Red Hat Satellite 6.10

Puppet 指南

构建您自己的 Puppet 模块并将其导入到 Satellite 6 的指南

Last Updated: 2024-01-31

Red Hat Satellite 6.10 Puppet 指南

构建您自己的 Puppet 模块并将其导入到 Satellite 6 的指南

Red Hat Satellite Documentation Team
satellite-doc-list@redhat.com

法律通告

摘要

Puppet 是红帽卫星 6 中使用的系统配置工具。本书通过创建基本的 Puppet 模块和运行，以及如何在红帽卫星 6 基础架构中使用此模块。

目录

第 1 章 概述

Puppet 是一款用于应用和管理系统配置的工具。Puppet 收集系统信息或事实，并使用此信息通过一组模块创建自定义系统配置。这些模块包含参数、条件参数、操作和模板。Puppet 用作本地系统命令行工具或客户端-服务器关系，其中服务器充当 Puppet 宿主，并使用 Puppet 代理将配置应用到多个客户端系统。这提供了一种方式，可以分别自动配置新调配的系统，也可以同时创建特定的基础架构。

1.1. 定义 PUPPET 工作流

Puppet 使用以下工作流，将配置应用到系统。

1. 收集有关各个系统的事实。这些事实包括硬件、操作系统、软件包版本和其他信息。各个系统上的 Puppet 代理收集此信息，并将它发送到 Puppet 宿主。
2. Puppet 宿主为每个系统生成自定义配置，并将其发送到 Puppet 代理。此自定义配置称为目录。
3. Puppet 代理将配置应用到系统。
4. Puppet 代理将报告发回到 Puppet 宿主，后者指示应用的更改以及任何更改都失败。
5. 第三方应用程序可以使用 Puppet 的 API 收集这些报告。

1.2. 在 SATELLITE 6 上使用 PUPPET

Red Hat Satellite 6 以多种方式使用 Puppet：

- 红帽卫星 6 导入用于定义系统配置的 Puppet 模块。这包括对模块版本及其环境的控制。
- Red Hat Satellite 6 从 Puppet 模块导入一组参数，也称为 Puppet 智能类参数。用户可以接受 Puppet 类的默认值，或在全局级别或系统特定级别上提供自己的默认值。
- 红帽卫星 6 触发了各个系统上主服务器与相应代理之间的 Puppet 执行。Puppet 运行可能会发生：
  - 比如在置备过程完成后或作为守护进程自动检查并管理机器的生命周期中的配置。
  - 手动，例如管理员需要触发即时 Puppet 运行的时间。
- 在配置工作流完成后，卫星 6 从 Puppet 收集报告。这有助于长期审核和归档系统配置。

这些功能为用户提供了使用 Puppet 控制应用程序生命周期的系统配置方面的简单方法。

如果需要，若要查找卫星环境中使用的 Puppet 版本，请查看 软件包清单。

1.3. SATELLITE 6 上的 PUPPET 性能和可扩展性

卫星 6 中 Puppet 的性能会影响卫星和胶囊存储容量、CPU 性能和可用内存，而非应用限制。因此，测试您的硬件和配置是确保性能可为您接受的唯一方法。

有关推荐的存储和系统要求的详情，请参考 *从 连接的网络安装 Satellite 中的准备您的环境*。

第 2 章 从调度构建 PUPPET 模块

本章介绍了如何构建和测试您自己的 Puppet 模块。这包括创建部署简单 Web 服务器配置的 Puppet 模块的基本教程。

2.1. 检查 PUPPET 模块的 ANATOMY

在创建模块之前，我们需要了解创建 Puppet 模块的组件。

Puppet 清单

不要与清单 （订阅清单） 混淆。Puppet 清单是包含用于定义一组资源及其属性的代码的文件。资源是系统的任何可配置部分。资源示例包括软件包、服务、文件、用户和组、SELinux 配置、SSH 密钥身份验证、cron 作业等。清单使用一组键值对为其属性定义各个所需的资源。例如：

```
package { 'httpd':
  ensure => installed,
}
```

此声明检查是否安装了 **httpd** 软件包。如果没有，清单将执行 **yum** 并安装它。

用于开始编译的 Puppet 清单称为"维护清单"或"站点清单"。

清单位于模块的 **清单目录中**。

Puppet 模块也使用 **测试** 目录来测试清单。这些清单用于测试官方清单中包含的特定类。

静态文件

模块可以包含 Puppet 可以复制到您系统上的特定位置的静态文件。这些位置和其他属性（如权限）通过清单中的 **文件** 资源声明进行定义。
静态文件位于模块 **的文件** 目录中。

模板

有时配置文件需要自定义内容。在这种情况下，用户可以创建模板而不是静态文件。与静态文件一样，模板在清单中定义，复制到系统上的位置。不同之处在于，模板允许 Ruby 表达式定义自定义内容和变量输入。例如，如果要使用可自定义端口配置 **httpd**，那么配置文件的模板会包括：

```
Listen <%= @httpd_port %>
```

本例中的 **httpd_port** 变量在引用此模板的清单中定义。

模板位于模块的 **templates** 目录中。

插件

插件允许在 Puppet 的核心功能之外扩展方面。您可以使用插件来定义自定义事实、自定义资源或新功能。例如，数据库管理员可能需要 PostgreSQL 数据库的资源类型。这有助于数据库管理员在安装 PostgreSQL 后使用一组新数据库填充 PostgreSQL。因此，数据库管理员需要创建一个 Puppet 清单，以确保 PostgreSQL 安装和数据库之后创建。
插件位于模块的 **lib** 目录中。其中包括一组子目录，具体取决于插件类型。例如：

- **/lib/facter** - 自定义事实的位置。

- **/lib/puppet/type** - 自定义资源类型定义的位置，概述了属性的键值对。
- **/lib/puppet/provider** - 自定义资源提供程序的位置，与资源类型定义一同使用来控制资源。
- **/lib/puppet/parser/functions** - 自定义功能的位置。

2.2. 设置 PUPPET 开发系统

Puppet 开发系统可用于创建和测试您自己的模块。

流程

使用此流程部署新的 Puppet 开发系统。

1. 部署新的 Red Hat Enterprise Linux 7 系统，并将该系统注册到红帽 CDN 或 Satellite 服务器。
2. 为 Puppet 5 启用 Red Hat Satellite Tools 6.10 软件仓库：

```
# subscription-manager repos \
--enable=rhel-7-server-satellite-tools-6.10-rpms
```

3. 安装 Puppet 代理：

```
# yum install puppet
```

4. 安装 Puppet Development Kit：

```
# yum install pdk
```

2.3. 生成新模块 BOILERPLATE

创建新模块的第一步是更改到 Puppet 模块目录并创建基本的模块结构。手动创建此结构，或使用 Puppet Development Kit 为您的模块创建样板：

```
# cd /etc/puppetlabs/code/modules
# pdk new module user_name-module_name
```

此时会出现交互式向导，指导您使用元数据填充模块的 **metadata.json** 文件。如需更多信息，请参阅 *Puppet Development Kit* 文档中的 创建一个模块。

模块生成过程完成后，新模块包含一些基本文件，包括 **清单目录**。此目录已经包含一个名为 **init.pp** 的清单文件，它是模块的主清单文件。查看文件以查看该模块的空类声明：

```
class mymodule {


}
```

该模块还包含 **示例** 目录，其中包含一个名为 **init.pp** 的清单。此测试清单包含对 **manifests/init.pp** 中的 **mymodule** 类的引用：

```
include::mymodule
```

Puppet 将使用此测试清单来测试模块。

现在，我们已准备好将系统配置添加到我们的模块中。

2.4. 安装 HTTP 服务器

我们的 Puppet 模块将安装运行 HTTP 服务器所需的软件包。这需要一个资源定义来为 **httpd** 软件包定义配置。

在模块的清单目录中， 创建一个名为 **httpd.pp** 的新清单文件：

```
# touch mymodule/manifests/httpd.pp
```

此清单将包含我们模块的所有 HTTP 配置。为了进行组织，我们将将此清单与 **init.pp** 清单分开。

在新的 **httpd.pp** 清单中添加以下内容：

```
class mymodule::httpd {
  package { 'httpd':
    ensure => installed,
  }
}
```

此代码定义了名为 **httpd** 的 **mymodule** 的子类，然后定义 **httpd** 软件包的软件包资源声明。**ensure latex installed** 属性告知 Puppet 检查是否安装了软件包。如果没有安装，Puppet 将执行 **yum** 来安装它。

