Red Hat Satellite 6.7

Puppet ガイド

独自の Puppet モジュールを構築し、これを Red Hat Satellite 6 にインポートするためのガイド

Last Updated: 2024-02-14

Red Hat Satellite 6.7 Puppet ガイド

独自の Puppet モジュールを構築し、これを Red Hat Satellite 6 にインポートするためのガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

Puppet は、Red Hat Satellite 6 で使用されるシステム設定ツールです。本書では、基本的な Puppet モジュールの作成および Red Hat Satellite 6 インフラストラクチャーでのこのモジュールの使用方法について説明します。

目次

第1章 概要

Puppet は、システム設定の適用および管理を行うためのツールです。Puppet はシステム情報 (ファクト) を収集し、この情報を使用して、モジュールセットでカスタマイズされたシステム設定を作成します。これらのモジュールには、パラメーター、条件付き引数、アクション、テンプレートが含まれます。Puppet は、ローカルシステムのコマンドラインツールとして使用されるか、またはサーバーが Puppet マスターとして機能し、Puppet エージェントを使用して複数のクライアントシステムに設定を適用するクライアント/サーバー関係において使用されます。これにより、新たにプロビジョニングしたシステムが (個別または同時に) 自動的に設定される方法が提供され、特定のインフラストラクチャーが作成されます。

1.1. PUPPET ワークフローの定義

Puppet は、以下のワークフローを使用して設定をシステムに適用します。

1. 各システムに関するファクトを収集します。これらのファクトには、ハードウェア、オペレーティングシステム、パッケージバージョンなどの情報が含まれます。各システムの Puppet エージェントは、この情報を収集して、Puppet マスターに送信します。
2. Puppet マスターは各システムのカスタム設定を生成し、Puppet エージェントに送信します。このカスタム設定はカタログと呼ばれます。
3. Puppet エージェントは、設定をシステムに適用します。
4. Puppet エージェントは、変更の適用済みおよび変更の失敗を示す Puppet マスターにレポートを送り返します。
5. サードパーティーアプリケーションは、Puppet の API を使用してこれらのレポートを収集できます。

1.2. SATELLITE 6 での PUPPET の使用

Red Hat Satellite 6 では、Puppet は複数の方法で使用します。

- Red Hat Satellite 6 は、システム設定の定義に使用する Puppet モジュールをインポートします。これには、モジュールバージョンおよびその環境の制御が含まれます。
- Red Hat Satellite 6 では、Puppet モジュールから Puppet スマートクラスパラメーターとも呼ばれるパラメーターセットをインポートします。ユーザーは、Puppet クラスからのデフォルト値を使用するか、グローバルまたはシステム固有のレベルで独自の値を提供できます。
- Red Hat Satellite 6 は、各システムのマスターと該当するエージェントの間で Puppet の実行をトリガーします。Puppet の実行は、以下のいずれかで実行されます。
    - 自動での実行 (プロビジョニングプロセスの完了後や、ライフサイクルでマシンの設定を確認および管理するデーモンなど)。
    - 手動での実行 (管理者が即時に Puppet 実行をトリガーする必要がある場合など)。
- Satellite 6 は、設定ワークフローの完了後に Puppet からレポートを収集します。これは、長期間にわたるシステム設定の監査やアーカイブに役立ちます。

これらの機能は、Puppet を使用してアプリケーションライフサイクルのシステム設定を簡単に制御する方法を提供します。

必要に応じて、お使いの Satellite 環境で使用される Puppet のバージョンを確認するには、『パッケージマニフェスト』を参照してください。

1.3. SATELLITE 6 での PUPPET のパフォーマンスおよびスケーラビリティー

Satellite 6 における Puppet のパフォーマンスは、アプリケーションの制限よりも Satellite および Capsule のストレージ容量、CPU パフォーマンス、および利用可能なメモリーの影響を受けます。したがって、ハードウェアと設定のテストは、パフォーマンスがニーズに合っているかどうかを確かめる唯一の方法となります。

推奨されるストレージ要件およびシステム要件の詳細は、『**オンラインネットワークからの Satellite Server のインストール**』の「**インストールのための環境準備**」を参照してください。

第2章 ゼロからの PUPPET モジュールの構築

本章では、独自の Puppet モジュールを構築してテストする方法を説明します。これには、簡単な Web サーバー設定をデプロイする Puppet モジュールの作成に関する基本的なチュートリアルが含まれます。

2.1. PUPPET モジュールの内容の検証

モジュールを作成する前に、Puppet モジュールを作成するコンポーネントを理解する必要があります。

Puppet Manifest (Puppet マニフェスト)

マニフェスト（サブスクリプションマニフェスト）と混同しないようにしてください。Puppet マニフェストは、リソースとその属性を定義するコードが含まれるファイルです。リソースは、システムの構成可能な部分です。リソースの例として、パッケージ、サービス、ファイル、ユーザーとグループ、SELinux 設定、SSH キー認証、cron ジョブなどが挙げられます。マニフェストは、属性のキーと値のペアのセットを使用して必要な各リソースを定義します。以下は例になります。

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

この宣言は、**httpd** パッケージがインストールされているかどうかを確認します。インストールされていない場合は、マニフェストが **yum** を実行してインストールします。

コンパイルを開始するのに使用する Puppet マニフェストは、「main manifest」または「site manifest」と呼ばれます。

マニフェストは、モジュールの **manifest** ディレクトリーに配置されています。

Puppet モジュールは、テストマニフェストの **テスト** ディレクトリーも使用します。これらのマニフェストを使用して、正式なマニフェストに含まれている特定のクラスをテストします。

静的ファイル

モジュールには、システムの特定の場所に Puppet がコピーできる静的ファイルを含めることができます。これらの場所や、パーミッションなどのその他の属性は、マニフェストの **file** リソース宣言で定義されます。
静的ファイルは、モジュールの **files** ディレクトリーに配置されています。

テンプレート

設定ファイルにはカスタムコンテンツが必要な場合があります。このような場合、ユーザーは静的ファイルの代わりにテンプレートを作成します。静的ファイルと同様に、テンプレートはマニフェストで定義され、システム上の場所にコピーされます。相違点は、テンプレートでは Ruby 式がカスタマイズのコンテンツと変数入力を定義できることです。たとえば、カスタマイズ可能なポートで **httpd** を設定する場合には、設定ファイルのテンプレートには以下が含まれます。

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

この場合の **httpd_port** 変数はこのテンプレートを参照するマニフェストで定義されています。

テンプレートは、モジュールの **templates** ディレクトリーに配置されています。

プラグイン

プラグインは、Puppet のコア機能を超える要素を可能にします。プラグインを使用してカスタムファクト、カスタムリソース、または新機能を定義できます。たとえば、データベースの管理者は、PostgreSQL データベースのリソースタイプが必要になる場合があります。これにより、データベース管理者は PostgreSQL のインストール後に新規データベースセットを PostgreSQL に投入しやすくなります。その結果、データベース管理者は、PostgreSQL のインストールとその後のデータベース作成を確実に行う Puppet マニフェストのみを作成するだけでよくなります。
プラグインは、モジュールの **lib** ディレクトリーに配置されています。これには、プラグインのタイプに応じたサブディレクトリーセットが含まれます。以下は例になります。

- **/lib/facter**: カスタムファクトの場所。
- **/lib/puppet/type**: 属性のキーと値のペアを記述するカスタムリソースタイプの定義の場所。
- **/lib/puppet/provider**: リソースを制御するためのリソースタイプの定義と併用するカスタムリソースプロバイダーの場所。
- **/lib/puppet/parser/functions**: カスタム関数の場所

2.2. PUPPET 開発システムの設定

Puppet 開発システムは、独自のモジュールを作成およびテストする際に役立ちます。

手順

以下の手順を使用して、新しい Puppet 開発システムをデプロイします。

1. 新しい Red Hat Enterprise Linux 7 システムをデプロイし、システムを Red Hat CDN または Satellite Server に登録します。
2. Puppet 5 の Red Hat Satellite Tools 6.7 リポジトリーを有効にします。

```
# subscription-manager repos \
--enable=rhel-7-server-satellite-tools-6.7-rpms
```

3. Puppet エージェントをインストールします。

```
# yum install puppet
```

2.3. 新しいモジュールボイラープレートの生成

新規モジュール作成の最初の手順は、Puppet モジュールディレクトリーに切り替え、基本的なモジュール構造を作成することです。この構造を手動で作成するか、Puppet を使用してモジュール用のボイラープレートを作成します。

```
# cd /etc/puppetlabs/code/modules
# puppet module generate user_name-module_name
```

インタラクティブなウィザードが表示され、メタデータでのモジュールの **metadata.json** ファイルの入力方法を説明します。

モジュールの生成プロセスが完了すると、新しいモジュールには **manifests** ディレクトリーを含む一部

の基本ファイルが含まれます。このディレクトリーには、モジュールのメインマニフェストファイルである **init.pp** というマニフェストファイルがすでに含まれています。ファイルを表示して、モジュールの空のクラス宣言を表示します。

```
class mymodule {


}
```

モジュールには、同じく **init.pp** という名前のマニフェストを含む **examples** ディレクトリーも含まれています。このテストマニフェストには、**manifests/init.pp** 内の **mymodule** クラスへの参照が含まれます。

```
include::mymodule
```

Puppet はこのテストマニフェストを使用してモジュールをテストします。

これで、システム設定をモジュールに追加する準備ができました。

2.4. HTTP サーバーのインストール

Puppet モジュールは、HTTP サーバーの実行に必要なパッケージをインストールします。これには、**httpd** パッケージの設定を定義するリソース定義が必要です。

モジュールの **manifests** ディレクトリーに、**httpd.pp** という名前の新規のマニフェストファイルを作成します。

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

このマニフェストには、モジュールのすべての HTTP 設定が含まれます。組織上の理由から、このマニフェストを **init.pp** マニフェストから分離します。

