Puppet 使用手册 V1.0

About Me

ID:Roger

Oracle Senior Consultant

Email:oracledba@live.cn

Blog:www.killdb.com

版本记录

版本编号	版本日期	修改者	说明
Version 1.0	2012-05-05	Roger	创建文档

目录

一、如何启动关闭 puppet？ 2
1）．启动 puppet 2
2）．关闭或重启 puppet....3
二、 Puppet 架构 3
1). /etc/puppet/fileserver.conf....4
2). /etc/puppet/manifests/site.pp....4
3). /etc/puppet/manifests/modules.pp....5
4). /etc/puppet/manifests/nodes/test.pp 5
5). /etc/puppet/modules/snmp/manifests/init.pp....5
三、Puppet 语法 6
1). 资源 6
2). 资源依赖关系 7
3). 类和函数 7
4). 节点 8
5). 模块 10
四、 资源....10
1). file 资源....10
2). package 资源 12
3). exec 资源....12

一、如何启动关闭 puppet？

1）．启动 puppet

启动 puppet 管理端：

```
     /etc/init.d/puppetmaster start 或 service puppetmaster start

 启动 puppet 客户端:

     /etc/init.d/puppet start 或 service puppet start
```

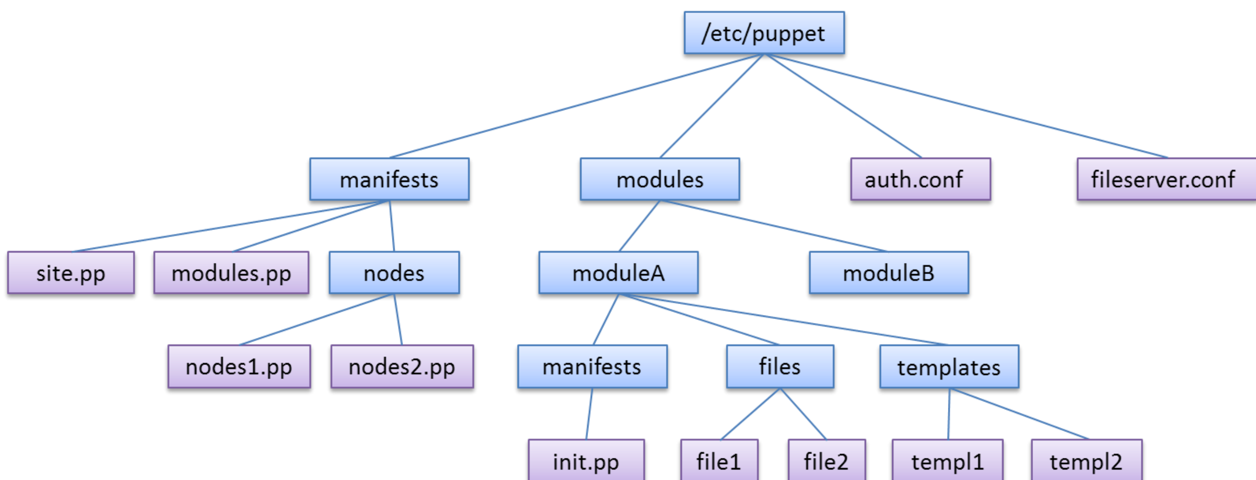
2). 关闭或重启 puppet

```
   停止/重启 puppet 管理端:
    /etc/init.d/puppetmaster stop/restart 或 service puppetmaster stop/restart

   停止/重启 puppet 客户端:

    /etc/init.d/puppet stop/restart 或 service puppet stop/restart
```

二、 Puppet 架构



说明：蓝色为文件夹，淡紫色为文件。

```
manifests/
 |- modules.pp  #模块导入接口，使用 import systembase（模块名）方式导入
 |- node.pp     #节点文件，在节文件里面 include 模块
 |- site.pp     #puppet 主要入口

modules/
 |- systembase  #模块的名称
 |- templates   #模板目录
 |- files       #文件目录
```

```
 |- hosts
 |- vimrc
 | - manifests   #模块主入口
 | - init.pp     #这个文件是必须存在的
```

默认系统 puppet 管理端 tree 结构如下：

```
[root@puppet-master ~]# tree /etc/puppet/
/etc/puppet/
|-- auth.conf
|-- autosign.conf
|-- fileserver.conf
|-- logrotate
|-- manifests
|   |-- site.pp
|   `-- site.pp.bak
`-- puppet.conf
```

如下是一些文件的配置例子：

1). /etc/puppet/fileserver.conf

```
[snmpfiles]
  path /etc/puppet/modules/snmp/files
  allow 192.168.1.0/24
```