我们还需要将这个子类包含在我们的主清单文件中。编辑 **init.pp** 清单：

```
class mymodule {
  include mymodule::httpd
}
```

现在需要测试模块。运行以下命令：

```
# puppet apply mymodule/examples/init.pp --noop
...
Notice: /Stage[main]/Mymodule::Httpd/Package[httpd]/ensure: current_value absent, should be
present (noop)
...
```

此输出通知消息是 **ensure InventoryService installed 属性** 的结果。**current_value absent** 表示 Puppet 已经检测到没有安装 **httpd** 软件包。如果没有 **--noop** 选项，Puppet 将安装 **httpd** 软件包。

puppet apply 命令将清单中的配置应用到您的系统。我们使用 test **init.pp** 清单，它引用 **init.pp** 清单。**--noop** 对配置执行空运行，它只显示输出，但不实际应用配置。

2.5. 运行 HTTP 服务器

安装 **httpd** 软件包后，我们使用另一个资源声明 **服务** 启动该服务。

编辑 **httpd.pp** 清单并添加突出显示的行：

```
class mymodule::httpd {
  package { 'httpd':
    ensure => installed,
  }
  service { 'httpd':
    ensure => running,
    enable => true,
    require => Package["httpd"],
  }
}
```

这可实现几个问题：

- **ensure InventoryService running** 属性检查该服务是否正在运行。如果没有，Puppet 会启动它。
- **enable Infoblox true** 属性将服务设置为在系统引导时运行。
- **require abrt Package["httpd"]** 属性定义一个资源声明与其他资源声明之间的顺序关系。在本例中，它会确保在 **httpd** 软件包安装之后启动 **httpd** 服务。这会在服务及其相应软件包之间创建一个依赖项。

再次运行 **puppet apply** 命令，以测试对模块的更改：

```
# puppet apply mymodule/tests/init.pp --noop
...
Notice: /Stage[main]/Mymodule::Httpd/Service[httpd]/ensure: current_value stopped, should be
running (noop)
...
```

此输出通知消息是 **httpd** 服务的新资源定义的结果。

2.6. 配置 HTTP 服务器

HTTP 服务器现已安装并启用。下一步是提供一些配置。HTTP 服务器已在 **/etc/httpd/conf/httpd.conf** 中提供一些默认配置，它在端口 80 上提供一个 Web 服务器。我们将添加一些额外的配置，以便在用户指定的端口上提供额外的 Web 服务器。

我们使用模板文件来存储配置内容，因为用户定义的端口需要变量输入。在我们的模块中，创建一个名为 **templates** 的目录，并在新目录中添加名为 **myserver.conf.erb** 的文件。在文件中添加以下内容：

```
Listen <%= @httpd_port %>
NameVirtualHost *:<%= @httpd_port %>
<VirtualHost *:<%= @httpd_port %>>
  DocumentRoot /var/www/myserver/
  ServerName <%= @fqdn %>
  <Directory "/var/www/myserver/">
    Options All Indexes FollowSymLinks
    Order allow,deny
    Allow from all
  </Directory>
</VirtualHost>
```

此模板遵循 Apache Web 服务器配置的标准语法。唯一的区别是包含 Ruby 转义字符来注入我们模块的变量。例如，**httpd_port**，我们用来指定 Web 服务器端口。

另请注意包含 **fqdn** 的变量，它是存储系统的完全限定域名的变量。这称为系统事实。系统事实是在生成各个系统的 Puppet 目录之前从每个系统收集而来。Puppet 使用 **facter** 命令收集这些系统事实，您也可以运行事实来查看这些事实的列表。

编辑 **httpd.pp** 清单并添加突出显示的行：

```
class mymodule::httpd {
  package { 'httpd':
    ensure => installed,
  }
  service { 'httpd':
    ensure => running,
    enable => true,
    require => Package["httpd"],
  }
  file {'/etc/httpd/conf.d/myserver.conf':
  notify => Service["httpd"],
    ensure => file,
    require => Package["httpd"],
    content => template("mymodule/myserver.conf.erb"),
  }
  file { "/var/www/myserver":
    ensure => "directory",
  }
}
```

这可实现：

- 我们为服务器配置文件 **/etc/httpd/conf.d/myserver.conf** 添加文件资源声明。
- 我们使用 **notify abrt Service[" httpd "] 属性在配置文件和 httpd** 服务之间添加一个关系。这会检查配置文件是否有更改。如果文件已更改，Puppet 将重新启动服务。
- 在添加此文件之前，我们检查是否安装了 **httpd** 软件包。
- 此 **/etc/httpd/conf.d/myserver.conf** 文件的内容是我们之前创建的 **myserver.conf.erb** 模板。
- **我们添加第二个 file resource 声明。这个目录为我们的 Web 服务器创建一个目录 /var/www/myserver/。**

我们还需要将 httpd_port 参数包含在我们的主清单文件中。编辑 init.pp 清单，添加以下文本，以粗体显示：

```
class mymodule (
  $httpd_port = 8120
) {
  include mymodule::httpd
}
```

这会将 **httpd_port** 参数设置为默认值 **8120**。您可以使用 **Satellite** 服务器覆盖这个值。

再次运行 **puppet apply** 命令，以测试对模块的更改：

```
# puppet apply mymodule/tests/init.pp --noop
...
Notice: /Stage[main]/Mymodule::Httpd/File[/var/www/myserver]/ensure: current_value absent, should
be directory (noop)
...
Notice: /Stage[main]/Mymodule::Httpd/File[/etc/httpd/conf.d/myserver.conf]/ensure: current_value
absent, should be file (noop)
...
```

这些输出会看到消息显示创建配置文件和 **Web** 服务器目录。

2.7. 配置防火墙

Web 服务器需要一个开放端口，以便用户可以访问在我们的 **Web** 服务器上托管的页面。开放问题是，不同版本的 **Red Hat Enterprise Linux** 使用不同的方法来控制防火墙。对于 **Red Hat Enterprise Linux 6** 及以下，我们使用 **iptables**。对于 **Red Hat Enterprise Linux 7**，我们使用 **firewalld**。

这一决策是使用条件逻辑和系统事实来处理的 **Puppet**。对于此步骤，我们添加一个语句来检查操作系统并运行相应的防火墙命令。

在您的 **mymodule::httpd** 类中添加以下代码：

```
  if versioncmp($::operatingsystemmajrelease, '6') <= 0 {
    exec { 'iptables':
      command => "iptables -I INPUT 1 -p tcp -m multiport --ports ${httpd_port} -m comment --
comment 'Custom HTTP Web Host' -j ACCEPT &amp;amp;&amp;amp; iptables-save > /etc/sysconfig/iptables",
      path => "/sbin",
      refreshonly => true,
      subscribe => Package['httpd'],
    }
    service { 'iptables':
      ensure => running,
      enable => true,
      hasrestart => true,
      subscribe => Exec['iptables'],
    }
  }
  elsif $operatingsystemmajrelease == 7 {
    exec { 'firewall-cmd':
```

```
    command => "firewall-cmd --zone=public --add-port=${httpd_port}/tcp --permanent",
    path => "/usr/bin/",
    refreshonly => true,
    subscribe => Package['httpd'],
  }
  service { 'firewalld':
    ensure => running,
    enable => true,
    hasrestart => true,
    subscribe => Exec['firewall-cmd'],
  }
}
```

这个代码执行以下操作：

- 使用 操作系统**majrelease** 事实来确定操作系统是否为 **Red Hat Enterprise Linux 6** 还是 **7**。
- 如果使用 **Red Hat Enterprise Linux 6**，则声明一个运行 **iptables** 和 **iptables-save** 的可执行（执行）资源来添加永久防火墙规则。**httpd_port** 变量是内嵌的，以定义要打开的端口。在 **exec** 资源完成后，我们触发 **iptables** 服务的刷新。为此，我们定义了一个服务资源，其中包含 **subscribe** 属性。此属性检查是否有对另一个资源的更改，如果是，则执行刷新。在这种情况下，它会检查 **iptables** 可执行文件资源。
- 如果使用 **Red Hat Enterprise Linux 7**，声明了运行 **firewall-cmd** 的类似可执行文件资源来添加持久防火墙规则。此外，也使用 **httpd_port** 变量来定义要打开的端口。在 **exec** 资源完成后，我们触发 **firewalld** 服务的刷新，但使用 **subscribe** 属性指向 **firewall-cmd** 可执行资源。
- 两个防火墙可执行文件的代码包含 刷新只刷新的 **10.10.10.2 true**，并订阅 **abrt Package['httpd']** 属性。这样可保证防火墙命令仅在 **httpd** 安装之后运行。如果没有这些属性，后续运行将添加同一防火墙规则的多个实例。