以下の内容を新しい **httpd.pp** マニフェスト追加します。

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

このコードは、**httpd** と呼ばれる **mymodule** のサブクラスを定義し、続いて **httpd** パッケージのパッケージリソース宣言を定義します。**ensure ⇒ installed** 属性は、パッケージがインストールされているかどうかを確認するよう Puppet に指示します。インストールされていない場合には、Puppet は **yum** を実行してパッケージをインストールします。

また、メインのマニフェストファイルにこのサブクラスを含める必要があります。**init.pp** マニフェストを編集します。

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

ここで、モジュールをテストします。以下のコマンドを実行します。

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

この出力通知メッセージは、**ensure ⇒ installed** 属性の結果です。**current_value absent** は、**httpd** パッケージがインストールされていないことを Puppet が検出したことを意味します。**--noop** オプションを指定しないと、Puppet は **httpd** パッケージをインストールします。

puppet apply コマンドは、マニフェスト内の設定をシステムに適用します。メインの **init.pp** マニフェストを参照するテスト **init.pp** マニフェストを使用します。**--noop** は、設定のドライランを実行します。これは出力のみを表示しますが、実際には設定を適用しません。

2.5. HTTP サーバーの実行

httpd パッケージのインストール後に、別のリソース宣言 **service** を使用してサービスを起動します。

httpd.pp マニフェストを編集し、強調表示されている行を追加します。

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

これにより、以下の操作が実行されます。

- **ensure ⇒ running** 属性は、サービスが実行中かどうかを確認します。実行中でない場合は、Puppet がサービスを起動します。
- **enable ⇒ true** 属性は、システムの起動時にサービスが実行されるように設定します。
- **require ⇒ Package["httpd"]** 属性は、リソース宣言間の順位関係を定義します。このケースでは、**httpd** サービスが **httpd** パッケージのインストール後に起動されるようにします。これにより、サービスと対応するパッケージの間で依存関係が作成されます。

puppet apply コマンドを再度実行して、モジュールへの変更をテストします。

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

この出力通知メッセージは、**httpd** サービスに対する新しいリソース定義の結果です。

2.6. HTTP サーバーの設定

HTTP サーバーがインストールされ、有効化されました。次のステップでは、設定を指定します。HTTP サーバーは、ポート 80 に Web サーバーを提供する **/etc/httpd/conf/httpd.conf** にデフォルト設定をすでにいくつか提供しています。ユーザー指定のポートに追加の Web サーバーを提供するために、設定を追加します。

ユーザー定義のポートには変数入力が必要なため、テンプレートファイルを使用して設定内容を保存します。モジュールで、**templates** という名前のディレクトリーを作成し、新しいディレクトリーに **myserver.conf.erb** という名前のファイルを追加します。ファイルに以下のコンテンツを追加します。

```
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>
```

このテンプレートは、Apache Web サーバー設定の標準構文に従います。唯一の相違点は、モジュールから変数を注入する際に Ruby のエスケープ文字が含まれる点です。たとえば、Web サーバーポートを指定するために使用する **httpd_port** などがあります。

fqdn が追加されている点にも注意してください。これは、システムの完全修飾ドメイン名を保存する変数です。これは、システムファクトとして知られています。システムファクトは、各該当システムの Puppet カタログを生成する前に各システムから収集します。Puppet は **facter** コマンドを使用して、これらのシステムファクトを収集します。また、**facter** を実行してこれらのファクトのリストを表示することもできます。

httpd.pp マニフェストを編集し、強調表示されている行を追加します。

```
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",
  }
}
```

これにより、以下の操作が実行されます。

- サーバー設定ファイル **/etc/httpd/conf.d/myserver.conf** のファイルリソース宣言を追加します。
- **notify ⇒ Service["httpd"]** 属性を使用して、設定ファイルと **httpd** サービスとの関係を追加します。これにより、設定ファイルへの変更の有無が確認されます。ファイルが変更された場合には、Puppet によりサービスが再起動されます。
- このファイルを追加する前に、**httpd** パッケージがインストールされていることを確認します。
- この **/etc/httpd/conf.d/myserver.conf** ファイルの **content** は、以前に作成した **myserver.conf.erb** テンプレートです。
- 2 つ目のファイルリソース宣言を追加します。これにより、Web サーバーの **/var/www/myserver/** ディレクトリーが作成されます。

また、メインのマニフェストファイルに **httpd_port** パラメーターを含める必要があります。**init.pp** マニフェストを編集し、以下の太字のテキストを追加します。

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

これにより、**httpd_port** パラメーターをデフォルト値 8120 に設定します。この値は Satellite Server で上書きできます。

puppet apply コマンドを再度実行して、モジュールへの変更をテストします。

```
# 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)
...
```

これらの出力通知メッセージでは、設定ファイルと Web サーバーディレクトリーの作成が表示されます。

2.7. ファイアウォールの設定

ユーザーが、Web サーバーでホストされるページにアクセスできるように、Web サーバーにはオープンポートが必要です。問題は、Red Hat Enterprise Linux のバージョンによってファイアウォールの制御方法が異なることです。Red Hat Enterprise Linux 6 以前では、**iptables** を使用します。Red Hat Enterprise Linux 7 では、**firewalld** を使用します。

この決定は、条件付きロジックとシステムファクトを使用して Puppet が処理するものです。この手順では、オペレーティングシステムを確認し、適切なファイアウォールコマンドを実行するステートメントを追加します。

mymodule::httpd クラス内に以下のコードを追加します。

```
  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'],
    }
  }
```

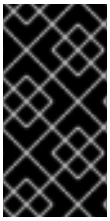
このコードは、以下を実行します。

- **operatingsystemmajrelease** ファクトを使用して、オペレーティングシステムが Red Hat Enterprise Linux 6 か 7 であるかを判断します。
- Red Hat Enterprise Linux 6 を使用している場合は、**iptables** と **iptables-save** を実行する実行ファイル(**exec**)リソースを宣言して永続的なファイアウォールルールを追加します。**httpd_port** 変数は、開放するポートを定義するためにインラインで使用されます。**exec** リソースの完了後に、**iptables** サービスの更新をトリガーします。これを行うには、**subscribe** 属性が含まれるサービスリソースを定義します。この属性は、別のリソースに変更があるかどうかを確認し、ある場合は更新を実行します。この場合は、**iptables** の実行可能なリソースを確認します。
- Red Hat Enterprise Linux 7 を使用している場合は、**firewall-cmd** を実行する同様の実行可能ファイルリソースを宣言して、永続的なファイアウォールルールを追加します。また、**httpd_port** 変数は、開放するポートを定義するためにインラインで使用されます。**exec** リソースの完了後に、**firewalld** サービスの更新をトリガーしますが、**subscribe** 属性は **firewall-cmd** 実行リソースを参照します。
- ファイアウォールの実行可能リソースのコードには、**refreshonly ⇒ true** および **subscribe ⇒ Package['httpd']** 属性が含まれます。これにより、ファイアウォールコマンドは **httpd** のインストール後にのみ実行されます。これらの属性がないと、その後を実行して同じファイアウォールルールの複数のインスタンスが追加されます。

puppet apply コマンドを再度実行して、モジュールへの変更をテストします。以下の例では、Red Hat Enterprise Linux 6 をテストしています。

```
# 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
...
```

これらの出力通知メッセージには、**subscribe** 属性の結果としてのファイアウォールルール作成の実行とそれに続くサービスの更新が表示されます。



重要

この設定は、条件付きステートメントの使用例としてのみ機能します。今後、複数のファイアウォールルールをシステムで管理する場合は、ファイアウォール用にカスタムリソースを作成することが推奨されます。実行可能リソースを使用して、多くの Bash コマンドを常にチェーンすることは推奨されていません。

2.8. SELINUX の設定

SELinux は、デフォルトで HTTP サーバーへの標準以外のアクセスを制限します。カスタムポートを定義する場合は、SELinux がアクセスを付与できる設定を追加する必要があります。

Puppet には、ブール値やモジュールなどの一部の SELinux 機能を管理するリソースタイプが含まれます。ただし、**semanage** コマンドを実行してポート設定を管理する必要があります。このツールは **policycoreutils-python** パッケージの一部で、Red Hat Enterprise Linux システムにはデフォルトではインストールされていません。

mymodule::httpd クラス内に以下のコードを追加します。

```
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,
}
```

このコードは、以下を実行します。

- **require ⇒ Package['policycoreutils-python']** 属性は、コマンドの実行前に **policycoreutils-python** がインストールされていることを確認します。
- Puppet は **httpd_port** を変数として使用して **semanage** を実行し、Apache のリッスンが許可される TCP ポートのリストにカスタムポートを追加します。
- **before ⇒ Service ['httpd']** により、**httpd** サービスを起動する前にこのコマンドが実行されます。SELinux コマンドの前に **httpd** が起動すると、SELinux はポートへのアクセスを拒否し、サービスの起動に失敗します。
- SELinux 実行可能ファイルリソースのコードには **refreshonly ⇒ true** および **subscribe ⇒**

Package['httpd'] の属性が含まれます。これにより、SELinux コマンドは、**httpd** のインストール後にのみ実行されます。これらの属性がないと、その後実行に失敗します。これは、ポートがすでに有効化されていることを SELinux が認識し、エラーを報告するためです。

puppet apply コマンドを再度実行して、モジュールへの変更をテストします。

```
# 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 は、最初に **policycoreutils-python** をインストールしてから、**httpd** サービスを起動する前にポートアクセスを設定します。

2.9. WEB ホストへの HTML ファイルのコピー

これで、HTTP サーバー設定が完了しました。これにより、Web ベースのアプリケーションをインストールするプラットフォームが提供され、Puppet も設定することができます。ただし、この例では、簡単なインデックス Web ページのみを Web サーバーにコピーします。

files ディレクトリーに **index.html** という名前のファイルを作成します。以下の内容をファイルに追加します。

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

manifests ディレクトリーに **app.pp** という名前のマニフェストを作成します。以下の内容をファイルに追加します。