2). /etc/puppet/manifests/site.pp

```
# Create "/tmp/testfile" if it doesn't exist.
File {
   owner => root,
   group => root,
   backup => true,
   mode => 0644
}

class test_class {
   file { "/tmp/testfile":
      ensure => present,
      mode   => 600,
      owner  => root,
      group  => root
   }
}

$fileserver = "gw"
```

```
class basepackage {
    package {
        ["sysstat"]:
        ensure => installed;
        ["NetworkManager", "bluez-*"]:
        ensure => purged;
    }
    service {
        ["ssh"]:
        ensure => running;
    }
}

# tell puppet on which client to run the class
node default {

}

import "modules.pp"
import "nodes/*.pp"
```

3). /etc/puppet/manifests/modules.pp

```
#import modules
import "snmp"
```

4). /etc/puppet/manifests/nodes/test.pp

```
node test1 {
    include basepackage
    include snmp
}

node test2 {
    include snmp
    include basepackage
}
```

5). /etc/puppet/modules/snmp/manifests/init.pp

```
class snmp {
    package {
        "net-snmp":
```

```
        ensure => installed;
    }
    file {
        "/etc/snmp/snmpd.conf":
        require => Package["net-snmp"],
        source => "puppet://$fileserver/snmpfiles/snmpd.conf";
    }
    file {
        "/etc/sysconfig/snmpd.options":
        require => Package["net-snmp"],
        mode => 0755,
        source => "puppet://$fileserver/snmpfiles/snmpd.options";
    }
    file {
        "/etc/snmp/iostat.sh":
        require => Package["net-snmp"],
        mode => 0755,
        source => "puppet://$fileserver/snmpfiles/iostat.sh";
    }
    service {
        ["snmpd"]:
        subscribe => File["/etc/snmp/snmpd.conf", "/etc/sysconfig/snmpd.options
"],
        hasrestart => true,
        hasstatus => true,
        ensure => running;
    }
}
```

三、Puppet 语法

1). 资源

定义一个资源，需要指定资源的类型和资源的 title，如下例子：

```
file {                     -----> file 意为这个资源类型为 file
"/etc/passwd" :            -----> title，puppet 用 title 来标示资源的唯一性
name => " /etc/passwd" ,
owner => root ,            -----> 赋予文件 owner 为 root
group => root ,            -----> 属主
mode => 644;               -----> 权限
}
```

2). 资源依赖关系

```
file {
" /etc/apache2/port.conf " :
content => " 80 " ,
require => Package["apache2"] ;
}
package {
"apache2" :
ensure => installed;
}
```

file 资源设置 port.conf 的内容为 80,但是在设置 file 资源之前，要求 apache2 这个软件包配置好了。

3). 类和函数

类的作用是把一组资源收集在一个盒子里面，一起使用，例如把 sshd 和他的配置文件做成一个 ssh 类，其他的地方要用到就直接包含 ssh 类就可以了，方便写出更简洁的代码，便于维护。类可以继承。

例子：

```
calss rsyncfiles {
node 'client1.puppet.com' {
       file
         { "/tmp/test/killdb.sh":
         source => "puppet://puppet-master.puppet.com/tmp/killdb.sh",
         checksum => md5,
         notify => Exec["exec-killdb-createfile"],
         }
       exec
         { "exec-killdb-createfile":
           cwd => "/tmp/test",
           command => "sh /tmp/test/killdb.sh",
           user => "root",
           path => "/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bi
n",
         }
}

node 'client2.puppet.com' {
       file
```

```
          { "/tmp/test/killdb.sh":
          source => "puppet://puppet-master.puppet.com/tmp/killdb.sh",
          checksum => md5,
          notify => Exec["exec-killdb-createfile"],
          }
        exec
          { "exec-killdb-createfile":
            cwd => "/tmp/test",
            command => "sh /tmp/test/killdb.sh",
            user => "root",
            path => "/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bi
n",
          }
}
```

puppet 的官方文档里面是没有 puppet 函数这一说法的，而是叫做 define;这里我写做函数，是因为 define 实现的功能其实和函数一样，而且在 ruby 里面也是用 define 来定义一个函数。这里写做函数，便于理解。

例子：

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

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.