再次运行 **puppet apply** 命令，以测试对模块的更改。以下示例是 **Red Hat Enterprise Linux 6** 的测试：

```
# puppet apply mymodule/tests/init.pp --noop
...
Notice: /Stage[main]/Mymodule::Httpd/Exec[iptables]/returns: current_value notrun, should be 0
(noop)
...
Notice: /Stage[main]/Mymodule::Httpd/Service[iptables]: Would have triggered 'refresh' from 1 events
...
```

这些输出通知消息显示防火墙规则创建和执行，以及后续服务刷新（subscription 属性）。

重要

此配置仅充当使用条件语句的示例。如果您以后为您的系统管理多个防火墙规则，建议为防火墙创建自定义资源。使用可执行资源持续链很多 Bash 命令不足。

2.8. 配置 SELINUX

SELinux 默认限制对 HTTP 服务器的非标准访问。如果我们定义了自定义端口，则需要添加允许 SELinux 授予访问权限的配置。

Puppet 包含用于管理某些 SELinux 功能的资源类型，如布尔值和模块。但是，我们需要执行 semanage 命令来管理端口设置。这个工具是 policycoreutils-python 软件包的一部分，默认情况下未在 Red Hat Enterprise Linux 系统上安装。

在您的 mymodule::httpd 类中添加以下代码：

```
exec { 'semanage-port':
  command => "semanage port -a -t http_port_t -p tcp ${httpd_port}",
  path => "/usr/sbin",
  require => Package['policycoreutils-python'],
  before => Service['httpd'],
  subscribe => Package['httpd'],
  refreshonly => true,
}
package { 'policycoreutils-python':
  ensure => installed,
}
```

这个代码执行以下操作：

- require 时间为 package['policycoreutils-python'] 属性，确保在执行此命令前确保已安装了 policycoreutils-python。
- Puppet 使用 httpd_port 作为变量执行 semanage，将自定义端口添加到 Apache 允许侦听的 TCP 端口列表中。
-

之前 InventoryService Service ['httpd'] 确保在 httpd 服务启动前执行此命令。如果在 SELinux 命令前启动 httpd，SELinux 会拒绝访问端口，服务无法启动。

- SELinux 可执行文件的代码包含 仅刷新 10.10.10.2 true 并订阅 abrt Package['httpd'] 属性。这样可确保 SELinux 命令仅在 httpd 安装之后运行。如果没有这些属性，后续运行会失败。这是因为 SELinux 检测到端口已经启用并报告错误。

再次运行 puppet apply 命令，以测试对模块的更改。

```
# puppet apply mymodule/tests/init.pp --noop
...
Notice: /Stage[main]/Mymodule::Httpd/Package[policycoreutils-python]/ensure: current_value absent, should be present (noop)
...
Notice: /Stage[main]/Mymodule::Httpd/Exec[semanage-port]/returns: current_value notrun, should be 0 (noop)
...
Notice: /Stage[main]/Mymodule::Httpd/Service[httpd]/ensure: current_value stopped, should be running (noop)
...
```

Puppet 首先安装 policycoreutils-python，然后在启动 httpd 服务前配置端口访问权限。

2.9. 将 HTML 文件复制到 WEB 主机

HTTP 服务器配置现已完成。这为安装基于 Web 的应用程序提供了一个平台，供 Puppet 进行配置。但在本示例中，我们将仅将一个简单索引网页复制到我们的网页上。

在文件目录中创建一个名为 index.html 的文件。在该文件中添加以下内容：

```
<html>
  <head>
    <title>Congratulations</title>
  <head>
  <body>
    <h1>Congratulations</h1>
    <p>Your puppet module has correctly applied your configuration.</p>
  </body>
</html>
```

在 manifests 目录中创建一个名为 app.pp 的清单。在该文件中添加以下内容：

```
class mymodule::app {
  file { "/var/www/myserver/index.html":
    ensure => file,
    mode   => '755',
    owner  => root,
    group  => root,
    source => "puppet:///modules/mymodule/index.html",
    require => Class["mymodule::httpd"],
  }
}
```

这个新类包含单个资源声明。此声明将模块的文件目录从 Puppet 服务器复制到系统，并设置其权限。此外，require 属性可确保 mymodule::httpd 类在应用 mymodule::app 前成功完成配置。

最后，将这个新清单包括在我们的主 init.pp 清单中：

```
class mymodule (
  $httpd_port = 8120
) {
  include mymodule::httpd
  include mymodule::app
}
```

再次运行 puppet apply 命令，以测试对模块的更改。输出应类似于以下内容：

```
# puppet apply mymodule/tests/init.pp --noop
....
Notice: /Stage[main]/Mymodule::App/File[/var/www/myserver/index.html]/ensure: current_value
absent, should be file (noop)
...
```

此输出通知消息显示 index.html 文件将复制到 web 服务器。

2.10. 最后介绍模块

我们的模块已经可供使用。要将模块导出到要使用的 Red Hat Satellite 6 的归档中，请输入以下命令：

```
# puppet module build mymodule
```

这会在 mymodule/pkg/mymodule-0.1.0.tar.gz 中创建一个存档文件，其中包含 mymodule 目录的内容。我们将此模块上传到我们的红帽卫星 6 服务器，以调配我们自己的 HTTP 服务器。

如果需要进行任何更改，请编辑模块目录中的文件并使用 puppet 模块构建命令重建模块。只有在模块版本增加时，更改才会反映在 Satellite 中。要增加版本号，请编辑 /etc/puppetlabs/code/modules/mymodule/metadata.json 文件，然后重新重建模块。在卫星服务器中上传并发布新版本。

第 3 章 在 RED HAT SATELLITE 中添加 PUPPET 模块

您可以从 Puppet 服务器上的模块导入 Puppet 类，并分类了类所属的主机和 Puppet 环境。您还可以自定义分配的类的参数。这样，您可以控制通过外部节点分类器(ENC)将哪些参数传递给主机上的 puppet-agent。

3.1. PUPPET 环境

Puppet 环境定义为一组隔离的 Puppet 代理节点，可与一组特定的 Puppet 模块关联。在卫星环境中，我们可以将 Puppet 环境视为一组运行与一组 Puppet 模块关联的 Puppet 代理的主机。例如，与环境 *Production* 关联的节点只能访问 环境 *Production* 中的模块。

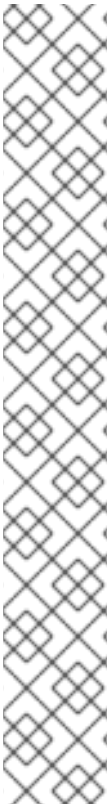
Puppet 环境用于将 Puppet 模块与不同类型的主机分隔开。典型的用途是启用在一个环境中测试模块的更改，然后再推送到另一个环境。Puppet 模块可以包含事实和功能，以及您可以分配给主机的一个或多个 Puppet 类。模块是环境的一部分，Puppet 类只是环境的一部分，因为它们是模块的一部分。

在 Red Hat Satellite 中，如果 CV 有 Puppet 模块，则会自动为内容视图创建一个 Puppet 环境。自动创建的 Puppet 环境名称包括组织标签、生命周期环境、内容视图名称和内容视图 ID。例如，KT_Example_Org_Library_RHEL6Server_3。

在 Red Hat Satellite 中创建 Puppet 环境

1. 导航到 Configure → Environments，再选择 New Puppet Environment。
2. 将新环境命名为，然后单击 Submit 以保存更改。

请注意，如果 Puppet master 上不存在环境，并且您随后运行导入，则会提示 Satellite 删除环境。



注意

在注册与生产环境中创建的 Puppet 环境关联的主机组时，Puppet 无法检索 Puppet CA 证书。要创建与主机组关联的适当 Puppet 环境，请按照以下步骤操作：

1. 手动创建目录并更改所有者：

```
# mkdir /etc/puppetlabs/code/environments/example_environment
# chown apache /etc/puppetlabs/code/environments/example_environment
```

2. 导航到 Configure → Environments，再从 单击 Import environment。按钮名称将包含内部或外部胶囊的 FQDN。

3. 选择创建的目录，然后单击 更新。

将 Puppet 环境导入到 Red Hat Satellite

卫星可以检测 Puppet master 中包含的所有环境和 Puppet 模块，再自动导入它们。要做到这一点，进入 Configure > Environments 并点 Import from 按钮。按钮名称将包含内部或外部胶囊的 FQDN。卫星将通过胶囊扫描 Puppet 宿主，并显示检测到的更改的列表。选择您要应用的更改，然后选择 Update 以应用更改。