```
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"],
  }
}
```

この新しいクラスには、単一のリソース宣言が含まれます。この宣言は、モジュールのファイルディレクトリのファイルを、Puppet Serverからシステムにコピーし、そのパーミッションを設定します。さらに、**require** 属性は **mymodule::httpd** クラスが、**mymodule::app** を適用する前に設定を正常に完了させます。

最後に、メインの **init.pp** マニフェストにこの新しいマニフェストを追加します。

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

puppet apply コマンドを再度実行して、モジュールへの変更をテストします。出力は以下のようになります。

```
# 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)
...
```

この出力通知メッセージは、**index.html** ファイルが Web サーバーにコピーされることを示しています。

2.10. モジュールの最終処理

モジュールの使用準備が整いました。モジュールを Red Hat Satellite 6 が使用するアーカイブにエクスポートするには、以下のコマンドを入力します。

```
# puppet module build mymodule
```

これにより、**mymodule** ディレクトリーの内容が含まれる **mymodule/pkg/mymodule-0.1.0.tar.gz** にアーカイブファイルが作成されます。このモジュールを Red Hat Satellite 6 Server にアップロードし、独自の HTTP サーバーをプロビジョニングします。

変更が必要な場合は、**puppet module build** コマンドを使用して、モジュールディレクトリーのファイルを編集し、モジュールを再構築します。この変更は、モジュールのバージョンが増加する場合にのみ Satellite に反映されます。バージョン番号を増やすには、**/etc/puppetlabs/code/modules/mymodule/metadata.json** ファイルを編集し、モジュールを再ビルドします。Satellite Server で新しいバージョンをアップロードして公開します。

第3章 RED HAT SATELLITE 6 への PUPPET モジュールの追加

Puppet モジュールは、Red Hat Satellite 6 の Satellite 製品の一部を構成します。つまり、カスタム製品を作成し、その製品の基礎を構成するモジュールをアップロードする必要があります。たとえば、カスタム製品は、HTTP サーバー、データベース、カスタムアプリケーションの設定に必要な一連の Puppet モジュールで構成される場合があります。カスタム製品には、アプリケーションに適用される RPM パッケージを含むリポジトリーを含めることもできます。

3.1. カスタム製品の作成

Puppet モジュールを追加する最初の手順は、カスタム製品を作成することです。

カスタム製品の作成

1. Red Hat Satellite 6 Server にログインします。
2. **コンテンツ** > **製品** に移動します。
3. **製品の作成** をクリックします。
4. カスタム製品に **名前** を指定します。この例では、**MyProduct** を名前として使用します。
5. **ラベル** フィールドには、**Name** に基づいてラベルが自動的に入力されます。
6. 必要な場合は **GPG キー**、**同期プラン**、および **説明** を指定します。この例では、これらのフィールドを空白のままにします。
7. **保存**をクリックします。

CLI をご利用の場合

カスタム製品を作成するには、以下のコマンドを入力します。

```
# hammer product create \
--name "MyProduct" \
--organization "Default Organization"
```

3.2. カスタム製品での PUPPET リポジトリーの作成

次の手順では、カスタム製品に Puppet リポジトリーを作成します。

カスタム Puppet リポジトリーの作成

1. **製品** ページで、以前に作成したカスタム製品 (**MyProduct**) をクリックします。
2. **リポジトリー** のサブタブに移動します。
3. **新規リポジトリー** をクリックします。
4. リポジトリーに **名前** を指定します。この例では **MyRepo** という名前を使用します。
5. **ラベル** フィールドには、**Name** に基づいてラベルが自動的に入力されます。
6. リポジトリー **タイプ** として **puppet** を選択します。

7. URL フィールドは空白のままにします。このフィールドはリモートリポジトリーに使用されますが、このケースでは、Satellite 6 が独自のリポジトリーを作成します。

8. **保存**をクリックします。

CLI をご利用の場合

カスタム製品に Puppet リポジトリーを作成するには、以下のコマンドを入力します。

```
# hammer repository create \
--organization "Default Organization" \
--product "MyProduct" \
--name "MyRepo" \
--content-type puppet
```

3.3. PUPPET モジュールのリポジトリーへのにアップロード

次に、**mymodule** モジュールを新たに作成されたリポジトリーにアップロードし、カスタムの Product に追加します。

1. 新規作成されたリポジトリーの **Name** をクリックします。

2. **Upload Puppet Module** セクションで **Browse** をクリックし、**mymodule** アーカイブを選択します。

3. **アップロード** をクリックします。

このリポジトリーに他のモジュールをアップロードすることができます。この例では、**mymodule** モジュールをアップロードすることのみが必要になります。

CLI をご利用の場合

Puppet モジュールをリポジトリーにアップロードするには、以下のコマンドを入力します。

```
# hammer repository upload-content \
--organization "Default Organization" \
--product "MyProduct" \
--name "MyRepo" \
--path path_to_the_module
```

3.4. リポジトリーからの PUPPET モジュールの削除

今後カスタムリポジトリーから冗長モジュールを削除する場合は、**Puppet モジュールの管理** 機能を使用します。

1. **Products** ページで、削除するモジュールが含まれるカスタムの Product をクリックします。

2. 削除するモジュールが含まれるリポジトリーの **Name** をクリックします。

3. **Manage Puppet Modules** をクリックします。画面には、リポジトリーに含まれる Puppet モジュールの一覧が表示されます。

4. 削除するモジュールを選択します。

5. **Puppet モジュールの削除** をクリックします。

CLI をご利用の場合

リポジトリーから Puppet モジュールを削除するには、以下のコマンドを実行します。

```
# hammer repository remove-content \
--organization "Default Organization" \
--product "MyProduct" \
--name "MyRepo" \
--ids array of content IDs to remove
```

3.5. GIT リポジトリーからの PUPPET MODULES の追加

モジュールを手動でアップロードする代わりに、Red Hat Satellite 6 には **pulp-puppet-module-builder** と呼ばれるユーティリティーが含まれています。このツールは、モジュールのセットが含まれるリポジトリーを確認し、モジュールをビルドし、それらを Satellite 6 が同期する構造で公開します。これにより、Git でモジュール開発を管理する効率的な方法が提供され、Satellite 6 ワークフローに追加されます。

Modulefile の使用は、Puppet バージョン 3 以降は非推奨となりました。Puppet バージョン 3.X の使用時に Modulefile を含む Puppet モジュールをビルドすると、非推奨の警告が表示されます。Satellite 6.7 には、モジュールファイルデータを無視する Puppet バージョン 4 が含まれます。Modulefile のデータは **metadata.json** ファイルに移動する必要があります。以下のように **metadata.json** ファイルを使用するようにモジュールを変換することができます。

1. **puppet module build** Module_Directory を一度実行します。
2. Modulefile を削除します。
3. 更新された **metadata.json** ファイルをリビジョンコントロールリポジトリーに確認します。

注記

pulp-puppet-tools パッケージを使用して、他のマシンに **pulp-puppet-module-builder** ツールをインストールすることもできます。

一般的な方法の 1 つは、Satellite 6 Server 自体でユーティリティーを実行し、ローカルディレクトリーに公開することです。

Git リポジトリーのローカルディレクトリーへの公開

1. Satellite Server でディレクトリーを作成し、モジュールを同期します。

```
# mkdir /var/www/puppet-modules
# chmod 755 /var/www/puppet-modules
```

モジュールを **/var/www/** ディレクトリーに保存すると、SELinux はリポジトリーの同期をブロックします。別のディレクトリーを使用する必要がある場合は、**httpd_sys_r_content_t** または **pulp_tmp_t** SELinux タイプを使用できます。**httpd_sys_r_content_t** SELinux タイプを使用すると、Web サーバーはファイルを読み取ることができます。SELinux ファイルタイプの設定に関する詳細は、『SELinux ユーザーおよび管理者のガイド』を参照してください。

2. **pulp-puppet-module-builder** を実行して Git リポジトリーをチェックアウトします。

```
# pulp-puppet-module-builder --output-dir=/var/www/puppet-modules \
--url=git@mygitserver.com:mymodules.git --branch=develop
```

これは、Git リポジトリーの **develop** ブランチを **git@mygitserver.com:mymodules.git** からチェックアウトし、モジュールを**/var/www/puppet-modules/** に公開します。

HTTP サーバーにモジュールを公開する場合も同様です。

Web サーバーへの Git リポジトリーの公開

1. Web サーバーのディレクトリーを作成し、モジュールを同期します。

```
# mkdir /var/www/html/puppet-modules
# chmod 755 /var/www/html/puppet-modules
```

2. **pulp-puppet-module-builder** を実行して Git リポジトリーをチェックアウトします。

```
# pulp-puppet-module-builder \
--output-dir=/var/www/html/puppet-modules \
--url=git@mygitserver.com:mymodules.git --branch=develop
```

これは、Git リポジトリーの **develop** ブランチを **git@mygitserver.com:mymodules.git** からチェックアウトし、モジュールを**/var/www/html/puppet-modules/**に公開します。

Satellite 6 Web UI で、URL が公開されたモジュールの場所に設定した新規リポジトリーを作成します。

Git からの Puppet モジュールのリポジトリーの作成

1. **製品** ページで、以前に作成したカスタム製品 (**MyProduct**) をクリックします。
2. **リポジトリー** のサブタブに移動します。
3. **新規リポジトリー** をクリックします。
4. リポジトリーに **名前** を指定します。この例では、**MyGitRepo** という名前を使用します。
5. **ラベル** フィールドには、**Name** に基づいてラベルが自動的に入力されます。
6. リポジトリー **タイプ** として **puppet** を選択します。
7. URL フィールドで、以前に定義した場所を設定します。たとえば、Satellite 6 Server のローカルディレクトリーは、**file:**// プロトコルを使用します。

