**
A source file, which will be copied into place on the local system. Values can be URIs pointing to remote files, or fully qualified paths to files available on the local system (including files on NFS shares or Windows mapped drives). This attribute is mutually exclusive with `content` and `target`
The available URI schemes are *puppet* and *file*. *Puppet* URIs will retrieve files from Puppet's built-in file server, and are usually formatted as:

`puppet:///modules/name_of_module/filename`

This will fetch a file from a module on the puppet master (or from a local module when using puppet apply). Given a `modulepath` of `/etc/puppetlabs/puppet/modules`, the example above would resolve to `/etc/puppetlabs/puppet/modules/name_of_module/files/filename`.
target
The target for creating a link. Currently, symlinks are the only type supported. This attribute is mutually exclusive with `source` and `content`.

service 资源类型

service 主要用于启动、关闭服务等操作

```
[root@puppet-1 manifests]# puppet resource service sshd
service { 'sshd':
  ensure => 'running',
  enable => 'true',
}
```

这里还是比较简单，后面会降到更复杂的用法，简单了解一下常用参数
enable开启是否启动
Whether a service should be enabled to start at boot.
This property behaves quite differently depending on the platform;
wherever possible, it relies on local tools to enable or disable
a given service. Valid values are `true`, `false`, `manual`. Requires
features enableable.
ensure启动还是停止服务

Whether a service should be running. Valid values are `stopped` (also called `false`), `running` (also called `true`).

package 资源类型

package 主要用于安装软件包，通过 yum 或者 apt

```
package {
    [ "vim" , " iproute " , " x?window?system " ] :
    ensure => i n s t a l l e d ;
    [ "pppoe" , "pppoe?conf " ] :
    ensure => absent ;
    }
```

常用参数有

ensure安装、更新、卸载

What state the package should be in. On packaging systems that can retrieve new packages on their own, you can choose which package to retrieve by specifying a version number or `latest` as the ensure value. On packaging systems that manage configuration files separately from "normal" system files, you can uninstall config files by specifying `purged` as the ensure value.

Valid values are `present` (also called `installed`), `absent`, `purged`, `held`, `latest`. Values can match `/./`.

source安装来源

Where to find the actual package. This must be a local file (or on a network file system) or a URL that your specific packaging type understands; Puppet will not retrieve files for you, although you can manage packages as `file` resources.

Providers

aix, appdmg, apple, apt, aptitude, aptrpm, blastwave, dpkg, fink, freebsd, gem, hpux, macports, msi, nim, openbsd, pacman, pip, pkg, pkgdmg, pkgutil, portage, ports, portupgrade, rpm, rug, sun, sunfreeware, up2date, urpmi, yum, zypper

exec 资源类型

调用 shell 执行一条命令，比如删除文件，似乎有人建议少用 exec 资源在不到万不得已的时候不要去用，简单说来 exec 资源就是在执行 puppet 的时候，调用 shell 执行一条 shell 语句，例如：

```
exec {
```

" delete ␣ config " :
path => " / bin : / usr / bin " ,
command => "rm␣ / etc / ssh / ssh_config " ;
}
exec 可以用 path 指定命令执行的预搜索路径，create 属性表明该 exec 将创建一个文件，当下一次 puppet 执行的时候，如果发现了这个文件，就不再执行这个 exec 资源。
exec 资源是不太好掌控的资源，如果能用脚本实现，尽量写成脚本通过 file 资源分发到服务器上面。然后用其他的方式来调用脚本。例如 crontab。说来 crontab 资源，罗嗦一句，虽然 puppet 提供了 crontab 资源，但是你完全可以用 file 资源来把 crontab 任务放到/etc/
cron.d 目录下来实现 crontab 资源的管理。使用 puppet 的时候，尽量用最简单的语法，越是花哨的语法也越容易出错。

第五章. Variables, Conditionals,and Facts

variables 变量

manifest 是支持变量的，可以是一个数组

```
$longthing = "Imagine I have something really long in here. Like an SSH key, let's say."

file {'authorized_keys':
    path => '/root/.ssh/authorized_keys',
    content => $longthing,
}
```