请注意，胶囊将仅检测包含一个或多个 Puppet 类的环境，因此请确保至少有一个包含类的 Puppet 模块已部署到 Puppet 宿主。

将 Puppet 环境分配给主机

导航到 Hosts > All Hosts，从 Hosts 列表中选择一个主机名，然后选择 Edit。在 Host 选项卡上，您可以为主机 选择 Puppet 环境。选择一个环境将过滤 Puppet Classes 选项卡中的类，以帮助区分所选环境中的类。

您可以将环境重新分配到一组主机。在 Hosts 列表中选择所需主机的复选框，然后从页面顶部的 Select Action 下拉菜单中选择 Change Environment。

将 Puppet 环境分配给主机组

在创建 主机组 时，环境字段将预先填充（若有）由关联的内容视图自动创建的环境。

默认情况下，创建新主机的用户并选择 主机组 将自动具有预选的环境。用户可以更改新主机的环境。

3.2. 参数

Red Hat Satellite 参数定义置备主机时要使用的键值对。它们类似于 Puppet 的默认范围参数的概念。您可以在使用 Puppet 设置主机时定义参数。

参数的类型

Red Hat Satellite 有两个参数：

简单参数

定义键值对之间的关系的字符串参数。它们不能被用户配置覆盖，但它们会根据 Satellite 的参数层次结构覆盖。以下参数是 Red Hat Satellite 中的简单参数：全局、组织级、位置级、域级、子网级别、操作系统级别、主机组和主机参数。

智能类参数

为键定义值的复杂参数，但为特定对象类型允许条件参数、验证和覆盖。智能类参数使 Puppet 类能够获取外部数据。它们在 Puppet 类中使用，在 Puppet 术语中称为参数化类。这些参数的层次结构可以在 Web UI 中配置。

以下参数层次结构适用于简单参数：

全局参数

适用于 Satellite 中的每个主机的默认参数。在 Configure > Global parameters 中配置。要使用 Hammer CLI 设置全局参数，请输入以下命令：

```
# hammer global-parameter set --name parameter_name --value parameter_value
```

机构级参数

影响给定机构中的所有主机的参数。机构级参数覆盖全局参数。在 Administer > Organizations > Edit > Parameters 中进行配置。要使用 Hammer CLI 设置机构级参数，请输入以下命令：

```
# hammer organization set-parameter --organization "Your Organization" \
--name parameter_name --value parameter_value
```

位置级别参数

影响给定位置中的所有主机的参数。位置级别参数覆盖机构级别和全局参数。在 Administer > Locations > Edit > Parameters 中进行配置。要使用 Hammer CLI 设置位置级参数，请输入以下命令：

```
# hammer location set-parameter --location "Your_Location" \
--name parameter_name --value parameter_value
```

域参数

影响给定域中的所有主机的参数。域参数覆盖位置级别或更高参数。在 Infrastructure > Domains > [choose_a_domain] > Parameters.要使用 Hammer CLI 设置域参数，请输入以下命令：

```
# hammer domain set-parameter --domain domain_name \
--name parameter_name --value parameter_value
```

子网参数

影响给定子网中有主接口的所有主机的参数。子网参数覆盖主机组系统级别和更高的参数。在 Infrastructure > Subnets > [choose_a_subnet] > Parameters.要使用 Hammer CLI 设置子网参数，请输入以下命令：

```
# hammer subnet set-parameter --subnet subnet_name \
--name parameter_name --value parameter_value
```

操作系统级别参数

影响给定操作系统所有主机的参数。操作系统级别的参数覆盖域或更高参数。在 Hosts > Operating system > [choose_an_ operating_system] > Parameters.要使用 Hammer CLI 设置操作系统参数，请输入以下命令：

```
# hammer os set-parameter --operatingsystem os_name \
--name parameter_name --value parameter_value
```

主机组参数

影响给定主机组中的所有主机的参数。主机组参数覆盖操作系统级别或更高参数。在 Configure > Host Groups > [choose_a_host_group] > Parameters.要使用 Hammer CLI 设置主机组参数，请输入以下命令：

```
# hammer hostgroup set-parameter --hostgroup hostgroup_name \
--name parameter_name --value parameter_value
```

主机参数

影响特定主机的参数。所有之前继承的参数在参数子选项卡中可见，并可被覆盖。在 **Hosts > All hosts > Edit > Parameters.**要使用 **Hammer CLI** 设置主机参数，请输入以下命令：

```
# hammer host set-parameter --host host_name \
--name parameter_name --value parameter_value
```

将参数用于 Puppet 类

Red Hat Satellite 有两种方法可以为要用于 **Puppet** 类的 **Puppet master** 提供值：

智能变量

为不含智能类参数的类，为 **Puppet** 宿主提供全局参数的工具，格式为 **key-value** 形式。它们启用覆盖 **Puppet** 清单中的参数值。当类没有智能类参数或者需要全局参数时，它们被设计为使用。它们可以有多个可能的值，它们都取决于用户可以应用的分层上下文或各种条件。**Puppet** 在 **Puppet** 具有参数化类前存在，并且现在保持向后兼容性，或者用于需要验证的全局参数，仅用于与特定 **Puppet** 类使用，以及用于字符串以外的类型（因为您可以仅使用简单参数）。

参数化类

包含智能类参数的 **Puppet** 类。类从 **Puppet** 宿主导入，参数的名称（如 **$::***名称*（首选）或 **$*name*** 由编写该类且无法更改的人员定义。它们允许您决定特定类而非全局的变量值。

配置的参数包含在每个主机的对应 **YAML** 文件中，并发送到 **Puppet** 宿主。**YAML** 文件可在特定主机的页面中的 **Web UI** 中查看。您不应该手动更改 **/etc/foreman/settings.yaml** 配置文件，因为它们在下次运行 **satellite-installer** 命令时被覆盖。

重要

卫星中默认启用参数化类支持。要确认它已启用，请导航到 **Administer > Settings**，选择 **Puppet** 选项卡，并确保 **ENC** 中的 **Parameterized Classes** 设置为 **True**。

3.3. 配置智能类参数

以下步骤配置类中的参数。包含参数的类被称为参数化类。

智能类参数对所有机构都很常见。具有 edit_external_parameters 权限的用户都可以编辑这些参数。如果要限制权限编辑智能类参数，请参阅 KCS 解决方案 Restrict 权限来编辑 puppet 类及其智能类参数，它们在多个机构之间常见。

配置智能类参数

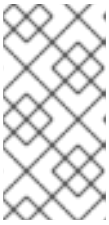
1. 单击 Configure > Puppet Classes。
2. 从具有 参数 列中指示的参数列表中选择一个类。
3. 单击 Smart Class Parameter 选项卡。这将显示一个新屏幕。left 部分包含类支持的可能参数列表。right 部分包含选择的参数的配置选项。
4. 从左侧列表中选择参数。
5. 编辑 Description 文本框，以添加任何纯文本注释。
6. 选择 Override 以允许 Satellite 对此变量进行控制。如果没有选中复选框，卫星不会将新变量传递给 Puppet。
7. 选择要传递的数据类型。这是最常见的字符串，但支持其他数据类型。
8. 如果发生主机匹配，则输入要发送到 Puppet master 的参数的 Default Value。
9. 可选： 选择 Omit 来 不向 Puppet 宿主发送值，除非发生覆盖匹配项。
10. 可选： 如果该字段包含您不想在操作时显示的数据，请选择 Hidden 值。
11. 使用 Optional Input Validator 部分来限制参数允许的值。在 Validator 规则字段中，选择 Validator 类型 （以逗号分隔的值或正则表达式、regexp） 的列表，并在 Validator 规则 字段中输入允许的值或正则表达式代码。

12. 如果选择了 Override 选项，则会出现 Prioritize Attribute Order 部分。这提供了根据条件参数为特定主机覆盖值的选项。属性类型及其值称为 matcher。

13. 在 Order 字段中，设置优先级顺序，其中将主机属性或事实的评估为根据匹配者进行评估，具体由列表中条目的排列。您可以添加一个属性，该属性在 Facter 中存在，且无法与默认列表混淆。