```
file:///modules
```

リモートリポジトリーは **http:**// プロトコルを使用します。

```
http://webserver.example.com/modules/
```

8. **保存**をクリックします。
9. **Sync Now** をクリックしてリポジトリーを同期します。

Git リポジトリーの Puppet モジュールが Satellite 6 Server に含まれているようになりました。

3.6. コンテンツビューの公開

Puppet モジュールの使用準備できた最後の手順は、コンテンツビューの一部として公開することです。このモジュールを既存のビューに追加できますが、この例では新しいビューを作成します。

コンテンツビューの公開

1. **Content** > **Content Views** に移動します。
2. **+ Create New View** をクリックします。
3. ビューに **Name** を指定します。この例では、**MyView** を名前として使用します。
4. **ラベル** フィールドには、**Name** に基づいてラベルが自動的に入力されます。
5. **Composite View** が選択されていないことを確認します。
6. **保存**をクリックします。
7. 新しく作成したビューの **Name** を選択します。
8. **Content** > **Repositories** に移動します。
9. ベース Red Hat Enterprise Linux Server RPM コレクションと、同じバージョンの Red Hat Satellite Tools 6.7 RPM コレクションなど、必要な Red Hat Enterprise Linux リポジトリーを追加します。Tools RPM コレクションには、プロビジョニングされたシステムにリモート Puppet 設定を設定するパッケージが含まれています。
10. **Puppet Modules** に移動します。
11. **新しいモジュールの追加** をクリックします。
12. モジュールまでスクロールし、**バージョンの選択** をクリックします。
13. モジュールバージョンの **最新バージョンを使う** にスクロールし、**バージョンの選択** をクリックします。
14. モジュールはコンテンツビューに含まれるようになりました。**バージョン** に移動して、コンテンツビューの新しいバージョンを公開し、プロモートします。
15. **新規バージョンの公開** をクリックします。**新規バージョンの公開** ページで、**保存** をクリックします。これにより、モジュールが含まれるコンテンツビューが公開されます。
16. 新しいバージョンのビューまでスクロールし、**プロモート** をクリックします。ライフサイクル環境を選択して **バージョンのプロモート** をクリックします。これにより、ビューは選択したライフサイクル環境の一部となります。

これで、コンテンツビューが公開されました。コンテンツビューの作成の一環として、Red Hat Satellite 6 は、プロビジョニングプロセスで使用する新規の Puppet 環境を作成します。この Puppet 環境にはモジュールが含まれます。この新規の Puppet 環境は、**設定** > **環境** ページで表示できます。

3.7. PUPPET 環境

Puppet 環境は、特定の Puppet モジュールのセットに関連付けることができる Puppet エージェントノードの単独のセットとして定義されます。Satellite 6 のコンテキストでは、Puppet 環境は、特定の Puppet モジュールセットに関連付けられる Puppet エージェントを実行しているホストセットと考えることができます。たとえば、**実稼働** 環境に関連付けられているノードは、**実稼働** 環境にあるモジュールにのみアクセスできます。

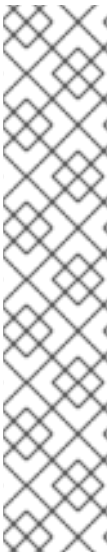
Puppet 環境は、さまざまなタイプのホストから Puppet モジュールを分離するために使用されます。典型的な使用方法は、別の環境にプッシュされる前に、モジュールへの変更を 1 つの環境でテストできるようにすることです。Puppet モジュールには、ファクトと関数のほか、ホストに割り当てることができる 1 つ以上の Puppet クラスを含めることができます。モジュールは環境の一部であり、Puppet クラスはモジュールの一部であることから環境の一部となります。

Red Hat Satellite では、CV に Puppet モジュールがある場合は、コンテンツビュー用に Puppet 環境が自動的に作成されます。自動的に作成される Puppet 環境名には、組織ラベル、ライフサイクル環境、コンテンツビュー名、コンテンツビュー ID が含まれます。例:
KT_Example_Org_Library_RHEL6Server_3

Red Hat Satellite 内での Puppet 環境の作成

1. **設定 → 環境** に移動し、**新規の Puppet 環境** を選択します。
2. 新しい環境に名前を付け、**送信** をクリックして変更を保存します。

環境が Puppet マスターに存在せず、その後にインポートを実行すると、Satellite は環境を削除するように求める点に留意してください。



注記

Puppet は、**Production** 環境内に作成した Puppet 環境に関連付けられているホストグループにホストを登録すると、Puppet CA 証明書の取得に失敗します。ホストグループに関連付けて、適切な Puppet 環境を作成するには、以下の手順を実行します。

1. 手動でディレクトリーを作成して、所有者を変更します。

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

2. **設定** > **環境** に移動し、**環境をインポート** をクリックします。ボタン名には、内部または外部の Capsule の FQDN が含まれます。
3. 作成したディレクトリーを選択し、**更新** をクリックします。

Red Hat Satellite への Puppet 環境のインポート

Satellite は、Puppet マスターに含まれる環境および Puppet モジュールをすべて検出し、自動的にインポートすることができます。これを行うには、**設定** > **環境** に移動して、**インポート元** ボタンをクリックします。ボタン名には、内部または外部の Capsule の FQDN が含まれます。Satellite は、Capsule から Puppet マスターをスキャンし、検出された変更の一覧を表示します。適用する変更を選択し、**更新** を選択して変更を適用します。

Capsule は、Puppet クラスが 1 つ以上含まれる環境のみを検出する点に留意してください。したがって、少なくとも 1 つのクラスを含む 1 つの Puppet モジュールが Puppet マスターにデプロイされていることを確認してください。

ホストへの Puppet 環境の割り当て

ホスト > **すべてのホスト** に移動し、**ホスト** リストから名前でホストを選択してから **編集** を選択します。**ホスト** タブで、ホストの **Puppet 環境** を選択できます。環境を選択すると、**Puppet クラス** タブでクラスが絞り込まれ、選択した環境のクラスが区別できるようになります。

ホストのグループに環境を大量に割り当てることができます。**ホスト** 一覧で必要なホストのチェックボックスを選択して、ページの上部にある **Select Action** ドロップダウンメニューから **Change Environment** を選択します。

ホストグループへの Puppet 環境の割り当て

ホストグループを作成するとき、環境フィールドには、関連付けられたコンテンツビュー（存在する場合）によって自動的に作成された環境が事前に入力されています。

デフォルトでは、新規ホストを作成して**Host Group**を選択すると、環境が自動的に事前に選択されます。ユーザーは、新規ホストの環境を変更することができます。

3.8. パラメーター

Red Hat Satellite パラメーターは、ホストのプロビジョニング時に使用するキーと値のペアを定義します。これらは、Puppet のデフォルトのスコープパラメーターの概念に似ています。Puppet でホストを設定する際にパラメーターを定義することができます。

パラメーターのタイプ

Red Hat Satellite には 2 種類のパラメーターがあります。

単純パラメーター

キーと値のペアとの関係を定義する文字列パラメーターです。ユーザー設定で上書きすることはできませんが、Satellite のパラメーター階層に従って上書きされます。グローバル、組織レベル、ロケーションレベル、ドメインレベル、サブネットレベル、オペレーティングシステムレベル、ホストグループ、およびホストのパラメーターは、Red Hat Satellite の単純なパラメーターです。

スマートクラスパラメーター

キーの値を定義するだけではなく、特定のオブジェクトタイプの条件付き引数、検証および上書きを可能にする複雑なパラメーターです。スマートクラスパラメーターを使用すると、Puppet クラスで外部データを取得できるようになります。これらは Puppet クラスで使用され、Puppet 用語でパラメーター化されたクラスと呼ばれます。これらのパラメーターの階層は Web UI で設定できます。

以下のパラメーター階層は単純なパラメーターに対して適用されます。

グローバルパラメーター

Satellite のすべてのホストに適用されるデフォルトのパラメーター。**設定 > グローバルパラメーター** で設定されます。Hammer CLI を使用してグローバルパラメーターを設定するには、以下のコマンドを入力します。

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

組織レベルのパラメーター

特定の組織の全ホストに影響するパラメーター。組織レベルのパラメーターは、グローバルパラメーターを上書きします。**管理 > 組織 > 編集 > パラメーター**で設定されます。Hammer CLI を使用して組織レベルのパラメーターを設定するには、以下のコマンドを入力します。

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

ロケーションレベルのパラメーター

指定の場所内の全ホストに影響するパラメーター。ロケーションレベルのパラメーターは、組織レベルのパラメーターおよびグローバルパラメーターを上書きします。**管理 > ロケーション > 編集 > パ**

ラメーター で設定されます。Hammer CLI を使用してロケーションレベルのパラメーターを設定するには、以下のコマンドを入力します。

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

ドメインパラメーター

指定のドメインのすべてのホストに影響するパラメーター。ドメインパラメーターは、ロケーションレベル以上のパラメーターを上書きします。**インフラストラクチャー > ドメイン > [choose_a_domain] > パラメーター** で設定されます。Hammer CLI を使用してドメインパラメーターを設定するには、以下のコマンドを入力します。

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

サブネットパラメーター

特定のサブネットにプライマリーインターフェースを持つすべてのホストに影響するパラメーター。サブネットパラメーターは、ホストグループのシステムレベル以上のパラメーターを上書きします。**インフラストラクチャー > サブネット > [choose_a_subnet] > パラメーター**で設定されます。Hammer CLI を使用してサブネットパラメーターを設定するには、以下のコマンドを入力します。

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

オペレーティングシステムレベルのパラメーター

指定したオペレーティングシステムを持つすべてのホストに影響するパラメーター。オペレーティングシステムレベルのパラメーターは、ドメイン以上のパラメーターを上書きします。**ホスト > オペレーティングシステム > [choose_an_operating_system] > パラメーター** で設定されます。Hammer CLI を使用してオペレーティングシステムのパラメーターを設定するには、以下のコマンドを入力します。

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

ホストグループパラメーター

指定のホストグループ内の全ホストに影響するパラメーター。ホストグループのパラメーターは、オペレーティングシステムレベル以上のパラメーターを上書きします。**設定 > ホストグループ > [choose_a_host_group] > パラメーター** で設定されます。Hammer CLI を使用してホストグループのパラメーターを設定するには、以下のコマンドを入力します。

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

ホストパラメーター

特定のホストに影響するパラメーター。以前に継承したパラメーターはすべてパラメーターサブタブに表示され、上書きすることができます。**ホスト > すべてのホスト > 編集 > パラメーター**で設定されます。Hammer CLI を使用してホストパラメーターを設定するには、以下のコマンドを入力します。

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

パラメーターと Puppet クラスの使用

Red Hat Satellite では、Puppet クラスで使用するために 2 つの方法でホストの Puppet マスターに値を指定することができます。

スマート変数

スマートクラスパラメーターを持たないクラス用に、グローバルパラメーターを Puppet マスターにキーと値の形式で提供するためのツールです。Puppet マニフェストのパラメーター値の上書きを有効にします。これらは、クラスにスマートクラスパラメーターがない場合や、グローバルパラメーターを必要とする特別なケースにおける使用を目的としています。これらは、階層コンテキストまたはユーザーが適用できるさまざまな条件に応じて、複数の値の候補を持つことができます。これらは、Puppet がパラメータ化されたクラスを持つ前から存在していたもので、現在は後方互換性を保つため、または検証が必要なグローバルパラメータの使用のため、特定のパペットクラスでのみ使用するため、および文字列以外のタイプのために（それ以外の場合は単純なパラメーターを使用できます）。

パラメーター化クラス

スマートクラスパラメーターを含む Puppet クラス。クラスは Puppet マスターからインポートされ、パラメーター名 (例 **$::name** (推奨) または **$name**) は、クラスを作成した人によって定義され、変更することはできません。グローバルではなく、特定のクラスの変数の値を決定できます。

設定したパラメーターは各ホストの対応する YAML ファイルに含まれ、Puppet マスターに送信されます。YAML ファイルは、特定のホストのページにある Web UI で表示できます。**/etc/foreman/settings.yaml** 設定ファイルは、次に **satellite-installer** コマンドを実行する際に上書きされるため、手動で変更しないでください。

重要

Satellite 6 では、パラメーター化されたクラスサポートがデフォルトで有効になっています。有効になっていることを確認するには、**管理 > 設定** に移動して **Puppet** タブを選択し、**ENC のパラメーター化されたクラス** が **True** に設定されていることを確認します。

3.9. スマートクラスパラメーターの設定

以下の手順では、クラス内でパラメーターを設定します。パラメーターが含まれるクラスは、パラメーター化されたクラスと呼ばれます。

スマートクラスパラメーターは、すべての組織で共通しています。**edit_external_parameters** パーミッションを持つユーザーなら誰でも、これらのパラメーターを編集できます。スマートクラスパラメーターを編集するパーミッションを制限する場合は、KCS のソリューション記事「Restrict permissions to edit puppet classes and their smart class parameters which are common between multiple organizations」を参照してください。

スマートクラスパラメーターの設定

1. **設定 > Puppet クラス**をクリックします。
2. **パラメーター** 列に示されるパラメーターを含むクラスを一覧から選択します。

3. **スマートクラスパラメーター** タブをクリックします。これにより、新しい画面が表示されます。左側のセクションには、クラスがサポートするパラメーターの候補の一覧が記載されています。右側のセクションには、選択したパラメーターの設定オプションが含まれます。

4. 左側の一覧からパラメーターを選択します。

5. **説明** テキストボックスを編集して、プレーンテキストのメモを追加します。

6. **上書き** を選択して、Satellite がこの変数を制御できるようにします。このチェックボックスを選択しないと、Satellite は新しい変数を Puppet に渡しません。

7. 渡すデータの **パラメータータイプ** を選択します。これは文字列であることが最も一般的ですが、他のデータタイプもサポートされています。

8. ホストが一致しない場合に、Puppet マスターに送信されるパラメーターの **デフォルト値** を入力します。

9. オプション:**省略** を選択して、上書きが一致しない限り、値を Puppet マスターに**送信しません**。

10. オプション: 作業中に表示したくないデータがフィールドに含まれる場合は、**非表示の値** を選択します。

11. **オプションの入力バリデーター** セクションを使用して、パラメーターに許可される値を制限します。**バリデータータイプ** (コンマ区切りの値の **リスト** または正規表現の **regexp** のいずれか) を選択し、**バリデータールール** フィールドに許可される値または正規表現コードを入力します。

12. **属性の優先順位付け** セクションは、**上書き** オプションが選択されている場合に表示されます。これにより、条件付き引数に基づいて特定のホストの値を上書きするためのオプションが提供されます。属性タイプとその値は matcher として知られています。

13. **順位** フィールドで、リスト内のエントリーを配置することにより、matcher に対してホスト属性またはファクトを評価する優先順位を設定します。Facter にある属性で、デフォルトのリストのホストの属性と混同しない属性を追加することができます。
matcher 間で **AND** 論理条件を作成し、複数の属性を matcher キーとして使用する場合は、以下の形式でコンマ区切りのリストとして 1 行に配置します。

