Red Hat Satellite 6.6

Managing Hosts

A guide to managing hosts in a Red Hat Satellite 6 environment.

Last Updated: 2020-01-27

Red Hat Satellite 6.6 Managing Hosts

A guide to managing hosts in a Red Hat Satellite 6 environment.

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

Legal Notice

Abstract

This guide describes how to configure and work with hosts in a Red Hat Satellite environment. Before continuing with this workflow you must have successfully installed a Red Hat Satellite 6 Server and any required Capsule Servers.

Table of Contents

CHAPTER 1. OVERVIEW OF HOSTS IN RED HAT SATELLITE 6

A host is any Red Hat Enterprise Linux client that Red Hat Satellite manages. Hosts can be physical or virtual. Virtual hosts can be deployed on any platform supported by Red Hat Satellite, such as KVM, VMware vSphere, OpenStack, Amazon EC2, Rackspace Cloud Services or Google Compute Engine.

Red Hat Satellite enables host management at scale, including monitoring, provisioning, remote execution, configuration management, software management, and subscription management. You can manage your hosts from the Satellite web UI or from the command line.

In the Satellite web UI, you can browse all hosts recognized by Satellite Server, grouped by type:

- **All Hosts** - a list of all hosts recognized by Satellite Server.
- **Discovered Hosts** - a list of bare-metal hosts detected on the provisioning network by the Discovery plug-in.
- **Content Hosts** - a list of hosts that manage tasks related to content and subscriptions.
- **Host Collections** - a list of user-defined collections of hosts used for bulk actions such as errata installation.

To search for a host, type in the **Search** field, and use an asterisk (*) to perform a partial string search. For example, if searching for a content host named **dev-node.example.com**, click the **Content Hosts** page and type **dev-node*** in the **Search** field. Alternatively, ***node*** will also find the content host **dev-node.example.com.**

WARNING

Satellite Server is listed as a host itself even if it is not self-registered. Do not delete Satellite Server from the list of hosts.

CHAPTER 2. ADMINISTERING HOSTS

This chapter describes creating, registering, administering, and removing hosts.

2.1. CREATING A HOST IN RED HAT SATELLITE

Use this procedure to create a host in Red Hat Satellite.

Procedure

1. In the Satellite web UI, click **Hosts** > **Create Host**.
2. On the **Host** tab, enter the required details.
3. Click the **Ansible Roles** tab, and from the **Ansible Roles** list, select one or more roles that you want to add to the host. Use the **arrow** icon to manage the roles that you add or remove.
4. On the **Puppet Classes** tab, select the Puppet classes you want to include.
5. On the **Interfaces** tab:
    a. For each interface, click **Edit** in the **Actions** column and configure the following settings as required:
        - **Type** – For a Bond or BMC interface, use the **Type** list and select the interface type.
        - **MAC address** – Enter the MAC address.
        - **DNS name** – Enter the DNS name that is known to the DNS server. This is used for the host part of the FQDN.
        - **Domain** – Select the domain name of the provisioning network. This automatically updates the **Subnet** list with a selection of suitable subnets.
        - **IPv4 Subnet** – Select an IPv4 subnet for the host from the list.
        - **IPv6 Subnet** – Select an IPv6 subnet for the host from the list.
        - **IPv4 address** – If IP address management (IPAM) is enabled for the subnet, the IP address is automatically suggested. Alternatively, you can enter an address. The address can be omitted if provisioning tokens are enabled, if the domain does not mange DNS, if the subnet does not manage reverse DNS, or if the subnet does not manage DHCP reservations.
        - **IPv6 address** – If IP address management (IPAM) is enabled for the subnet, the IP address is automatically suggested. Alternatively, you can enter an address.
        - **Managed** – Select this check box to configure the interface during provisioning to use the Capsule provided DHCP and DNS services.
        - **Primary** – Select this check box to use the DNS name from this interface as the host portion of the FQDN.
        - **Provision** – Select this check box to use this interface for provisioning. This means TFTP boot will take place using this interface, or in case of image based provisioning, the script to complete the provisioning will be executed through this interface. Note

that many provisioning tasks, such as downloading RPMs by **anaconda**, Puppet setup in a **%post** script, will use the primary interface.