    如果要在匹配者间创建逻辑 AND 条件，并使用多个属性作为匹配程序键，将它们放在一行中，格式为用逗号分开的列表：

    ```
    location,environment
    ```

    注意

    在 BZ#1772381 被解决前，请注意，Order tooltip 显示使用多个属性作为匹配程序键的不正确的示例配置。

14. 单击 Add Matcher 以添加条件参数。要匹配的属性对应于 Order 列表中的条目。如果没有配置 matcher，则只有默认值可用于覆盖功能。

15. 从 Attribute type 列表，选择 属性。

16. 在属性旁边的字段中，输入属性字符串。

17. 在 Value 字段中输入您想要的值。

    可以使用嵌入式 Ruby(ERB)模板语法中的 Value 字段中的参数和 Puppet 事实实现动态数据。例如，要将 Puppet 事实用作值的一部分：

    ```
    <%= @host.facts['network_eth0'] %>
    ```

    若要列出可用的 Puppet 事实，可导航到 Monitor > Facts。

    如需有关 ERB 语法的更多信息，请参阅 *管理主机* 中的模板编写参考。

18. 点 Submit。

将单个属性设置为 matcher 键

您可以将单个属性设置为 matcher 键。

例如，若要为 Default_Location 位置的任何主机的 Puppet master 提供 test 值，请完成以下步骤：

1. 在 Order 字段中，附加 位置。
2. 单击 Add Matcher，再从 Attribute 类型 列表中选择 位置。
3. 在属性旁边的字段中，输入 Default_Location。
4. 在 Value 字段中输入 test。
5. 点 Submit。

将多个属性设置为 matcher 键

您可以将多个属性设置为匹配器。

例如，若要为 Default_Location 位置和 开发环境中 任何主机的 Puppet master 提供参数值，请完成以下步骤：

1. 在 Order 字段中，附加 location,environment。
2. 单击 Add Matcher，再从 Attribute 类型 列表选择 location,environment。
3. 在属性旁边的字段中，输入 Default_Location,development。

4. 在 Value 字段中输入 test。

5. 点 Submit。

匹配概述的优先级

在匹配的主机时，Satellite 会使用以下优先级：

1. 卫星在主机属性中搜索匹配者。

2. 如果主机属性中没有匹配项，卫星将搜索主机参数中的匹配程序，这会根据参数层次结构进行继承。

3. 如果主机参数中没有匹配项，卫星会在主机事实中搜索匹配程序。

3.4. 使用父主机组应用事实

您可以使用父主机组将事实应用到多个主机组的成员。

先决条件

- 您有分配了主机的主机组。
- 您拥有分配了主机组的父主机组。

将事实应用到多个主机组的成员

1. 在 Satellite Web UI 中，导航到 Configure > Classes。

2. 选择要使用的 Puppet 类。

3. 单击 Smart Class Parameter 选项卡。

4. 从您要覆盖的左侧列表中选择参数。

5. 可选：编辑 Description 字段，以添加任何纯文本注释。

6. 选择 Override 来授予 Satellite 对这个变量的控制权。

7. 选择要传递的数据类型。这是最常见的字符串，但支持其他数据类型。

8. 可选：在没有主机匹配时，为参数输入一个 默认值，以发送到 Puppet master。

9. 可选： 选择 Omit 以不向 Puppet 宿主发送值，除非发生覆盖匹配项。

10. 可选： 如果该字段包含您不想在操作时显示的数据，请选择 Hidden Value。

11. 可选：使用 Optional Input Validator 部分来限制参数允许的值。选择 Validator Type，它是以逗号分隔的值或正则表达式、regexp 的列表，并在 Validator rule 字段中添加允许的值或正则表达式代码。

12. 在 Prioritize Attribute Order 区域中，设置优先级顺序，该顺序在将主机属性或事实(fact)应用到匹配者时，按范围列表中条目来评估主机属性或事实。您可以添加到默认列表中。要在匹配者间逻辑 AND 条件，请将一行中的匹配者分隔为用逗号分开的列表。

13. 在 Specify Matchers 区域，单击 Add Matcher 以添加一个条件参数。

14. 将 Attribute type 设置为 hostgroup。

15. 在 = 符号后，输入您要匹配的主机组。在本例中，父主机组。

16. 在 Value 字段中输入您要发送到属于父主机组的 Content Hosts 的值。

17. 点 Submit。

如果父主机组是主机组层次结构的一部分，则按 matcher 值输入所有父项：*top_host_group*/*intermediate_host_group*/*parent_host_group*。

3.5. 使用多个自定义事实

以下是在 Puppet 智能类参数匹配器中创建和使用多个自定义事实及其值的示例。

先决条件

- 您已导入了 Puppet 环境，并在 Satellite 中使用智能类的 Puppet 模块。
- 您已将客户端置备并注册到 Satellite。
- 对于没有由 Satellite 置备的客户端，检查客户端是否已如 第 6.2 节 “将配置应用到现有客户端” 所述配置。

1. 在客户端上，创建两个或更多自定义事实，并为它们分配值。例如：

```
# vi /etc/facter/facts.d/my_custom_facts
#! /bin/bash
echo example_fact1=myfact1
echo example_fact2=myfact2
```

2. 在客户端中配置文件权限：

```
# chmod a+x /etc/facter/facts.d/my_custom_facts
```

3. 在客户端上，检查事实及其对应值：

```
# facter | grep example
 example_fact1 => myfact1
 example_fact2 => myfact2
```

4. 在 Satellite Web UI 中：

   a. 导航到 Configure > Classes 并选择您要配置的 Puppet 类。

   b. 点击 Smart Class Parameter 选项卡，然后选择您要覆盖的参数。

   c. 在 Default Behavior 区域，选中 Override 复选框。

   d. 在 Prioritize Attribute Order 区域（ Order 字段中），将示例事实添加到列表的末尾。要在两个事实之间创建逻辑 AND 条件，将其添加为逗号分隔列表 example_fact1,example_fact2。

   e. 选择 Add Matcher。

   f. 从 Attribute 类型 菜单中，选择 example_fact1,example_fact2，然后在 = 符号后面的框中选择 myfact1,myfact2。

   g. 在 Value 字段中，为两个属性及对应的值输入您要发送到内容主机的值。

   h. 点 Submit。

5. 在客户端上，将事实从客户端发送到 Puppet 宿主：

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

6. 在 Satellite Web UI 中：

   a. 导航到 Hosts > Content Hosts 并选择 Content Host 的名称。

   b. 点 YAML 并找到 class 部分。检查 参数是否具有您想要的值。

3.6. 配置智能变量

以下流程配置智能变量以覆盖 Puppet 类中的值。

配置智能变量

1. 单击 **Configure > Puppet Classes**。
2. 从列表中选择一个类。
3. 点 **Smart Variables** 选项卡。这将显示一个新屏幕。**left** 部分包含类支持的可能参数列表。**right** 部分包含选择的参数的配置选项。点 **Add Variable** 添加新参数。否则，从左侧列表中选择参数。
4. 在 **Key** 字段中输入参数的名称。
5. 编辑 **Description** 文本框，以添加任何纯文本注释。
6. 选择要 传递的数据的参数类型。这是最常见的字符串，但支持其他数据类型。
7. 如果发生主机匹配，则输入要发送到 **Puppet master** 的参数的 **Default Value**。
8. 可选： 如果该字段包含您不想在操作时显示的数据，请选择 **Hidden Value**。
9. 使用 **Optional Input Validator** 部分来限制参数允许的值。在 **Validator** 规则字段中，选择以逗号分隔的值或正则表达式列表(**regexp**)并在 **Validator** 规则 字段中输入允许的值或正则表达式代码。
10. **Prioritize Attribute Order** 部分提供了根据条件参数覆盖特定主机的值的选项。属性类型及其值称为 匹配程序。

    a. 设置 要针对 匹配人员评估主机属性或事实(**fact**)顺序排列排列列表中的条目的优先顺

序。您可以添加到默认列表中。要在匹配者间创建一个逻辑 AND 条件，将它们放在一行中，以逗号分隔列表。

b. 单击 Add Matcher 以添加条件参数。要匹配的属性应当对应于 Order 列表中的条目。如果没有配置 matcher，则只有默认值可用于覆盖功能。

例如，如果提供给 Puppet master 所需的参数值是 测试 任何具有全限定域名 server1.example.com 的主机，则将 matcher 指定为 fqdn=server1.example.com，值为 test。

