КОМПЬЮТЕРНЫЕ ИССЛЕДОВАНИЯ
И МОДЕЛИРОВАНИЕ 2015 Т. 7 № 3 С. 735–740

Ки&М

СЕКЦИОННЫЕ ДОКЛАДЫ

УДК: 004.75, 004.45

Опыт использования puppet для управления вычислительным грид-кластером Tier-1 в НИЦ «Курчатовский институт»

И. А. Ткаченко

Национальный исследовательский центр «Курчатовский институт»,
Россия, 123182, г. Москва, пл. Академика Курчатова, д. 1
E-mail: tia@grid.kiae.ru

Получено 4 декабря 2014 г.

Статья посвящена организации системы управления кластером при помощи puppet. Рассматриваются вопросы: безопасности использования, с точки зрения массового применения к вычислительному кластеру неверной конфигурации (в виду человеческого фактора); организации совместной работы и создания для каждого администратора возможности, независимо от других, написания и отладки собственных сценариев, до включения их в общую систему управления; написания сценариев, которые позволят получить как целиком настроенный узел, так и обновлять конфигурацию по частям, не затрагивая остальные компоненты, независимо от текущего состояния узла вычислительного кластера.

Сравниваются различные подходы к созданию иерархии puppet сценариев: описываются проблемы, связанные с использованием «include» для организации иерархии и переход к системе последовательного вызова классов через shell-скрипт.

Ключевые слова: puppet, автоматизация настройки, совместное управление кластером, варианты использования puppet

Experience of puppet usage for managment of Tier-1 GRID cluster at NRC “Kurchatov Institute”

I. A. Tkachenko

National Research Center “Kurchatov Institute”, 1 Kurchatov Sq., Moscow, 123182, Russia

This article is about the organization of the cluster management using puppet. It tells about: safety of usage, from the point of view of mass apply at a computing cluster wrong configuration (by reason of human factor); collaboration work and the creation of opportunities for each cluster administrator, regardless of others, writing and debugging your own scripts, before include them in the overall system of cluster managment; writing scripts, which allow to get as fully configured nodes, and updates the configuration of any system parts, without affecting the rest of the nodes components, regardless of the current state of the node of computing cluster.

The article compares different methods of the creation of the hierarchy of puppet scenarios, describes problems associated with the use of “include” for the organization hierarchy, and tells about the transition to a system of sequential call classes through shell-script.

Keywords: puppet, automation of configuration, joint cluster management, puppet use cases

Вычисления выполнялись на компьютерных ресурсах ЦКП «Комплекс моделирования и обработки данных исследовательских установок мегакласса».

Citation: *Computer Research and Modeling*, 2015, vol. 7, no. 3, pp. 735–740 (Russian).

Любой, кто занимается администрированием более двух одинаково настраиваемых серверов, рано или поздно начинает смотреть на системы, позволяющие автоматизировать процесс настройки. В НИЦ «Курчатовский институт» для этого используется puppet. В этой статье будет рассмотрен опыт его применения на большом количестве серверов (в данный момент управляемая инфраструктура — около 120 Linux серверов (Centos 5 и Centos 6), которые должны работать в режиме 24/7/365.

Автоматизации иногда бывает слишком много

Puppet [Puppetlabs, Web site] удобен тем, что все сценарии хранятся на центральном сервере и автоматически распространяются на клиентские машины. Это означает, что любая пропущенная ошибка за конечное время распространится на все серверы. Puppet предлагает в принципе готовое решение — environment, с помощью которого можно задать поведение клиента для разных случаев. Но это означает, что нужно отдельное хранилище для тестируемых правок — раз, отдельный puppet-master — два, дополнительный рестрат puppet-клиента (с новым environment) — три.

Также существенно усложняется работа системных администраторов при отладке сценариев и написании новых манифестов: каждый системный администратор должен работать в собственном environment и не забывать переключать клиентский puppet-сервер на него и обратно — на основной.

Чтобы избежать этих проблем, можно воспользоваться системой ручного запуска манифестов. Для этого манифесты должны находиться на локальной машине и вызов осуществляется через puppet apply. Это порождает новую проблему, которая раньше решалась puppet автоматически: синхронизация наборов сценариев на управляющем узле (его еще называют master) и клиентах.