```
location,environment
```

注記

BZ#1772381 が解決されるまで、**順位** のツールチップには、複数の属性を matcher キーとして使うことによる間違った設定例が表示される点に留意してください。

14. **Matcher の追加** をクリックして条件付き引数を追加します。一致する属性は、**順位** リストのエントリーに対応しています。matcher が設定されていない場合は、上書き機能にデフォルト値のみを使用できます。

15. **属性タイプ** リストから属性を選択します。

16. 属性の横にあるフィールドに、属性の文字列を入力します。

17. **値** フィールドで、必要な値を入力します。

動的データは、Embedded Ruby (ERB) テンプレート構文の **値** フィールドでパラメーターおよび Puppet ファクトを使用することで可能です。たとえば、Puppet ファクトを値の一部として使用するには、以下のコマンドを実行します。

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

利用可能な Puppet ファクトを一覧表示するには、**監視 > ファクト** に移動します。

ERB 構文の詳細は、『**ホストの管理**』の「テンプレート作成の参照」を参照してください。

18. **送信** をクリックします。

単一の属性を matcher キーとして設定

単一の属性を matcher キーとして設定できます。

たとえば、**Default_Location** のロケーション内の任意のホストの Puppet マスターにパラメーターの **test** 値を指定するには、以下の手順を実行します。

1. **順位** フィールドに **location** を追加します。
2. **Matcher の追加** をクリックし、**属性タイプ** リストから **location** を選択します。
3. 属性の横にあるフィールドに **Default_Location** を入力します。
4. **値** フィールドに **test** を入力します。
5. **送信** をクリックします。

複数の属性を matcher キーとして設定

複数の属性を matcher として設定できます。

たとえば、**Default_Location** のロケーションおよび開発環境内の任意のホストの Puppet マスターにパラメーターの **test** 値を指定するには、以下の手順を実行します。

1. **順位** フィールドに **location,environment** を追加します。
2. **Matcher の追加** をクリックし、**属性タイプ** リストから **location,environment** を選択します。
3. 属性の横にあるフィールドに **Default_Location,development** を入力します。
4. **値** フィールドに **test** を入力します。
5. **送信** をクリックします。

一致の優先順位の概要

Satellite は、ホストを一致させる場合、以下の優先順位を使用します。

1. Satellite は、ホスト属性で matcher を検索します。
2. ホスト属性に一致するものがない場合には、Satellite はホストパラメーターの matcher を検索します。これは、パラメーター階層に従って継承されます。
3. ホストパラメーターに一致するものがない場合には、Satellite はホストのファクトの matcher を検索します。

3.10. 親ホストグループを使用したファクトの適用

親ホストグループを使用して、ファクトを複数のホストグループのメンバーに適用することができます。

前提条件

- ホストが割り当てられたホストグループがある。
- ホストグループが割り当てられた親ホストグループがある。

複数のホストグループのメンバーへのファクトの適用

1. Satellite Web UI で、**設定** > **クラス** に移動します。
2. 使用する Puppet クラスを選択します。
3. **スマートクラスパラメーター** タブをクリックします。
4. 左側の一覧から上書きするパラメーターを選択します。
5. オプション: **説明** フィールドを編集して、プレーンテキストのメモを追加します。
6. **上書き** を選択して、Satellite がこの変数を制御できるようにします。
7. 渡すデータの **パラメータータイプ** を選択します。これは文字列であることが最も一般的ですが、他のデータタイプもサポートされています。
8. オプション: ホストが一致しない場合に、Puppet マスターに送信されるパラメーターの **デフォルト値** を入力します。
9. オプション: **省略** を選択して、上書きが一致しない限り、値を Puppet マスターに送信しません。
10. オプション: 作業中に表示したくないデータがフィールドに含まれる場合は、**非表示の値** を選択します。
11. オプション: **オプションの入力バリデーター** セクションを使用して、パラメーターに許可される値を制限します。**バリデータータイプ** (コンマ区切りの値の リストまたは正規表現の **regexp** のいずれかを選択し、**バリデータールール** フィールドに許可される値または正規表現コードを入力します。
12. **属性の優先順位付け** エリアで、リスト内のエントリーを配置することにより、matcher に対してホスト属性またはファクトを評価する優先 **順位** を設定します。デフォルトのリストに追加できます。matcher 間で **AND** 論理条件を作成するには、コンマ区切りのリストとして matcher を 1 行に配置します。
13. **Specify Matchers** エリアで、**Add Matcher** をクリックして条件付き引数を追加します。
14. **Attribute type** を **hostgroup** に設定します。
15. **=** 記号の後に、一致するホストグループを入力します。この例では、親ホストグループになります。
16. **Value** フィールドには、親ホストグループに属するコンテンツホストに送信する値を入力します。

17. **送信** をクリックします。
親ホストグループがホストグループの階層の一部である場合は、matcher 値に対してすべての親(例: **top_host_group/intermediate_host_group/parent_host_group**)を入力します。

3.11. 複数のカスタムファクトの使用

以下は、Puppet Smart クラスパラメーター matcher で、複数のカスタムファクトを作成および使用する例です。

前提条件

- Puppet 環境をインポートして、Satellite にスマートクラスパラメーターがある Puppet モジュールがある。
- Satellite にプロビジョニングして登録されているクライアントです。
- Satellite でプロビジョニングされていないクライアントの場合は、「設定の既存クライアントへの適用」 の説明に従って、クライアントが設定されていることを確認してください。

1. クライアントで、2 つ以上のカスタムファクトを作成し、それらのファクトに値を割り当てます。以下は例になります。

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

2. クライアントで、ファイルパーミッションを設定します。

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

3. クライアントで、ファクトとそれに対応する値を確認します。

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

4. Satellite Web UI で以下を実行します。
   a. **Configure** > **Classes** に移動し、設定する Puppet クラスを選択します。
   b. **Smart Class Parameter** タブをクリックして、上書きするパラメーターを選択します。
   c. **デフォルトの動作** エリアで、**上書き** チェックボックスを選択します。
   d. **属性の優先順位付け** エリアの **順位** フィールドに、サンプルファクトをリストの最後に追加します。2 つのファクト間で AND 論理条件を作成するには、コンマ区切りのリスト **example_fact1,example_fact2** として追加します。
   e. **Matcher の追加** を選択します。
   f. **Attribute type** メニューから **example_fact1,example_fact2** を選択し、**=** 記号の後に **myfact1,myfact2** を入力します。

g. **Value** フィールドには、2 つの属性とその値が一致した時にコンテンツホストに送信する値を入力します。

h. **送信** をクリックします。

5. クライアントで、ファクトをクライアントから Puppet マスターに送信します。

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

6. Satellite Web UI で以下を実行します。

a. **Hosts** > **Content Hosts** に移動し、コンテンツホストの名前を選択します。

b. **YAML** をクリックし、**classes** セクションを見つけます。パラメーターに、必要な値があることを確認します。

3.12. スマート変数の設定

以下の手順では、Puppet クラスで値を上書きできるようにスマート編集を設定します。

スマート変数の設定

1. **設定** > **Puppet クラス**をクリックします。
2. 一覧からクラスを選択します。
3. **スマート変数** タブをクリックします。これにより、新しい画面が表示されます。左側のセクションには、クラスがサポートするパラメーターの候補の一覧が記載されています。右側のセクションには、選択したパラメーターの設定オプションが含まれます。**変数の追加** をクリックして、新しいパラメーターを追加します。追加しない場合は、左側のリストからパラメーターを選択します。
4. **キー** フィールドにパラメーターの名前を入力します。
5. **説明** テキストボックスを編集して、プレーンテキストのメモを追加します。
6. 渡すデータの **パラメータータイプ** を選択します。これは文字列であることが最も一般的ですが、他のデータタイプもサポートされています。
7. ホストが一致しない場合に、Puppet マスターに送信されるパラメーターの **デフォルト値** を入力します。
8. オプション: 作業中に表示したくないデータがフィールドに含まれる場合は、**非表示の値** を選択します。
9. **オプションの入力バリデーター** セクションを使用して、パラメーターに許可される値を制限します。**バリデータータイプ** (コンマ区切りの値の **リスト** または正規表現の **regexp** のいずれか) を選択し、**バリデータールール** フィールドに許可される値または正規表現コードを入力します。
10. **属性の優先順位付け** セクションは、条件付き引数に基づいて特定のホストの値を上書きするためのオプションを提供します。属性タイプとその値は **matcher** として知られています。

a. リスト内のエントリーを配置することにより、matcher に対してホスト属性またはファクトを評価する優先 **順位** を設定します。デフォルトのリストに追加できます。matcher 間で AND 論理条件を作成するには、それらをコンマ区切りのリストとして 1 行に配置します。

b. **Matcher の追加** をクリックして条件付き引数を追加します。一致する属性は **順位** リストのエントリーに対応している必要があります。matcher が設定されていない場合は、上書き機能にデフォルト値のみを使用できます。
たとえば、Puppet マスターに指定するパラメーターの値が、**server1.example.com** の完全修飾ドメイン名を持つホストの **test** の場合、matcher を **fqdn=server1.example.com** として、**値** を **test** として設定します。

マッチングの優先順位は以下のとおりです。

i. マッチャーがホスト属性の場合はそれを使用します。

ii. 該当する名前を持つ属性がない場合は、一致するホストパラメーター (パラメーターの階層に基づいて継承される) を探します。

iii. それでも一致しなければホストのファクトをチェックします。
Facter にある属性で、ホストの属性と混同しない属性を使用することが推奨されています。ホストの属性は、ホストグループ、ドメイン、組織など、ホストパラメーターまたはホストの関連付けのいずれかになります。matcher は、ホストが1つだけ持つ必要があります。たとえば、ホストは多くの構成グループを持つことができますが、ホストにはロケーションが 1 つしかなく、ロケーションは有効な matcher であるため、構成グループは使用できません。

動的データは、**Embedded Ruby** (ERB) テンプレート構文の **値** フィールドでパラメーターおよび Puppet ファクトを使用することで可能です。たとえば、Puppet ファクトを値の一部として使用するには、以下のコマンドを実行します。

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

利用可能な Puppet ファクトを一覧表示するには、**監視 > ファクト** に移動します。

11. **送信** をクリックして変更を保存します。

ERB 構文の詳細は、『**ホストの管理**』の「テンプレート作成の参照」を参照してください。

3.13. PUPPET マスターからのパラメーター化されたクラスのインポート

以下の手順では、Puppet マスターからパラメーター化されたクラスをインポートします。

注記

パラメーター化されたクラスのインポートは、Puppet モジュールが製品およびコンテンツビューで管理される場合に自動的に実行されます。

パラメーター化されたクラスのインポート

1. Satellite Web UI で、コンテキストメニューの **Any Organization** と **Any Location** を選択します。

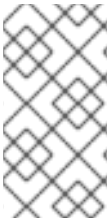
2. **設定 > Puppet クラス** をクリックします。

3. **Import from Host Name** をクリックして、Puppet マスターからパラメーター化されたクラスをインポートします。

4. **Puppet Classes** ページが新規クラスの一覧と共に表示されます。

3.14. スマート変数ツールの使用

スマート変数は、スマートクラスパラメーターを含まない Puppet クラスで使用するために、Puppet マスターにグローバルパラメーターを提供するツールです。同じスマートな Matcher ルールは、スマート変数とスマートクラスパラメーターの両方に使用されます。



注記

スマート変数ツールは、Puppetモジュールがスマートクラスパラメーターをサポートする前の暫定措置として導入されました。スマート変数を使用する特別な理由がない場合は、「スマートクラスパラメーターの設定」 で説明されているように、Red Hat は、代わりにスマートクラスパラメーターを使用することを推奨しています。

スマートクラスパラメーターが導入される前は、グローバルパラメーターを使用するようにPuppetコードを書き直すように求められたパラメーターをオーバーライドしたいユーザー。以下は例になります。

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

上記の例では、**$global_var** は Web UI の スマート変数セクションで設定され、値は "example1" クラスに関連付けられます。Puppetがグローバルスコープを検索するのを制限するために、グローバル変数の前に**::**を付けることが推奨されますが、付けなくても、変数が非グローバル変数になるわけではありません。

スマートクラスパラメーターの導入により、以下の形式を使用できます。

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

上記の例では、**$var** は Web UI の Smart Class Parameters セクションで設定され、値は "example2" クラスに関連付けられます。上記の **class example2($var="default")** のように、クラスヘッダーで定義された変数が表示された場合は、**$var** がクラスパラメーターであることを確認できます。スマートクラスパラメーター関数を使用して、変数を上書きする必要があります。

示されているように、スマート変数には、Puppetコミュニティの標準モジュールではなく、global-namespace パラメーターを使用してカスタム設計されたモジュールが必要です。結果は同じです。上記の例では、テキストは '/tmp/foo' に置かれています。そのため、レガシーモジュールのサポート以外に、スマート変数を使用する理由がなくなりました。