4). 节点

puppet 如何区分不同的客户端，并且给不同的服务端分配 manifest 呢？puppet 使用叫做 node 的语法来做这个事情，node 后面跟客户端的主机名，如下例子：

```

node 'client1.puppet.com' {
        file
          { "/tmp/test/killdb.sh":
          source => "puppet://puppet-master.puppet.com/tmp/killdb.sh",
          checksum => md5,
```

```
         notify => Exec["exec-killdb-createfile"],
         }
       exec
         { "exec-killdb-createfile":
           cwd => "/tmp/test",
           command => "sh /tmp/test/killdb.sh",
           user => "root",
           path => "/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bi
n",
         }
}

node 'client2.puppet.com' {
       file
         { "/tmp/test/killdb.sh":
         source => "puppet://puppet-master.puppet.com/tmp/killdb.sh",
         checksum => md5,
         notify => Exec["exec-killdb-createfile"],
         }
       exec
         { "exec-killdb-createfile":
           cwd => "/tmp/test",
           command => "sh /tmp/test/killdb.sh",
           user => "root",
           path => "/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bi
n",
         }
}

node 'client3.puppet.com' {
       file
         { "/tmp/test/killdb.sh":
         source => "puppet://puppet-master.puppet.com/tmp/killdb.sh",
         checksum => md5,
         notify => Exec["exec-killdb-createfile"],
         }
       exec
         { "exec-killdb-createfile":
           cwd => "/tmp/test",
           command => "sh /tmp/test/killdb.sh",
           user => "root",
           path => "/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bi
n",
         }
}
```

每个节点也可以执行不同的内容，如下例子：

```
node 'host1.example.com' {
include ssh
}
node 'host2.example.com' {
include apache,mysql,php
}
```

当主机 host1.example.com 来连服务端时，只会执行 node'host1.example.com'里面的代码，不会执行 nodehost2.example.com 里面的代码。正如前面所说，可以定义一个 default 结点。比如没有针对 host3 的 node 配置，host3 就用 default 的配置了。在这里 include 的意思是 include 类。

5). 模块

简单来说，一个模块就是一个/etc/puppet/modues 目录下面的一个目录和它的子目录，在 puppet 的主文件 site.pp 里面用 importmodulename 可以插入模块。新版本的 puppet 可以自动插入/etc/puppet/modues 目录下的模块。引入模块，可以结构化代码，便于分享和管理。

例如关于 apache 的所有配置都写到 apache 模块下面。一个模块目录下面通常包括三个目录，files,manifests,templates。

manifests 里面必须要包括一个 init.pp 的文件，这是该模块的初始文件，导入一个模块的时候，会从 init.pp 开始执行。可以把所有的代码都写到 init.pp 里面，也可以分成多个 pp 文件，init 再去包含其他文件。

files 目录是该模块的文件发布目录，puppet 提供一个文件分发机制，类似 rsync 的模块。
templates 目录包含 erb 模型文件，这个和 file 资源的 template 属性有关。

puppet 安装好以后，modules 目录是没有的，需要手工创建，然后在里面可以新增加你的模块。

四、 资源

1). file 资源

管理系统本地文件:
设置文件权限和属主
管理文件内容,可以基于模板的内容管理
支持管理目录

从远程服务器复制整个目录到本地

参数：

backup

决定文件的内容在被修改前是否进行备份.

checksum

怎样检查文件是否被修改,这个状态用来在复制文件的时候使用, 这里有几种检测方式,包括 md5 ,mtime,time,timestamp 等. 默认为 MD5。

content

把文件的内容设置为 content 参数后面的字符串, 新行,tab,空格可用 escaped syntax 表示

ensure

如果文件本来不存在是否要新建文件,可以设置的值是 absent 和 present,file 和 directory. 如果指定 present,就会检查该文件是否存在,如果不存在就新建该文件,如果指定是 absent, 就会删除该文件(如果 recurse => true ,就会删除目录).

force

force 当前的唯一作用是用在把一个目录变成一个链接,可用的值是 ture 和 false

group

指定那个该文件的用户组,值可以是 gid 或者组名

ignore

当用 recursion 方法复制一个目录的时候,可以用 ignore 来设定过滤条件,符合过滤条件的文件不被复制. 使用 ruby 自带的匹配法则.因此 shell 级别的过滤表达式完全支持,例如[a-g]*

links

定义操作符合链接文件. 可以设置的值是 follow 和 manage; 文件拷贝的时候,设置 follow,会拷贝文件的内容,而不是只拷贝符合链接本身,如果设置成 manage ,会拷贝符合链接本身.

mode

mode 用于设置文件的权限

owner

设置文件的属主

path

指定要管理文件的路径,必须用引号引起来, 这也是一个资源的 namevar ,通常 path 等于资源的 title

recurse

设置是否以及如何进行递归操作, 可以设置的值是 false,true ,inf ,remote

recurselimit

递归的深度,设置的值可以匹配/^[0-9]+$/.