- **Virtual NIC** – Select this check box if this interface is not a physical device. This setting has two options:
  - **Tag** – Optionally set a VLAN tag. If unset, the tag will be the VLAN ID of the subnet.
  - **Attached to** – Enter the device name of the interface this virtual interface is attached to.

b. Click **OK** to save the interface configuration.

c. Optionally, click **Add Interface** to include an additional network interface. See Chapter 4, *Adding Network Interfaces* for details.

d. Click **Submit** to apply the changes and exit.

6. On the **Operating System** tab, enter the required details. For Red Hat operating systems, select **Synced Content** for **Media Selection**. If you want to use non Red Hat operating systems, select **All Media**, then select the installation media from the **Media Selection** list. You can select a partition table from the list or enter a custom partition table in the **Custom partition table** field. You cannot specify both.

7. On the **Parameters** tab, click **Add Parameter** to add any parameter variables that you want to pass to job templates at run time. This includes all Puppet Class, Ansible playbook parameters and host parameters that you want to associate with the host. To use a parameter variable with an Ansible job template, you must add a **Host Parameter**.
   When you create a Red Hat Enterprise Linux 8 host, you can set system purpose attributes. System purpose attributes define what subscriptions to attach automatically on host creation. In the **Host Parameters** area, enter the following parameter names with the corresponding values. For the list of values, see Configuring system purpose in the *Performing a standard RHEL installation* guide.
   - **syspurpose_role**
   - **syspurpose_sla**
   - **syspurpose_usage**
   - **syspurpose_addons**

8. On the **Additional Information** tab, enter additional information about the host.

9. Click **Submit** to complete your provisioning request.

For CLI Users

To create a host associated to a host group, enter the following command:

```
# hammer host create \
--name "host_name" \
--hostgroup "hostgroup_name" \
--interface="primary=true, \
          provision=true, \
          mac=mac_address, \
          ip=ip_address" \
```

```
--organization "Your_Organization" \
--location "Your_Location" \
--ask-root-password yes
```

This command prompts you to specify the root password. It is required to specify the host's IP and MAC address. Other properties of the primary network interface can be inherited from the host group or set using the **--subnet**, and **--domain** parameters. You can set additional interfaces using the **--interface** option, which accepts a list of key-value pairs. For the list of available interface settings, enter the **hammer host create --help** command.

2.2. EDITING THE SYSTEM PURPOSE OF A HOST

You can edit the system purpose attributes for a Red Hat Enterprise Linux 8 host. System purpose attributes define which subscriptions to attach automatically to this host. For more information about system purpose, see Configuring system purpose in the *Performing a standard RHEL installation* guide.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **Content Hosts** and click the name of the Red Hat Enterprise Linux 8 host that you want to edit.
2. In the **System Purpose** area, click the **Edit** or **Remove** icon for the system purpose attributes that you want to edit, add, or remove.
3. Click **Save**.
4. Click the **Subscriptions** tab and select **Subscriptions**.
5. Click **Run Auto-Attach** to attach subscriptions to your host automatically.
6. Refresh the page to verify that the subscriptions list contains the correct subscriptions.

For CLI Users

1. Log in to the host and edit the required system purpose attributes. For example, set the usage type to **Production**, the role to **Red Hat Enterprise Linux Server**, and add the **addon** add on. For the list of values, see Configuring system purpose in the *Performing a standard RHEL installation* guide.

```
# syspurpose set-usage Production
# syspurpose set-role Red Hat Enterprise Linux Server
# syspurpose add-addons 'your_addon'
```

2. Verify the system purpose attributes for this host:

```
# syspurpose show
```

3. Automatically attach subscriptions to this host:

```
# subscription-manager attach --auto
```

4. Verify the system purpose status for this host:

```
# subscription-manager status
```

2.3. CHANGING A MODULE STREAM FOR A HOST

If you have a Red Hat Enterprise Linux 8 host, you can modify the module stream for the repositories you install. After you create the host, you can enable, disable, install, update, and remove module streams from your host in the Satellite web UI.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **Content Hosts** and click the name of the host that contains the modules you want to change.
2. Click the **Module Streams** tab.
3. From the **Available Module Streams** list, locate the module that you want to change. You can use the **Filter** field to refine the list entries. You can also use the **Filter Status** list to search for modules with a specific status.
4. From the **Actions** list, select the change that you want to make to the module.
5. In the **Job Invocation** window, ensure that the job information is accurate. Change any details that you require, and then click **Submit**.