スマート変数はグローバル変数ですが、Puppet クラスに関連付けられ、Satellite で特定の Puppet クラスが割り当てられたホストにのみ送信されます。任意の名前でスマート変数を作成できますが、検証は Satellite で行われませんが、関連する Puppet モジュールにコードに一致する変数がない場合は、スマート変数は使用されません。

Satellite は、Satellite で作成した変数をホスト YAML ファイルに追加します。このファイルは、Web UI で **Hosts** > **All Hosts** に移動して、ホスト名を選択してから **YAML** ボタンをクリックします。Satellite は、ホスト YAML ファイルを **外部ノードの分類子**（ENC）（Puppet マスターに含まれる機能）に送信します。Puppet マスターがホストの ENC にクエリーを実行すると、ENC はホストの状態を記述する YAML ドキュメントを返します。この YAML ドキュメントは Puppet マニフェストから取得したデータに基づいていますが、スマートクラスパラメーターの上書きとスマート変数が適用されます。

スマート変数のホストへの適用

スマート変数は、グローバルパラメーターを含むように以前に変更されたカスタム Puppet モジュールをサポートするためにのみ使用する必要があるため、次の例では、**anothermodule**と呼ばれる簡単な例を使用します。**anothermodule** Puppet マニフェストは以下のとおりです。

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

この例では、**$::content_for_motd** パラメーターの値を指定します。

1. Web UI で、**Configure** > **Classes** に移動します。
2. 一覧から Puppet クラスの名前を選択します。
3. **スマート変数** タブをクリックします。これにより、新しい画面が表示されます。左側のセクションには、以前に作成したパラメーターの一覧（ある場合）が含まれます。右側のセクションには設定オプションが含まれます。
4. **Add Variable** をクリックして、新しいパラメーターを追加します。
5. **Key** フィールドにパラメーターを入力します。この例では、**content_for_motd** です。
6. **Description** テキストボックス（例: **Testing /tmp motd Text**）を編集します。
7. 渡すデータの **パラメータータイプ** を選択します。**string** を選択します。
8. パラメーターに **デフォルト値** を入力します。たとえば、**No Unauthorized Use** はありません。
9. **オプションの入力バリデーター** セクションを使用して、パラメーターに許可される値を制限します。**バリデータータイプ** (コンマ区切りの値の **リスト** または正規表現の **regexp** のいずれか) を選択し、**バリデータールール** フィールドに許可される値または正規表現コードを入力します。
10. **属性の優先順位付け** セクションを使用して、ホスト属性またはファクトが matcher に対して評価される優先順位を設定します (以下に設定)。リストでエントリーを再配置し、デフォルトのリストに追加できます。matcher 間で AND 論理条件を作成するには、matcher の名前をコンマ区切りのリストとして 1 行に配置します。
11. **Matcher の指定** セクションで、**Matcher の追加** をクリックして条件付き引数を追加します。一致する属性は、上記の **順位** リストのエントリーに対応している必要があります。matcher が設定されていない場合は、デフォルト値のみを使用できます。たとえば、完全修飾ドメイン名が**server1.example.com**のホストに対して、パラメータの値を**This is for Server1**とする場合、Matchを**fqdn=server1.example.com**とし、Valueを**This is for Server1**とします。
12. **送信** をクリックして変更を保存します。

第4章 ホスト情報の保存および維持

Red Hat Satellite 6 は、アプリケーションの組み合わせを使用して管理対象ホストに関する情報を収集し、それらのホストが望ましい状態で維持されるようにします。これらのアプリケーションには、以下が含まれます。

- Foreman: 物理システムと仮想システムのプロビジョニングおよびライフサイクル管理用です。Foreman は、キックスタートや Puppet モジュールなどの各種の方法を使って、これらのシステムを自動的に設定します。
- Puppet: ホストを設定するためのクライアント/サーバーアーキテクチャーです。Puppet マスター (サーバー) および Puppet エージェント (クライアント) で構成されます。
- Facter: Puppet のシステムインベントリーツール。Facter は、ハードウェアの詳細、ネットワーク設定、OS の種類およびバージョン、IP アドレス、MAC アドレス、SSH キーなど、ホストに関する基本的な情報（ファクト）を収集します。その後、これらのファクトは Puppet マニフェストで変数として利用できます。

Puppet、Facter およびファクトの使用については、以下で詳述します。

4.1. PUPPET アーキテクチャー

Puppet は通常、エージェント/マスター (クライアント/サーバーとしても知られる) アーキテクチャーで実行されます。ここで、Puppet サーバーは重要な設定情報を管理し、管理対象ホスト (クライアント) は独自の設定カタログのみを要求します。Puppet は 2 つの手順でホストを設定します。

- カタログのコンパイル
- カタログの該当ホストへの適用

エージェント/マスターの設定では、Puppet クライアントは Facter によって収集されるファクトや他の情報を Puppet マスターに送信します。Puppet マスターは、これらのファクトに基づいてカタログをコンパイルし、このカタログをクライアントに送信します。クライアントは変更したすべての変更のレポートを送信するか、**--noop** パラメーターを使用した場合は Puppet マスターに結果を送信し、その結果を Foreman に送信します。このカタログは、1 つの特定ホストの必要な状態を記述します。リソース間の依存関係を含む、そのホストで管理するリソースを一覧表示します。エージェントはカタログをホストに適用します。

マスターとエージェント間の通信は、デフォルトで 30 分ごとに実行されます。**runinterval** パラメーターを使用して、**/etc/puppetlabs/puppet/puppet.conf** ファイルに別の値を指定できます。**puppet agent apply** を実行し、通信を手動で開始することもできます。

4.2. FACTER およびファクトの使用

Facter は Puppet のシステムインベントリーツールで、多くの組み込みファクトが含まれています。Facter は、ローカルホストのコマンドラインで実行して、ファクト名と値を表示できます。カスタムファクトで Facter を拡張し、これを使用してホストのサイト固有の詳細を Puppet マニフェストに公開することができます。Facter が提供するファクトを使用して、Puppet で条件式に通知することもできます。

Puppetは、リソースに基づいてシステム状態を判別します。たとえば、**httpd**サービスは常に実行している必要があり、Puppetはそれを処理する方法を知っていることをPuppetに伝えることができます。異なるオペレーティングシステムを管理する場合は、**osfamily** ファクトを使用して条件式を作成し、どのサービスを監視するか、どのパッケージをインストールするかを示すことができま

す。**operatingsystemmajrelease** パラメーターおよび **versioncmp** パラメーターを使用して、同じオペレーティングシステムの異なるバージョンに基づいて条件式を作成できます。以下の例は、Facts を使用した条件式の使用を示しています。

条件式とファクトの使用

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

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

注記

この例では、式 **versioncmp($::operatingsystemmajrelease, '7') >= 0** を使用して、Red Hat Enterprise Linux のバージョン 7 以降をテストします。このテストを実行するために、式 **$::operatingsystemmajrelease >= '7'** を使用しないでください。この関数およびその他のPuppet関数の詳細は、https://docs.puppetlabs.com/references/latest/function.html#versioncmpを参照してください。

また、Puppet は、ファクトが多く動作する他の特別な変数も設定します。詳細は、Special Variables Added by PuppetおよびCore Factsを参照してください。

4.2.1. 特定ホストのファクトの表示

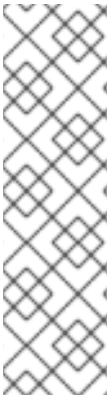
Puppet は Facter の組み込みコアファクトや、Puppet モジュールにあるカスタムまたは外部ファクトにアクセスできます。コマンドライン(**facter -p**)および Web UI(**Monitor > Facts**)から利用可能なファクトを表示できます。ファクトの一覧を参照するか、**Search** ボックスを使用して特定のファクトを検索できます。たとえば、"facts." と入力して、利用可能なファクトの一覧を表示します。

注記

利用可能なファクトの一覧は非常に長くなります。UI は一度に 20 ファクトのみを表示します。詳細の入力時のファクトの段階的フィルターの一覧です。たとえば、"facts.e" と入力して、"e" で始まるすべてのファクトを表示します。

参加者のホストのファクトの表示

1. メインメニューで **Hosts > All Hosts** をクリックしてから、検査するホストの名前をクリックします。
2. **Details** ペインで **Facts** をクリックし、ホストに関する既知のファクトをすべて表示します。



注記

- このページに一覧表示されているファクトについて、**Chart** をクリックし、このファクト名のすべての管理対象ホストにおけるディストリビューションチャートを表示します。
- 検索をブックマークすると、今後簡単に使用できるようになります。検索を改良したら、**Search**ボタンの横にあるドロップダウン矢印をクリックし、**Bookmark this search**をクリックします。ブックマークされた検索は **Search** ドロップダウンリストに表示され、メインメニューの **Administer > Bookmarks**の下にも表示されます。

4.2.2. ファクトに基づくホストの検索

Facter 情報を使用して、特定のホストを検索できます。つまり、**facts.architecture = x86_64** など、特定のファクトの基準に一致するホストをすべて検索できます。

ファクトに基づくホストの検索

1. メインメニューで、**Monitor > Facts** をクリックして、**Fact Values** ページを表示します。
2. **Search** フィールドで、フィルタリングするファクトの名前を入力します。特定の名前、名前と値のペアなどで検索できます。
3. **Search** をクリックして、一致するホストの一覧を取得します。

4.2.3. カスタムファクトのレポート

管理ホストからのカスタム情報の取得は、Red Hat Satellite 6 で完全にサポートされています。このセクションでは、Puppet Forge から取得した Puppet モジュールを使用していますが、原則は Puppet モジュールの他のソースに対して同等に適用されます。

標準の Facter インターフェースを介して報告されるファクトの数は拡張できます。たとえば、モジュールの変数として使用するファクトを収集します。インストールされたパッケージを説明するファクトが利用可能な場合は、このデータを検索し、情報に基づいて設定管理の決定を行うことができるようになります。

ホストにインストールされたパッケージについてのレポートを取得するプロセスは以下のようになります。

- マニフェスト **pkginventory** は Puppet Forge から取得され、ベースシステムに保存されます。
- Puppet モジュールはコンテンツビューに追加され、これはシステムにプロモートされてからそのシステムにデプロイされます。
- システムのファクトは、パッケージ名を使用してクエリーされます。この例では、**hostname** という名前のホストで、認証情報 **username** と **password** を持つSatelliteユーザーを使用している場合、次のAPIクエリは検索文字列 "bash"に一致するファクトを返します。

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

検索は、パッケージバージョンを返します。その後、外部データベースの設定に使用していました。

4.2.3.1. pkginventory Puppet モジュールの追加

pkginventory Puppet モジュールを RRed Hat Satellite Server アプリケーションに追加するには、https://forge.puppetlabs.com/ody/pkginventory から Satellite Server アプリケーションがインストールされているベースシステムにモジュールをダウンロードし、以下の手順に従います。

Puppet モジュールは通常、Puppet Modules という名前のカスタムリポジトリーに保存されます。以下の手順は、その名前のカスタムリポジトリーを作成していることを前提としています。Puppet モジュールのカスタムリポジトリーをまだ作成していない場合は、『**クイックスタートガイド**』の「カスタム製品の作成」を参照してください。

Puppet モジュールのリポジトリーへのにアップロード

1. ベースシステムに Puppet モジュールをダウンロードします。ダウンロードしたモジュールの拡張子は **.tar.gz** になります。
2. **Content** > **Products** をクリックしてから、Puppet モジュールリポジトリーに関連付けられた **Name** フィールドで製品名をクリックします。たとえば、**Custom Products** です。
3. **Repositories** タブで、変更する Puppet モジュールリポジトリーを選択します。たとえば、**Puppet Modules** などです。
4. **Upload Puppet Module** セクションで **Browse** をクリックし、ダウンロードしたモジュールに移動します。
5. **アップロード** をクリックします。

Puppet モジュールをクライアントに配布するには、モジュールはコンテンツビューに適用して公開する必要があります。以下の手順に従って、コンテンツビューにモジュールを追加します。

コンテンツビューへのモジュールの追加

1. **Content** > **Content Views** をクリックして、**Name** メニューからコンテンツビューを選択します。
2. **Puppet モジュール** タブで **Add New Module** をクリックします。インストールされたモジュールの一覧が表示されます。
3. **Actions** コラムから **Select a Version** をクリックし、追加するモジュールを選択します。利用可能なバージョンの表が表示されます。
4. 追加するモジュールのバージョンの横の **バージョンの選択** をクリックします。
5. **Publish New Version** をクリックして、新しいコンテンツビューを作成します。
6. 必要に応じて説明を追加し、**Save** をクリックします。

第5章 設定管理のためのクライアントおよびサーバー設定

Red Hat Satellite 6 の設定プロセスの重要な部分は、内部 Satellite Capsule または外部 Satellite Capsule のいずれかで Puppet クライアント（Puppet エージェントと呼ばれる）が Puppet サーバー（Puppet マスターと呼ばれる）と通信できるようにすることです。本章では、Red Hat Satellite 6 が Puppet マスターと Puppet エージェントの両方を設定する方法を検証します。

5.1. RED HAT SATELLITE SERVER での PUPPET の設定

Red Hat Satellite 6 は、全 Satellite Capsule 上の Puppet マスターの主な設定を制御します。追加の設定は不要であり、これらの設定ファイルを手動で変更しないようにすることが推奨されます。たとえば、メインの **/etc/puppetlabs/puppet/puppet.conf** 設定ファイルには、以下の **[master]** セクションが含まれます。