匹配的优先级如下：

i. 如果 matcher 是主机属性，请使用：

ii. 如果没有具有该名称的属性，请查找匹配的 host 参数（这会根据参数层次结构进行继承）。

iii. 如果仍没有匹配项，请检查主机事实。

建议您使用一个在 Facter 中存在的属性，且无法与 host 属性混淆。主机属性可以是主机参数，也可以是与主机关联，如主机组、域和组织。matcher 只能是主机 中的一个内容，例如 config 组不能被使用，因为主机可以有多个配置组，但主机只能有一个位置，因此位置是有效的匹配器。

可以使用 嵌入式 Ruby (ERB)模板语法中的 Value 字段中的参数和 Puppet 事实实现动态数据。例如，要将 Puppet 事实用作值的一部分：

```
<%= @host.facts['network_eth0'] %>
```

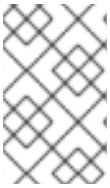
若要列出可用的 Puppet 事实，可导航到 Monitor > Facts。

11. 点 Submit 保存您的更改。

如需有关 ERB 语法的更多信息，请参阅 *管理主机* 中的模板编写参考。

3.7. 从 PUPPET MASTER 导入参数类

以下过程从 Puppet 宿主导入参数化类。



注意

如果您的 Puppet 模块通过产品和内容视图管理，则会自动发生参数化类导入。

导入参数化类

1. 在卫星 Web UI 中，从 Any Organization 和 Any Location 的上下文菜单中选择。
2. 单击 Configure > Puppet Classes。
3. 单击 Import from *Host Name*，以从 Puppet master 导入参数化类。
4. Puppet Classes 页面显示有列出的新类。

3.8. 使用智能变量工具

智能变量是为 Puppet 宿主提供全局参数的工具，可用于包含智能类参数。智能变量和智能类参数使用相同的智能匹配规则。

注意

智能变量工具在 Puppet 模块支持智能类参数之前作为内期测量进行介绍。如果您没有使用智能清单的具体原因，红帽建议使用智能类参数，如 第 3.3 节 “配置智能类参数” 所述。

在引入智能类参数前，希望覆盖一个参数的用户（需要重写其 Puppet 代码以使用全局参数）。例如：

```
class example1 {
  file { '/tmp/foo': content => $global_var }
}
```

对于上例，在 Web UI 的 Smart Variables 部分中设置了 $global_var，其值与 "example1" 类相关联。虽然建议在全局变量前面带有 :: 以限制 Puppet 搜索全局范围，但它们没有表示变量不是全局变量。

在引入 Smart Class Parameters 时，可以使用以下格式：

```
class example2($var="default") {
  file { '/tmp/foo': content => $var }
}
```

在上例中，web UI 的 Smart Class Parameters 部分中设置了 $var，其值与 "example2" 类关联。如果您看到类标头中定义的变量，如上面的 类 example2($var="default") 中所述，您可以确定 $var 是一个类参数，且您应该使用 Smart Class Parameter 功能覆盖变量。

如上所示，智能变量需要使用全局命名空间参数来自定义模块，而不是来自 Puppet 社区的标准模块，其结果与上述示例中的"/tmp/foo"中放入的文本相同。因此，除了支持旧模块外，不再使用智能卡验证。

虽然智能变量是全局变量，但它们与 Puppet 类关联，并且仅发送到在卫星中分配了特定 Puppet 类的主机。您可以使用任何名称创建智能变量，不会在卫星中执行验证，除非应用的 Puppet 模块在其代码中具有匹配的变量，否则不会使用智能变量。

Satellite 将您在 Satellite 中创建的变量添加到 Host YAML 文件中。可在 Web UI 中查看此文件，方法是导航到 Hosts > All Hosts，选择主机的名称，然后单击 YAML 按钮。卫星将主机 YAML 文件发送到 *外部节点分类器* (ENC)，这是 Puppet 宿主中包含的功能。当 Puppet 宿主查询有关主机的 ENC 时，ENC 会返回描述主机状态的 YAML 文档。此 YAML 文档基于从 Puppet 清单获取的数据，但受智能类参数覆盖和任何智能清单。

将智能变量应用到主机

由于智能变量应仅用于支持之前修改的自定义 Puppet 模块以包含全局参数，以下示例使用一个名为 anothermodule 的简单示例。另一模块 Puppet 清单如下所示：

```
class anothermodule {
   file { '/tmp/motd':
     ensure  => file,
     content => $::content_for_motd,
  }
}
```

本例将为 $::content_for_motd 参数提供值。

1. 在 Web UI 中，导航到 Configure > Classes。

2. 从列表中选择 Puppet 类的名称。

3. 点 Smart Variables 选项卡。这将显示一个新屏幕。left 部分包含之前创建的参数的列表（若有）。right 部分包含配置选项。

4. 点击 Add Variable 添加新参数。

5. 在 Key 字段中输入 参数。在本例中，content_for_motd。

6. 编辑 Description 文本框，如 Testing /tmp motd Text。

7. 选择要 传递的数据的参数类型。选择 字符串。

8. 为 参数输入 Default Value。例如，No Unauthorized Use。

9. 使用 Optional Input Validator 部分来限制参数允许的值。在 Validator 规则字段中，选择以逗号分隔的值或正则表达式列表(regexp)并在 Validator 规则 字段中输入允许的值或正则表达式代码。

10. 使用 Prioritize Attribute Order 部分设置针对匹配者评估主机属性或事实的优先级顺序（配置如下）。您可以重新排列列表中的条目，并添加到默认列表中。要在匹配者间逻辑 AND 条件，将匹配者的名称放在一行中，以逗号分隔列表形式排列。

11. 在 Specify Matchers 部分，单击 Add Matcher 以添加一个条件参数。要匹配的属性应当对应于上述 Order 列表中的条目。如果没有配置 matcher，则只能使用默认值。例如，如果参数所需的值 为 server1.example.com 的任何主机的 Server1，则可将 Match 指定为 fqdn=server1.example.com，并将 Value 指定为 Server1。

12. **点 Submit 保存您的更改。**

第 4 章 存储和维护主机信息

红帽卫星 6 使用应用程序组合来收集有关受管主机的信息，并确保这些主机在所需状态维护。这些应用程序包括：

- Foreman：提供物理和虚拟系统的调配与生命周期管理。Foreman 使用各种方法自动配置这些系统，包括 kickstart 和 Puppet 模块。
- Puppet：用于配置主机的客户端/服务器架构，由 Puppet 宿主(server)和 Puppet 代理(client)组成。
- Facter：Puppet 的系统清单工具。Facter 收集有关主机的基本信息（事实），如硬件详细信息、网络设置、操作系统类型和版本、IP 地址、MAC 地址、SSH 密钥等。然后，这些事实在 Puppet 清单作为变量提供。

下方更为详细地阐述了 Puppet、Bquik 和 facts 的使用。

4.1. PUPPET 架构

Puppet 通常在 agent/master（也称为 client/server）架构中运行，其中 Puppet 服务器控制重要配置信息，受管主机（客户端）仅请求自己的配置目录。Puppet 配置两个步骤的主机：

- 它编译目录。
- 它将该目录应用到适当的主机。

在 agent/master 设置中，Puppet 客户端会发送通过 Facter 和其他信息收集的事实到 Puppet 宿主。Puppet 宿主根据这些事实编译目录，然后将此目录发送到客户端。客户端会发送它进行的所有更改的报告，或者在已使用 --noop 参数时进行 Puppet 宿主，后者将结果发送到 Foreman。此目录描述了一个特定主机所需的状态。它列出了要管理该主机上的资源，包括这些资源之间的任何依赖项。代理将目录应用到主机。

默认情况下，master 和代理之间的通信每 30 分钟进行。您可以使用 runinterval 参数在 /etc/puppetlabs/puppet/puppet.conf 文件中指定不同的值。您还可以运行 puppet 代理来 手动启动通信。

4.2. 使用 FACTER 和事实

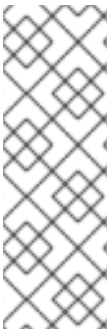
Facter 是 Puppet 的系统清单工具，包含大量内置事实。您可以在本地主机上命令行中运行 Facter，以显示事实名称和值。您可以使用自定义事实扩展 Facter，然后使用这些事实将主机的特定站点详细信息公开给 Puppet 清单。您还可以使用 Facter 提供的事实在 Puppet 中通知条件表达式。

Puppet 基于资源来确定系统状态；例如，您可以告知 Puppet httpd 服务应始终处于运行状态，并且 Puppet 知道如何处理它们。如果要管理不同的操作系统，您可以使用 osfamily 事实来创建条件表达式，以告知 Puppet 哪个服务监视或要安装的软件包。您可以使用 operatingsystemmajrelease 和 versioncmp 参数根据不同版本的同一操作系统创建条件表达式。下例演示了将条件表达式与事实搭配使用。