变量申明一定要用$开头，当然 puppet 也集成了一些内部变量，可以通过命令 facter 查看

```
[root@puppet-1 manifests]# facter
fqdn => puppet1.com
hardwareisa => x86_64
hardwaremodel => x86_64
hostname => puppet1
id => rootinterfaces => eth0,eth1,lo
ipaddress => 211.211.211.140
ipaddress_eth0 => 211.211.211.140
```

```
kernelrelease => 2.6.32-358.el6.x86_64
.....省略
```

这些变量是可以直接调用的，例如官方给的一个例子：

```
#/etc/puppet/manifest/motd.pp
file {'motd':
ensure => file,
path => '/etc/motd',
mode => 0644,
content => "This Learning Puppet VM's IP address is ${ipaddress}. It
thinks its
hostname is ${fqdn}, but you might not be able to reach it there
from your host machine. It is running ${operatingsystem}
${operatingsystemrelease} and
Puppet ${puppetversion}.
Web console login:
URL: https://${ipaddress_eth0}
User: puppet@example.com
Password: learningpuppet
",
}
# puppet apply /root/examples/motd.pp
notice: /Stage[main]//Host[puppet]/ensure: created
notice: /Stage[main]//File[motd]/ensure: defined content as
'{md5}bb1a70a2a2ac5ed3cb83e1a8caa0e331'
# cat /etc/motd
This Learning Puppet VM's IP address is 172.16.52.135. It thinks its
hostname is learn.localdomain, but you might not be able to reach it there
from your host machine. It is running CentOS 5.7 and
Puppet 2.7.21 (Puppet Enterprise 2.8.1).
Web console login:
URL: https://172.16.52.135
User: puppet@example.com
Password: learningpuppet
```

这里内部变量有时候会很省事的

Conditional Statements 条件语句

条件预计可以让我们根据不通的情况执行不通的代码

if 条件

```
if condition {
block of code
}
elsif condition {
block of code
}
else {
block of code
}
```

if example：

```
if str2bool("$is_virtual") {
    service {'ntpd':
    ensure => stopped,
    enable => false,
}
}
else {
    service { 'ntpd':
    name => 'ntpd',
    ensure => running,
    enable => true,
    hasrestart => true,
    require => Package['ntp'],
}
}
```

case 语句

直接看例子更好理解：

```
case $operatingsystem {
    centos: { $apache = "httpd" }
# Note that these matches are case-insensitive.
    redhat: { $apache = "httpd" }
    debian: { $apache = "apache2" }
    ubuntu: { $apache = "apache2" }
    default: { fail("Unrecognized operating system for webserver") }
```

```
}
package {'apache':
    name => $apache,
    ensure => latest,
}
```

也可以这样写：

```
case $operatingsystem {
    centos, redhat: { $apache = "httpd" }
    debian, ubuntu: { $apache = "apache2" }
    default: { fail("Unrecognized operating system for webserver") }
}
```

Selectors 条件

直接看例子会更好理解

```
$apache = $operatingsystem ? {
    centos => 'httpd',
    redhat => 'httpd',
    /(?i)(ubuntu|debian)/ => 'apache2',
    default => undef,
}
```

先来看一个 Package/File/Service 例子，不然后面这些 exercise 就不知道怎么弄

```
package { 'openssh-server':
ensure => present,
before => File['/etc/ssh/sshd_config'],
}
file { '/etc/ssh/sshd_config':
ensure => file,
mode => 600,
source => '/root/examples/sshd_config',
}
service { 'sshd':
ensure => running,
enable => true,
subscribe => File['/etc/ssh/sshd_config'],
}
```