Для решения проблемы синхронизации был разработан специальный манифест:

```
class sync inherits site_settings {
  File { owner => 'root', group => 'root', mode  => 0600 }
  $puppet_conf_dir = $site_settings::puppet
  $target_path = $environment ? {
    'production' => 'modules',
    default      => "environments/$environment/modules"
  }

  file { "$puppet_conf_dir/environments":
    ensure => 'directory',
  }

  file { "$puppet_conf_dir/environments/$environment":
    ensure => 'directory',
  }

  each($site_settings::admins) |$admin| {
    if (!defined(File["$puppet_conf_dir/environments/$admin"]))
{

      file { "$puppet_conf_dir/environments/$admin":
        ensure => 'directory',
      }
    }
  }
```

```
  file { "$puppet_conf_dir/$target_path":
      source => [        "pup-
pet://$site_settings::master/private/$site_settings::host_type/
puppet",        "puppet://$site_settings::master/files/",
"puppet://$site_settings::master/puppet/",
      ],
      ensure => 'directory',
      recurse => true,
      purge => true,
      force => true,
      ignore => ['.svn','.*.swp'],
      sourceselect => 'all',
  }

  file { "$puppet_conf_dir/puppet.conf":
    source => "pup-
pet://$site_settings::master/sysconf/puppet/puppet.conf",
    ensure => 'file',
  }
}
```

Поскольку не используется автоматическое распространение изменений на клиентские машины, правки можно делать прямо на мастере, синхронизировать один узел и тестироваться на нем. Затем, например через pdsh, синхронизировать остальные серверы и, опять же через pdsh, запускать нужные классы на всех серверах через `puppet apply -e "include some_class"` [Puppet apply…, Web site]. Также существенно упрощается одновременная работа: тестируемые правки не смешиваются с рабочей версией и окружения для разных администраторов полностью изолированы. Это позволяет избежать таких ошибок, как, например, пропущенная синхронизация для отмены изменений — puppet, при запуске без указания environment будет работать со стабильной рабочей версией.

Установка и перенастройка — одно и то же

Второе, с чем пришлось столкнуться, — первичная установка сервера.
При правильно написанных сценариях большую часть установки можно выполнить на этапе разливки узла, через `%post kikstart` файлов [Anaconda…, Web site].
Чтобы каждый раз не думать, какие классы должны исполняться для настройки узла, разумно вызывать всего один — `install`, который «подумает» за нас примерно таким образом:

```
class install inherits site_settings {
  if $site_settings::mlist {
    $scripts_dir=$site_settings::scripts_dir
    $list="$scripts_dir/install_list.sh"
    $mlist=$site_settings::mlist

    File {owner=> 'root', group => 'root', mode => 0700}
    file {"$scripts_dir":
      ensure => 'directory'
    } ->
    file {"$list":
      content => inline_template("#!/bin/sh -x\n<% mlist.each
do |val| -%>puppet apply -e 'include <%= val %>::install' &&
```

```
\\\n<% end -%>exit 0 || exit 1"),
    } ->
    exec {"$list":
      provider => 'shell',
      logoutput => true,
      timeout => 0,
    }
  }
}
```

Сам же mlist определяется на основе типа узла из двух частей, общей для всех узлов и специфичной для конкретного типа узла:

```
case $fqdn {
  ...
  /^io.*/ :                    { $group='eos-mgm' }
  ...
  /^edg[12].*/ :               { $group='eos-dsi' }
  'somehost':                 { $group='eos-fst' }
  /^sdns.*/ :                  { $group='dcache-namespace' }
  default:                     { $group='test' }
}

...

$mlist_default = ['hosts','dns', 'modules', 'sudo', 'ntp',
'mail','firstboot', 'firewall', 'ssh', 'yum', 'nagios',
'staff', 'logwatch']

...

$mlist_addons = $group ? {
  ...

  'nagios' => ['x509', 'nagios'],

  ...

  default     => ['']
}

$mlist = split(inline_template("<%=
(@mlist_default+@mlist_addons).join(',') %>"),',')
```