```
[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 =
```

このセクションには、さまざまな環境の設定の作成に Satellite 6 が使用する変数（**$environment**など）が含まれます。

一部の Puppet 設定オプションが Satellite 6 Web UI に表示されます。**Administer** > **Settings** に移動して、**Puppet** サブタブを選択します。このページには、Puppet 設定オプションのセットと、それぞれの説明が記載されています。

5.2. プロビジョニングされたシステムでのPUPPETエージェントの設定

プロビジョニングプロセスの一環として、Satellite 6 は Puppet をシステムにインストールします。このプロセスでは、選択した Capsule 上の Puppet マスターのエージェントとして Puppet を設定するために **/etc/puppetlabs/puppet/puppet.conf** ファイルもインストールされます。この設定ファイルは、Satellite 6 のプロビジョニングテンプレートスニペットとして保存されます。**Hosts** > **Provisioning templates** に移動し、**puppet.conf** スニペットをクリックして表示します。

デフォルトの **puppet.conf** スニペットには、以下のエージェント設定が含まれます。

```
[agent]
pluginsync      = true
report          = true
ignoreschedules = true
daemon          = false
ca_server       = <%= @host.puppet_ca_server %>
certname        = <%= @host.certname %>
environment     = <%= @host.environment %>
server          = <%= @host.puppetmaster %>
```

このスニペットには、テンプレート変数が含まれています。

- **@host.puppet_ca_server** および **@host.certname** – Puppet 通信のセキュリティーを保護する証明書および認証局。
- **@host.environment** – 設定に使用する Satellite 6 Server の Puppet 環境。
- **@host.puppetmaster** – Puppet マスターを含むホスト。これは、Satellite 6 Server の内部 Capsule または外部の Satellite Capsule のいずれかです。

第6章 クライアントへの設定の適用

この時点で、Satellite 6 Server の Puppet エコシステムが設定され、**mymodule** モジュールが含まれます。ここでは、このモジュールの設定を登録済みのシステムに適用することを目的としています。

6.1. プロビジョニング時のクライアントへの設定の適用

まず、以下の手順で新規ホストの Puppet 設定を定義します。この手順では、サンプルとしてアップロードされた **mymodule** を使用します。

プロビジョニング時のクライアントへの設定の適用

1. **Hosts** > **New host** に移動します。
2. **Host** タブをクリックします。ホストの **Name** を入力し、システムの組織とロケーションを選択します。**ライフサイクル環境** とそのプロモート済みの **コンテンツビュー** を選択します。これは、設定に使用する Puppet 環境を定義します。また、**Capsule の設定** から **Puppet CA** および **Puppet Master** を選択します。選択した Capsule は、設定を制御し、新規ホストのエージェントと通信するサーバーとして機能します。
3. **Puppet Classes** タブをクリックして、**Available Classes** セクションから、適用する設定が含まれる Puppet クラスを選択します。この例では、以下を選択します。
   - **mymodule**
   - **mymodule:httpd**
   - **mymodule:app**
4. **Operating System** タブから必要なオプションを選択します。これらのオプションは、独自の Satellite 6 インフラストラクチャーによって異なります。**Provisioning templates** オプションに **Satellite Kickstart Default** テンプレートが含まれていることを確認してください。このテンプレートには、新規ホストでの Puppet エージェントのインストールコマンドが含まれています。
5. **Parameters** タブをクリックして、Puppet クラスパラメーターにカスタムオーバーライドを提供します。この機能を有効にするには、「スマートクラスパラメーターの設定」 を参照してください。
6. プロビジョニングオプションをすべて完了したら、**Submit** をクリックします。

プロビジョニングプロセスが開始されます。Satellite 6は、**Satellite Kickstart Default**プロビジョニングテンプレートの一部として必要な設定ツールをインストールします。このプロビジョニングテンプレートには、以下が含まれます。