Exercise: Build Environment

Use the $operatingsystem fact to write a manifest that installs a C build environment on
Debian-based (“debian,” “ubuntu”) and Enterprise Linux-based (“centos,” “redhat”)
machines. (Both types of system require the gcc package, but Debian-type systems also
require build-essential.)

```
case $operatingsystem {
    centos,redhat:{$apache = “gcc”}
    ununtu,debian:{$apache =  “build-essential”}
    default: { fail("Unrecognized operating system for webserver") }
}
package {‘$apache’:
    ensure => installed,
}
```

Exercise: Simple NTP

Write a manifest that installs and configures NTP for Debian-based and Enterprise Linux-?
based Linux systems. This will be a package/file/service pattern where both kinds of systems?
use the same package name (ntp), but you’ll be shipping different config files (?Debian
version, Red Hat version – remember the file type’s “source” attribute) and using different?
service names (ntp and ntpd, respectively).

```
$ntp = $operatingsystem ?{
    centos => ntpd,
    debian => ntp,
    default => undef,
}
file { '/etc/ssh/sshd_config1':
ensure => file,
mode => 600,
source => '/root/examples/sshd_config1',
}
file { '/etc/ssh/sshd_config2':
ensure => file,
mode => 600,
source => '/root/examples/sshd_config2',
}
if $apache = ntpd {
    package { 'ntpd':
```

```
    ensure => present,
    before => File['/etc/ssh/sshd_config1'],
}
else {
    package { 'ntp':
    ensure => present,
    before => File['/etc/ssh/sshd_config1'],
}
```

第六章.class 类和 modules

Classes are Puppet's way of separating out chunks of code, and modules are Puppet's way of organizing classes so that you can refer to them by name.

classes

简单来说 class 就是定义一段全局 puppet 代码块，供后续调用。
先来看看定义一个 class 的例子：

```
class ntp {
    case $operatingsystem {
    centos, redhat: {
    $service_name = 'ntpd'
    $conf_file = 'ntp.conf.el'
}
    debian, ubuntu: {
    $service_name = 'ntp'
    $conf_file = 'ntp.conf.debian'
}
}
    package { 'ntp':
    ensure => installed,
}
    file { 'ntp.conf':
    path => '/etc/ntp.conf',
    ensure => file,
```

```
    require => Package['ntp'],
    source => "/root/examples/answers/${conf_file}"
}
    service { 'ntp':
    name => $service_name,
    ensure => running,
    enable => true,
    subscribe => File['ntp.conf'],
}
}
```

这里定义一个 class，puppet apply 执行什么都不做，需要用 include 进行调用
在最后加入 **include ntp** 就行了
Classes: Define, then declare

module

modules and the module autoloader.
It works like this:
❶ Modules are just directories with files, arranged in a specific, predictable structure. The manifest files within a module have to obey certain naming restrictions.
❷ Puppet looks for modules in a specific place (or list of places). This set of directories is known as the modulepath, which is a configurable setting
❸ If a class is defined in a module, you can declare that class by name in any manifest. Puppet will automatically find and load the manifest that contains the class definition.

官方的解释更好理解 modules 的作用
创建一个 ntp 的 module：
在 modules 目录下面创建 ntp 目录，在 ntp 目录下创建 manifest，
mkdir /etc/puppet/modules/ntp/manifest
touch /etc/puppet/modules/ntp/manifest/init.pp
将上面的内容拷贝到里面，去掉 include，这里不需要，测试就在 site.pp 里面加入 include ntp 就行，前提是在 puppet.conf 配置文件中定义了 modulepath 的位置

puppet:///

source 里面可以通过 puppet:///来从 puppet 服务器上下载配置文件
mkdir /etc/puppetlabs/puppet/modules/ntp/files

```
# mv /root/examples/answers/ntp.conf.*
/etc/puppetlabs/puppet/modules/ntp/files/
....
file { 'ntp.conf':
path => '/etc/ntp.conf',
ensure => file,
require => Package['ntp'],
source => "puppet:///modules/ntp/${conf_file}",
}
}
....
```