将条件表达式与事实搭配使用

```
if $::osfamily == 'RedHat' {
  if $::operatingsystemmajrelease == '6' {
   $ntp_service_name = 'ntpd'
   }

  elseif versioncmp($::operatingsystemmajrelease, '7') &gt;= 0 {
   $ntp_service_name = 'chrony'
   }
 }
```



注意

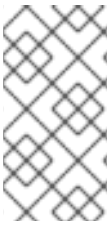
这个示例使用表达式 versioncmp（$:: operatingsystemmajrelease、'7'） >= 0 测试 Red Hat Enterprise Linux 版本 7 或更高版本。不要使用表达式 $:: operatingsystemmajrelease >= '7' 来执行这个测试。有关此问题和其他 Puppet 功能的更多信息，请参阅 https://docs.puppetlabs.com/references/latest/function.html#versioncmp。

Puppet 也会设置其他行为类似于事实的特殊变量。如需更多信息，请参阅 Puppet 和 核心事实 添加的特殊变量。

4.2.1. 显示 Particular Host 的事实

Puppet 可以访问 Facter 的内置核心事实，以及 Puppet 模块中存在的任何自定义或外部事实。您可

以从命令行中查看可用事实（事实(facter -p)），也可从 Web UI(Monitor > Facts)查看可用事实。您可以浏览事实列表或使用 搜索框 来搜索特定事实。例如，键入 "facts." 以显示可用事实的列表。



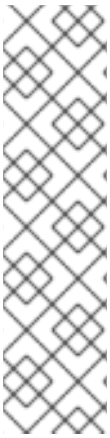
注意

可用事实列表非常长。UI 一次仅显示 20 事实。在输入更多详细信息时，事实列表会逐渐增加过滤器。例如，键入 "facts.e" 以显示以字母"e"开头的所有事实。

查看 Particicular Host 的事实

1. 在主菜单中点击 Hosts > All Hosts，然后点您要检查的主机的名称。

2. 在 Details 窗格中，单击 Facts 以显示有关主机的所有已知事实。



注意

- 对于此页面列出的任何事实，您可以点击 Chart 来显示所有受管主机上此事实名称的分布图表。

- 您可以为搜索添加书签，使其在将来更易于使用。优化搜索后，点击搜索按钮旁边的下拉箭头，然后单击 此搜索的书签。书签搜索会出现在 搜索 下拉列表中，也显示在主菜单中的 Administer > Bookmarks 下。

4.2.2. 根据事实搜索主机

您可以使用 Facter 信息来搜索特定的主机。这意味着，您可以搜索与特定事实条件匹配的所有主机，如 facts.architecture = x86_64。

根据事实搜索主机

1. 在主菜单中，点击 Monitor > Facts 以显示 Fact Values 页面。

2. 在 搜索字段中，开始输入您要过滤的事实的名称。您可以根据具体名称、名称/值对进行搜索，以此类推。

3. 单击 **Search** 以检索匹配主机的列表。

4.2.3. 定制事实报告

在 **Red Hat Satellite 6** 中完全支持从受管主机获取自定义信息。本节演示了使用从 **Puppet Forge** 获取的 **Puppet** 模块，但原则在 **Puppet** 模块的其他来源中同样适用。

通过标准 **Facter** 接口报告的事实数量可以扩展。例如，若要收集模块中用作变量的事实：如果事实说明了安装的软件包可用，您可以搜索此数据并根据信息作出明智的配置管理决策。

要获得在主机中安装的软件包报告，如下所示：

- 清单 **pkginventory** 从 **Puppet Forge** 获取，并保存到基本系统中。
- **Puppet** 模块添加到内容视图中，然后将其提升到系统并部署到该系统。
- 然后会使用软件包名称查询系统的事实。在本例中，对于名为 ***hostname*** 的主机，使用带有凭证 *用户名和密码* 的 **Satellite** 用户，以下 **API** 查询会返回与搜索字符串 **"bash"** 匹配的事实：

```
curl -u username:password -X GET http://localhost/api/hosts/:hostname/facts?search=bash
{"hostname":{"pkg_bash":"4.2.45-5.el7_0.4"}}
```

搜索返回软件包版本。然后，这可用于填充外部数据库。

4.2.3.1. 添加 pkginventory Puppet 模块

要将 **pkginventory Puppet** 模块添加到 **Red Hat Satellite Server** 应用程序，请将模块从 https://forge.puppetlabs.com/ody/pkginventory 下载到安装 **Satellite** 服务器应用程序的基本系统中，然后按照以下步骤操作。

Puppet 模块通常存储在名为 **Puppet Modules** 的自定义存储库中。以下流程假设您已使用该名称进行了自定义存储库。如果您还没有为 **Puppet** 模块提供自定义存储库，请参阅 *快速入门指南* 中的创建自定义产品。

上传 Puppet 模块到存储库

1. 将 Puppet 模块下载到基本系统。下载的模块将具有 .tar.gz 扩展。
2. 单击 Content > Products，然后单击与 Puppet 模块存储库关联的 Name 字段中的产品名称。例如，自定义产品。
3. 在 Repositories 选项卡上，选择要修改的 Puppet Modules 存储库。例如，Puppet 模块。
4. 在 Upload Puppet Module 部分中，单击 Browse，然后导航到您下载的模块。
5. 点 Upload。

要将 Puppet 模块分发到客户端，内容主机，模块必须应用到内容视图并发布。按照以下步骤将模块添加到内容视图中。

在内容视图中添加模块

1. 点 Content > Content Views，然后从 Name 菜单中选择 Content View。
2. 在 Puppet 模块 选项卡上，单击 Add New Module。此时会出现安装的模块列表。
3. 在 Actions 列中，单击 Select a Version 以选择要添加的模块。此时会出现可用版本表。
4. 单击要添加的模块版本旁边的 Select Version。
5. 单击 Publish New Version 以创建新内容视图。

6. **（可选）添加描述信息并点 Save。**

第 5 章 配置和管理的客户端和服务器设置

红帽卫星 6 配置流程的一个重要部分是确保 Puppet 客户端（称为 Puppet 代理）可以在内部卫星胶囊或外部卫星胶囊上与 Puppet 服务器（称为 Puppet 宿主）通信。本章阐述红帽卫星 6 如何配置 Puppet 宿主和 Puppet 代理。

5.1. 在 RED HAT SATELLITE 服务器中配置 PUPPET

红帽卫星 6 控制所有卫星胶囊上 Puppet 宿主的主要配置。不需要额外的配置，建议避免手动修改这些配置文件。例如，主 /etc/puppetlabs/puppet/puppet.conf 配置文件包含以下 [master] 部分：

```
[master]
    autosign      = $confdir/autosign.conf { mode = 664 }
    reports       = foreman
    external_nodes = /etc/puppetlabs/code/node.rb
    node_terminus  = exec
    ca            = true
    ssldir        = /var/lib/puppet/ssl
    certname      = sat6.example.com
    strict_variables = false

    manifest      = /etc/puppetlabs/code/environments/$environment/manifests/site.pp
    modulepath    = /etc/puppetlabs/code/environments/$environment/modules
    config_version =
```

本节包含变量（如 $环境变量），Satellite 6 用于为不同的环境创建配置。

一些 Puppet 配置选项显示在 Satellite 6 web UI 中。导航到 Administer > Settings 并选择 Puppet 子选项卡。此页面列出了一组 Puppet 配置选项和各个选项的描述。

5.2. 在 PROVISIONED 系统中配置 PUPPET 代理

作为调配过程的一部分，卫星 6 将 Puppet 安装到系统。此过程还会在所选胶囊上安装 /etc/puppetlabs/puppet/puppet.conf 文件，该文件将 Puppet 配置为 Puppet 宿主的代理。此配置文件在卫星 6 中存储为置备模板片段。导航到 Hosts > Provisioning 模板，再点击 puppet.conf 片断来查看它。

默认的 puppet.conf 片段包含以下代理配置：

```
[agent]
pluginsync      = true
report          = true
```

```
ignoreschedules = true
daemon          = false
ca_server       = <%= @host.puppet_ca_server %>
certname        = <%= @host.certname %>
environment     = <%= @host.environment %>
server          = <%= @host.puppetmaster %>
```