```
<% 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 -%>
```

本セクションでは、以下を実行します。

- Red Hat Satellite Tools 6.7 リポジトリーから **puppet** パッケージをインストールします。
- **/etc/puppetlabs/puppet/puppet.conf** にあるシステムに Puppet 設定スニペットをインストールします。
- Puppet サービスがシステムで実行できるようになります。
- 初めて Puppet を実行し、ノードを初期化します。

プロビジョニングプロセスと設定プロセスが新規ホストで完了したら、Web ブラウザーでユーザー定義のポートでホストにアクセスします。たとえば、**http://newhost.example.com:8120/** に移動します。以下のメッセージがブラウザーに表示されるはずです。

```
Congratulations

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

6.2. 設定の既存クライアントへの適用

Puppet 設定は、Red Hat Satellite 6 でプロビジョニングされていない既存のクライアントに適用されている可能性があります。このような場合には、Red Hat Satellite 6 に登録した後に、既存のクライアントに Puppet をインストールして設定します。

既存のシステムを Red Hat Satellite 6 に登録します。既存ホストの登録に関する情報は、**『ホストの管理』ガイドの「ホストの登録」を参照し** てください。

重要

puppet パッケージは、Red Hat Satellite Tools 6.7 リポジトリーに含まれます。続行する前に、このリポジトリーを有効にしてください。

Puppet エージェントをインストールし、有効にする方法:

1. 端末コンソールを開き、root としてログインします。
2. Puppet エージェントをインストールします。

```
# yum install puppet
```

3. システムの起動時に Puppet エージェントが起動するように設定します。
   - Red Hat Enterprise Linux 6 の場合:

```
# chkconfig puppet on
```

   - Red Hat Enterprise Linux 7 の場合:

```
# systemctl enable puppet
```

Puppet エージェントの設定

1. **/etc/puppetlabs/puppet/puppet.conf** ファイルを変更して Puppet エージェントを設定します。

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

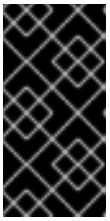
```
[main]
    # The Puppet log directory.
    # The default value is '$vardir/log'.
    logdir = /var/log/puppet

    # 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

environment パラメーターを Satellite Server からホストの Puppet 環境に設定します。Puppet 環境ラベルには、組織ラベル、ライフサイクル環境、コンテンツビュー名、およびコンテンツビュー ID が含まれます。Satellite 6 Web UI で Puppet 環境一覧を表示するには、**Configure** > **Environments** に移動します。

2. ホスト上で Puppet エージェントを実行します。

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

3. Satellite Server Web インターフェースを使って Puppet クライアント用の SSL 証明書に署名します。

   a. Web インターフェースから Satellite Server にログインします。

   b. **Infrastructure** > **Capsules** を選択します。

   c. 必要なホストの右側にある **Certificates** をクリックします。

   d. **Sign** をクリックします。

   e. **puppet agent** コマンドを再度実行します。

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

注記

ホストに Puppet エージェントが設定されている場合は、**All Hosts** に表示されますが、ホストは組織またはロケーションに割り当てられないため、**Any Context** が選択されている場合に限ります。

第7章 設定グループを使用した PUPPET クラスの管理

Red Hat Satellite には、モジュール形式の方法でシステムのグループを構築し、管理するための**Config Groups**と**Host Groups**の概念が含まれています。

設定グループは、ホストとホストグループの設定で使用するビルディングブロックを形成するために作成する Puppet クラスのコレクションです。構成グループは、プロファイルのコミュニティPuppetの概念に類似しています。これは、ビルディングブロックを形成するためのPuppetクラスのコレクションを含むPuppetクラスです。設定グループは Satellite Web UI で作成および管理できます。

ホストグループは、ホストサーバーのコレクションであり、システムを定義するためのコンテナーであり、コンテンツビュー、割り当てられたライフサイクル、および一連のPuppetモジュールを備えています。ホストグループは、特定のビジネスロールでシステムを構築するプロファイルが多数含まれる Puppet クラスである、コミュニティー Puppet の概念に類似しています。ホストグループは、Satellite Web UI、Hammer CLI、および API を使用して作成および管理できます。

設定グループの作成

左側のコンテキストドロップダウンメニューから、**Any Organization** および **Any Location** を選択します。

Configure > **Config groups** に移動します。

New Config Group を選択し、**TestConfGroup** などの名前を入力します。

利用可能なクラス一覧から 1 つ以上の **Puppet クラス** を選択します。

送信 を選択して変更を適用します。

設定グループを作成したら、ホストまたはホストグループの設定時に Puppet クラスタブで選択できるようになります。ホストグループ の作成に関する情報は、『**Red Hat Satellite プロビジョニングガイド**』 の「Satellite Server でのホスト グループの作成」を参照してください。

第8章 RED HAT SATELLITE 6 での PUPPET レポートの確認

Puppet は、設定を適用するたびにレポートを生成します。プロビジョニングしたホストはこのレポートを Red Hat Satellite 6 Server に送信します。ホストの情報ページで、これらのレポートを表示します。

Red Hat Satellite 6 での Puppet レポートの確認

1. **ホスト** > **すべてのホスト** に移動します。
2. 必要なホストの **名前** をクリックします。
3. **レポート** ボタンをクリックします。
4. 表示するレポートを選択します。

各レポートには、各 Puppet リソースのステータスと、ホストに適用された設定が表示されます。