这里是 files 文件的使用，可以 client 端下载配置文件

还有其他的一些特殊的目录：
manifests/ — Contains all of the manifests in the module.
files/ — Contains static files, which managed nodes can download.
templates/ — Contains templates, which can be referenced from the module's manifests. More on templates later.这个比较重要，用的比较多，但难理解
lib/ — Contains plugins, like custom facts and custom resource types.
tests/ or examples/ — Contains example manifests showing how to declare the module's classes and defined types.
spec/ — Contains test files written with rspec-puppet

The Puppet Forge: How to Avoid Writing Modules

官方网站提供了很多的 module，可以直接下载使用，避免自己编写
安装 module：
#puppet module install puppetlabs-mysql
查询安装了哪些 module：
#puppet module list

使用 svn 管理 puppet

svn 简介

Subversion 是一个自由/开源的版本控制系统。也就是说，在 Subversion 管理下，文件和目录可以超越时空。也就是 Subversion 允许你数据恢复到早期版本，或者是检查数据修改的历史。正因为如此，许多人将版本控制系统当作一种神奇的“时间机器”。

Subversion 的版本库可以通过网络访问，从而使用户可以在不同的电脑上进行操作。从某种程度上来说，允许用户在各自的空间里修改和管理同一组数据可以促进团队协作。因为修改不再是单线进行，开发速度会更快。此外，由于所有的工作都已版本化，也就不必担心由于错误的更改而影响软件质量—如果出现不正确的更改，只要撤销那一次更改操作即可。

某些版本控制系统本身也是软件配置管理(SCM)系统，这种系统经过精巧的设计，专门用来管理源代码树，并且具备许多与软件开发有关的特性—比如，对编程语言的支持，或者提供程序构建工具。不过 Subversion 并不是这样的系统。它是一个通用系统，可以管理任何类型的文件集。对你来说，这些文件这可能是源程序—而对别人，则可能是一个货物清单或者是数字电影。

Subversion 的组件

安装好的 Subversion 由几个部分组成，下面将简单的介绍一下这些组件。下文的描述或许过

于简略，不易理解，但不用担心—本书后面的章节中会用更多的内容来详细阐述这些组件。

svn

命令行客户端程序

svnversion

此工具用来显示工作副本的状态(用术语来说，就是当前项目的修订版本)。

svnlook

直接查看 Subversion 版本库的工具

svnadmin

建立, 调整和修复 Subversion 版本库的工具

mod_dav_svn

Apache HTTP 服务器的一个插件，使版本库可以通过网络访问

svnserve

一个单独运行的服务器程序，可以作为守护进程或由 SSH 调用。这是另一种使版本库

可以通过
网络访问的方式。
svndumpfilter
过滤 Subversion 版本库转储数据流的工具
svnsync
一个通过网络增量镜像版本库的程序

svn 安装配置

先安装 epel 源：
rpm -ivh

[root@puppet1 ~]# yum install subversion –y
[root@puppet1 ~]# yum install mod_dav_svn –y
Apache HTTP 服务器的一个插件，使版本库可以通过网络访问
安装 apache，可以支持网络访问：yum install httpd
配置 apache 服务器：
[root@puppet1 puppet]# cat /etc/httpd/conf.d/subversion.conf
LoadModule dav_svn_module modules/mod_dav_svn.so
LoadModule authz_svn_module modules/mod_authz_svn.so
<Location /repos>
DAV svn
SVNParentPath /var/www/svn

Limit write permission to list of valid users.
<LimitExcept GET PROPFIND OPTIONS REPORT>
Require SSL connection for password protection.
SSLRequireSSL

AuthType Basic
AuthName "Authorization Realm"
AuthUserFile /var/www/svn/passwd
Require valid-user
</LimitExcept>
</Location>
删除红色部分的注释，按照文件中的提示操作：
cd /var/www/svn
svnadmin create stuff
chown -R apache.apache stuff
chcon -R -t httpd_sys_content_t stuff
接着配置 svnserver.conf 文件，如果没有配置这个问，会提示无权限访问 staff 目录

将下面 4 行的注释去掉：

```
anon-access = read
 auth-access = write
 password-db = passwd
 authz-db = authz
```