此片段包含一些模板变量，它们如下：

- **@host.puppet_ca_server 和 @host.certname - 用于保护 Puppet 通信的证书和证书颁发机构。**
- **@host.environment - Satellite 6 服务器上的 Puppet 环境用于配置。**
- **@host.puppetmaster - 包含 Puppet 宿主的主机。这是卫星 6 服务器的内部胶囊或外部卫星胶囊。**

第 6 章 在客户端应用配置

此时，卫星 6 服务器的 Puppet 生态系统已经配置，包含 mymodule 模块。现在，我们的目的是将模块的配置应用到注册的系统。

6.1. 在置备过程中应用客户端的配置

我们首先使用以下步骤定义新主机的 Puppet 配置。此流程使用上传的 mymodule 作为示例。

在置备过程中应用客户端的配置

1. 导航到 **Hosts > New host**。
2. 单击 **主机** 选项卡。输入主机的 **Name**，并为系统选择机构和位置。选择 **生命周期环境** 及其 **提升** 的内容视图。这将定义要用于配置的 Puppet 环境。另外，从 **Capsule Settings** 中选择 **Puppet CA** 和 **Puppet Master**。所选的胶囊充当控制配置并与新主机上的代理通信的服务器。
3. 单击 **Puppet Classes** 选项卡，再从 **Available Classes** 部分中选择包含要应用的配置的 Puppet 类。在我们的示例中，选择：
   - **mymodule**
   - **mymodule:httpd**
   - **mymodule:app**
4. 从 **Operating System** 选项卡中选择所需选项。这些选项取决于您的 Satellite 6 基础架构。确保 **Provisioning templates** 选项包括 **Satellite Kickstart Default kickstart** 模板。此模板包含新主机上 Puppet 代理的安装命令。
5. 单击 **Parameters** 选项卡，再向 Puppet 类参数提供任何自定义覆盖。请参阅 第 3.3 节 “配置智能类参数” 启用此功能。

6. 完成所有置备选项后，点 Submit。

置备过程开始。卫星 6 将安装所需的配置工具，作为卫星 Kickstart 默认 置备模板的一部分。此置备模板包含以下内容：

```
<% if puppet_enabled %>
# and add the puppet package
yum -t -y -e 0 install puppet

echo "Configuring puppet"
cat > /etc/puppetlabs/puppet/puppet.conf << EOF
<%= snippet 'puppet.conf' %>
EOF

# Setup puppet to run on system reboot
/sbin/chkconfig --level 345 puppet on

/usr/bin/puppet agent --config /etc/puppetlabs/puppet/puppet.conf -o --tags no_such_tag <%=
@host.puppetmaster.blank? ? '' : "--server #{@host.puppetmaster}" %> --no-daemonize
<% end -%>
```

本节执行以下操作：

- 从 Red Hat Satellite Tools 6.10 软件仓库安装 puppet 软件包。
- 将 Puppet 配置片断安装到位于 /etc/puppetlabs/puppet/puppet.conf 中的系统。
- 启用 Puppet 服务在系统上运行。
- 第一次运行 Puppet，并初始化节点。

在新的主机上完成置备和配置过程后，请在 Web 浏览器中通过用户定义的端口访问主机。例如，导航到 http://newhost.example.com:8120/。您的浏览器中应该显示以下消息：

```
Congratulations

Your puppet module has correctly applied your configuration.
```

6.2. 将配置应用到现有客户端

您可能希望将 Puppet 配置应用到未通过 Red Hat Satellite 6 置备的现有客户端。在这种情况下，在将 Puppet 注册到红帽卫星 6 后，在现有客户端上安装和配置 Puppet。

将现有系统注册到 Red Hat Satellite 6。有关注册现有主机的详情，请参考 *管理主机指南中的注册主机*。https://access.redhat.com/documentation/zh-cn/red_hat_satellite/6.10/html/managing_hosts/registering_hosts

重要

puppet 软件包是 Red Hat Satellite Tools 6.10 软件仓库的一部分。在继续操作前，请确保启用此软件仓库。

要安装并启用 Puppet 代理：

1. 打开终端控制台并以 root 用户身份登录。
2. 安装 Puppet 代理：

```
# yum install puppet
```

3. 将 Puppet 代理配置为在引导时启动：
   - 在 Red Hat Enterprise Linux 6 中：

```
# chkconfig puppet on
```

   - 在 Red Hat Enterprise Linux 7 上：

```
# systemctl enable puppet
```

配置 Puppet 代理

1. **通过更改 /etc/puppetlabs/puppet/puppet.conf 文件来配置 Puppet 代理：**

```
# vi /etc/puppetlabs/puppet/puppet.conf
```

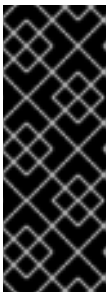
```
[main]
    # The Puppet log directory.
    # The default value is '$vardir/log'.
    logdir = /var/log/puppet

    # Where Puppet PID files are kept.
    # The default value is '$vardir/run'.
    rundir = /var/run/puppet

    # Where SSL certificates are kept.
    # The default value is '$confdir/ssl'.
    ssldir = $vardir/ssl

[agent]
    # The file in which puppetd stores a list of the classes
    # associated with the retrieved configuratiion.  Can be loaded in
    # the separate puppet executable using the --loadclasses
    # option.
    # The default value is '$confdir/classes.txt'.
    classfile = $vardir/classes.txt
    pluginsync = true
    report = true
    ignoreschedules = true
    daemon = false
    ca_server = satellite.example.com
    server = satellite.example.com
    environment = KT_Example_Org_Library_RHEL6Server_3

    # Where puppetd caches the local configuration.  An
    # extension indicating the cache format is added automatically.
    # The default value is '$confdir/localconfig'.
    localconfig = $vardir/localconfig
```



重要

从卫星服务器，将 environment 参数设置为主机的 Puppet 环境。Puppet 环境标签包含组织标签、生命周期环境、内容视图名称和内容视图 ID。要在 Satellite 6 web UI 中查看 Puppet 环境列表，请导航到 Configure > Environments。

2. **在主机上运行 Puppet 代理：**

```
# puppet agent -t --server satellite.example.com
```

3. 通过 Satellite 服务器 Web 界面为 Puppet 客户端签名 SSL 证书：

   a. 通过 Web 界面登录卫星服务器。

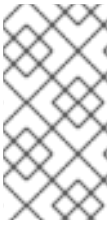
   b. 选择 Infrastructure > Capsules。

   c. 点 所需主机右侧的证书。

   d. 点 Sign。

   e. 再次运行 puppet agent 命令：

```
# puppet agent -t --server satellite.example.com
```



注意

当在主机上配置 Puppet 代理时，它将列在 All Hosts 下，但只有选择为主机 的任何上下文 时，才会分配到组织或位置。

第 7 章 使用配置组管理 PUPPET 类

红帽卫星包括配置 *组和主机 组的概念*，以支持以模块化方式构建和管理一组系统。

配置组是您创建的 Puppet 类的集合，用于配置主机和主机组的构建块。配置组与配置集社区 Puppet 概念类似于 Puppet 类，后者包含一组 Puppet 类来组成构建块。可以在 Satellite Web UI 中创建和管理配置组。

主机组既是主机服务器和容器的集合，用于系统定义、内容视图、分配的生命周期以及一组 Puppet 模块。主机组与社区 Puppet 概念类似于 Puppet 类，该类包含许多配置集来构建具有特定业务角色的系统。可以使用 Hammer CLI 并使用 API 在卫星 Web UI 中创建和管理主机组。

创建配置组

从左侧的上下文下拉菜单中，选择 Any Organization 和 Any Location。

进入 Configure > Config groups。

选择 New Config Group 并输入名称，例如 *TestConfGroup*。

从可用 类列表中选择一个或多个 Puppet 类。

选择 Submit 以应用更改。

创建配置组后，它将可用于在配置主机或主机组时选择 Puppet Classes 选项卡。有关创建主机组的更多信息，请参阅《红帽卫星 *配置指南》中的在卫星服务器上创建* 主机组。

第 8 章 查看 RED HAT SATELLITE 6 中的 PUPPET 报告

每当应用配置时，Puppet 都会生成报告。调配的主机将此报告发送到红帽卫星 6 服务器。在主机详情页面中查看这些报告。

查看 Red Hat Satellite 6 中的 Puppet 报告

1. 进入 **Hosts > All hosts**。
2. 点所需主机的 **Name**。
3. 点 **Reports** 按钮。
4. 选择要查看的报告。

每个报告显示每个 Puppet 资源的状态及其应用到主机的配置。