replace

是否覆盖已经存在的文件。可以设置的值是(true,yes),(false,no),注,true 和 yes 一样，false 和 no 是一样。

SElrange

文件内容是否属于 SElinux 哪个组成部分，只适于开启了 Selinux 的机器。

SElrole

文件所属的 SeLinux 角色

SEltype

文件所属的 Selinux type

SEluser

文件所属的 Selinux user

source

拷贝一个文件覆盖当前文件,用 checksum 来判断是否有必要进行复制,可以设置的值是一个引用的完整的文件路径,或者是 URI,当前支持的 URI 只有 puppet 和 file

sourceselect

选择拷贝目录级别，默认，source 是递归的。

```
target
```

是为创建链接的。可以设置的值为 notlink.

```
type
```

检查文件是否只读

2). package 资源

package 资源管理系统的软件包安装，该资源的主要属性是 ensure;设置该软件包应该在什么状态. installed 表示要安装该软件,也可以写成 present;absent 表示反安装该软件,pureged 表示干净的移除该软件,latest 表示安装软件包的最新版本。

例子：

```
package {
["vim","iproute","x-window-system"]:
ensure => installed;
["pppoe", "pppoe conf"]:
ensure => absent;
}
```

安装 vim 等包，删除 pppoe,pppoe-conf 包。

3). exec 资源

例子：

```
file { "/etc/aliases":
source => "puppet://server/module/aliases",
}
exec {
newaliases:
path => ["/usr/bin", "/usr/sbin"],
subscribe => File["/etc/aliases"],
refreshonly => true,
}
```

当 aliases 文件配置改变时，后面的 exec 才会执行。

puppet 的 exec 资源提供了大量的参数，可以参考如下说明：

```
command:
```

将会被执行的命令，必须为被执行命令的绝对路径，或者得提供该命令的搜索路径。如果命令被成功执行，所有的输出会被记录在实例的正常（normal）日志里，但是如果命令执行失败（既返回值与我们所指定的不同），那么所有的输出会在错误（err）日志中被记录。

这个是 exec 资源类型的名变量（namevar)。

creates:

指定命令所生成的文件。如果提供了这个参数，那么命令只会在所指定的文件不存在的情况的被执行：

cwd:

指定命令执行的目录。如果目录不存在，则命令执行失败。

env:

我们不建议使用这个参数，请使用‘environment’。这一部分还未完成。

environment

为命令设定额外的环境变量。要注意的是如果你用这个来设定 PATH，那么 PATH 的属性会被覆盖。多个环境变量应该以数组的形式来设定。

group

定义运行命令的用户组。

logoutput

是否记录输出。默认会根据 exec 资源的日志等级（loglevel） 来记录输出。值为：true，false 和其他合法的日志等级。

onlyif

如果这个参数被设定了，则 exec 只会在 onlyif 设定的命令返回 0 时才执行。例如：

```
exec { "logrotate": path => "/usr/bin:/usr/sbin:/bin", onlyif => "test `du /var/log/messages | cut -f1` -gt 100000" }
```

只有在 test 返回 true 的时候 logrotate 才会被运行。

path

命令执行的搜索路径。如果 path 没有被定义，命令需要使用绝对路径。路径可以以数组或以冒号分隔的形式来定义。

refresh

定义如何更新命令。当 exec 收到一个来自其他资源的事件时，默认只会重新执行一次命令。不过这个参数允许你定义更新时执行不同的命令。

refreshonly

该属性可以使命令变成仅刷新触发的，也就是说只有在一个依赖的对象被改变时，命令才会被执行。仅当命令与其他对象有依赖关系时，这个参数才有意义。

要注意的是只有 subscribe 和 notify 可以促发行为，而不是 require，所以在使用 refreshonly 时，只有同时使用 subscribe 或 notify 才有意义。有效的值为 true, false。

returns

指定返回的代码。如果被执行的命令返回了其他的代码，一个错误（error）会被返回。默认值是 0，可以定义为一个由可以接受的返回代码组成的数组或单值。

timeout

命令运行的最长时间。如果命令运行的时间超过了 timeout 定义的时间，那么这个命令就会被终止，并作为运行失败处理。当定义为负值时就会取消运行时间的限制。timeout 的值是以秒为单位的。

unless

如果这个变量被指定了，那么 exec 会执行，除非 unless 所设定的命令返回 0。

user

定义运行命令的用户。