Таким образом, как только мы захотели получить настроенный узел, `puppet apply -e 'include install'` сделает это для нас в любой момент времени и из любого места. install-подкласс в каждом классе описывает, то в каком порядке какие подклассы нужно подключать, но об этом ниже. Таким образом, например, настройка ssh — это `puppet apply -e 'include ssh::install'`

Порядок прежде всего

Puppet устроен так, что порядок выполнения команд задается набором зависимостей, а не так, в каком порядке директивы следуют в тексте манифеста [Puppet resource…]. Это одновременно удобно и неудобно. С одной стороны, можно легко задавать сложные зависимости и условия, а с другой — проблемы возникают в самых неожиданных местах.

Например:

```
incldue class1
include class2
include class3
```

в большинстве случаев подключит все в правильном порядке. Но как только `class2` у нас сам делает `include`, например, вот так:

```
include class2.1
include class4
```

говорить о том, что `class4` отработает до `class3`, нельзя. Не спасает даже подключение классов как объектов:

```
Class{"calss1": } -> Class{"clas2":}-> Class{"class3":}
```

`class2` должен выполнится до `class3`, но условия причинности на все подключаемые внутри класса подклассы не распространяются.

Чтобы исключить подобные ситуации, проще всего от последовательного подключения классов перейти к созданию и запуску shell-скриптов.

Как было отмечено выше, подкласс `install` отвечает за правильный порядок выполнения манифеста.

На примере того же ssh:

```
class ssh inherits site_settings {
  File { owner => 'root', group => 'root' }
  $install_list=['sshd', 'keys::install']
  $users=[
    'root',
    't1monitor',
  ]
}

define apply::by_list($list, $sub_class='none') {
  if ( $sub_class == 'none' ) {
    $script="$site_settings::scripts_dir/
      ${caller_module_name}"
    $m_name=$caller_module_name
  } else {
    $script="$site_settings::scripts_dir/
      ${caller_module_name}_${sub_class}"
    $m_name="${caller_module_name}::${sub_class}"
  }
  File {mode => 0700, owner => 'root', group =>'root'}

  file {"$site_settings::scripts_dir":
    ensure => 'directory'
  } ->

  file {"$script":
    content => inline_template ("#!/bin/sh -x \n<% list.each do
|val| -%>puppet apply -e 'include <% if val == @m_name then
%><%= @m_name %><% else %><%= @m_name %>::<%= val %><% end %>'
&& \\\n<% end -%>exit 0 || exit 1")
  } ->
```

```
    exec {"$title":
      command => "$script",
      provider => 'shell',
      logoutput => true,
      timeout => 0,
    }
  }
```

Здесь вместо прямого подключения классов формируется скрипт, последовательно вызывающий `puppet apply` для всех нужных классов.

Такой подход позволяет игнорировать мнение puppet о том, в каком порядке подключать классы, и делать это в том порядке, в котором это нужно нам.

Кроме этого, это позволяет разбивать манифесты на смысловые части и, например, не обновлять rpm, когда нужно только обновить конфигурационные файлы.

Последние версии puppet начали поддерживать директиву `contain` [Relationships…], которая говорит puppet, что в этом классе содержатся вызовы других классов и к ним нужно применить все условия и зависимости, которые были определены для «родительского» класса.

Такой отказ от излишней, в нашем случае, автоматизации puppet позволяет без потери функционала существенно повысить удобство администрирования вычислительного комплекса и избегать многих ошибок.

Список литературы

Anaconda, Kickstart. Web site. URL: http://fedoraproject.org/wiki/Anaconda/Kickstart. 2015.

Puppetlabs. Web site. URL: http://puppetlabs.com. 2015.

Puppet apply man page. Web site. URL: https://docs.puppetlabs.com/references/3.7.0/man/apply.html. 2011.

Puppet resource ordering. Web site. URL: https://docs.puppetlabs.com/learning/ordering.html. 2015.

Relationships and Ordering. Web site.
URL: https://docs.puppetlabs.com/puppet/latest/reference/lang_relationships.html. 2015.