接着就可以启动 httpd 服务了，然后就是导入文件到 staff 版本库了，就可以通过 http 访问了。

apache 设置的认证无效，可能是 authfile 路径写错或者密码未创建

安装 svn 及 puppet 服务器端的脚本

```
 #!/bin/bash
set -x
#执行本脚本必须得联网
#disable selinux and filwall
/etc/init.d/iptables stop
chkconfig iptables off

setenforce 0
sed -i "s/\(SELINUX=\).*/\1disabled/g" /etc/selinux/config

#define global valiables
dir='/var/www/svn'
name='staff'
user='test'
passwd='test'

#install epel sources
rpm -ivh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm

#install http svn puppet
yum -y install httpd httpd-devel --skip-broken
yum -y install ruby puppet-server
yum -y install subversion mod_dav_svn

#create directory
mkdir -p $dir
cd $dir/
svnadmin create $name
chown -R apache:apache $name
```

```
#config apache
cat >> /etc/httpd/conf.d/subversion.conf<<EOF
<Location /svn>
DAV svn
SVNListParentPath on
SVNPath "$dir/$name"
AuthType Basic
AuthName "Subversion repository"
AuthUserFile "$dir/$name/conf/authfile"
Require valid-user
SVNAutoversioning on
ModMimeUsePathInfo on
</Location>
EOF

#config authz
echo "
 [/]
* = r
[$name:/]
test = rw">>$dir/$name/conf/authz

#config  apache auth user:test passwd:test
echo 'test:$apr1$1TF9wHuF$ZidF3M1W7a6y7CN3qxnmn0' > $dir/$name/conf/authfile
echo “the user and passwd are test”
chown apache:apache $dir/$name/conf/authfile

#import file
svn import /etc/puppet file://$dir/$name -m "Initial repository"

#chage svnserve.conf for visualsvn
sed -i 's/# anon-access = read/anon-access = none/g' $dir/$name/conf/svnserve.conf
sed -i 's/# auth-access = write/auth-access = write/g' $dir/$name/conf/svnserve.conf
sed -i 's/# password-db = passwd/password-db = passwd/g' $dir/$name/conf/svnserve.conf
sed -i 's/# authz-db = authz/authz-db = authz/g' $dir/$name/conf/svnserve.conf
sed -i 's/# realm = My First Repository/realm = puppt Repository/g' $dir/$name/conf/svnserve.conf

/etc/init.d/httpd restart
svnserve -d -r $dir/$name/

#auto start
chkconfig httpd on
```

chkconfig puppetmaster on
echo 'svnserve -d -r $dir/$name/' >>/etc/rc.local

svn 基本操作

学会使用 svn help subcommand，svn help import
导入数据到你的版本库
$ svnadmin create /var/svn/newrepos
$ svn import mytree file:///var/svn/newrepos/some/project -m "Initial import"导入
$ svn list 查看
$ svn checkout http://svn.collab.net/repos/svn/trunk subv 导出
基本命令
1. 更新你的工作副本。
• svn update
2. 做出修改
• svn add
• svn delete
• svn copy
• svn move
3. 检验修改
• svn status
• svn diff
4. 可能会取消一些修改
• svn revert
5. 解决冲突(合并别人的修改)
• svn update
• svn resolve
6. 提交你的修改
• svn commit

svn 管理 puppet

第一步.创建一个仓库 puppet
svnadmin create /project/puppet
第二步.导入 puppet 配置文件
svn import /etc/puppet -m “initialization”
第三步.checkout 文件用来修改然后提交到服务器
svn checkout
第四步.修改好后提交到服务器

svn commit –m “2013”

第五步.自动更新到 puppet 目录

svn up /etc/puppet/manifests

svn up /etc/puppet/modules

可以在 hooks 里添加一个脚本来自动触发更新

总结：详细的配置使用方法可以参考官方文档，我个人目前觉得 svn 的用处不大，不过网上很多前辈都在用，或许是我还没有理解其精华的原因吧。

puppet 集群