2.4. CREATING A HOST GROUP

If you create a high volume of hosts, many of the hosts can have common settings and attributes. Adding these settings and attributes for every new host is time consuming. If you use host groups, you can apply common attributes to hosts that you create.

A host group functions as a template for common host settings, containing many of the same details that you provide to hosts. When you create a host with a host group, the host inherits the defined settings from the host group. You can then provide additional details to individualize the host.

Host Group Hierarchy

You can create a hierarchy of host groups. Aim to have one base level host group that represents all hosts in your organization and provide general settings, and then nested groups to provide specific settings. For example, you can have a base level host group that defines the operating system, and two nested host groups that inherit the base level host group:

- **Hostgroup: Base** (Red Hat Enterprise Linux 7.6)
  - **Hostgroup: Webserver** (applies the **httpd** Puppet class)
    - **Host: webserver1.example.com** (web server)
    - **Host: webserver2.example.com** (web server)
  - **Hostgroup: Storage** (applies the **nfs** Puppet class)
    - **Host: storage1.example.com** (storage server)
    - **Host: storage2.example.com** (storage server)
  - **Host: custom.example.com** (custom host)

In this example, all hosts use Red Hat Enterprise Linux 7.6 as their operating system because of their inheritance of the **Base** host group. The two web server hosts inherit the settings from the **Webserver**

host group, which includes the **httpd** Puppet class and the settings from the **Base** host group. The two storage servers inherit the settings from the **Storage** host group, which includes the **nfs** Puppet class and the settings from the **Base** host group. The custom host only inherits the settings from the **Base** host group.

Procedure

1. In the Satellite web UI, navigate to **Configure** > **Host Groups** and click **Create Host Group**.
2. If you have an existing host group that you want to inherit attributes from, you can select a host group from the **Parent** list. If you do not, leave this field blank.
3. Enter a **Name** for the new host group.
4. Enter any further information that you want future hosts to inherit.
5. Click the **Ansible Roles** tab, and from the **Ansible Roles** list, select one or more roles that you want to add to the host. Use the **arrow** icon to manage the roles that you add or remove.
6. Click the additional tabs and add any details that you want to attribute to the host group.

NOTE

Puppet fails to retrieve the Puppet CA certificate while registering a host with a host group associated with a Puppet environment created inside a **Production** environment.

To create a suitable Puppet environment to be associated with a host group, manually create a directory and change the owner:

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

7. Click **Submit** to save the host group.

For CLI Users

Create the host group with the **hammer hostgroup create** command. For example:

```
# hammer hostgroup create --name "Base" \
--lifecycle-environment "Production" --content-view "Base" \
--puppet-environment "production" --content-source-id 1 \
--puppet-ca-proxy-id 1 --puppet-proxy-id 1 --domain "example.com" \
--subnet `ACME's Internal Network` --architecture "x86_64" \
--operatingsystem "RedHat 7.2" --medium-id 9 \
--partition-table "Kickstart default" --root-pass "p@55w0rd!" \
--locations "New York" --organizations "ACME"
```

2.5. CREATING A HOST GROUP FOR EACH LIFECYCLE ENVIRONMENT

Use this procedure to create a host group for the Library lifecycle environment and add nested host groups for other lifecycle environments.

Procedure

To create a host group for each life cycle environment, run the following Bash script:

```
MAJOR="7"
ARCH="x86_64"
ORG="Your Organization"
LOCATIONS="Your Location"
PTABLE_NAME="Kickstart default"
DOMAIN="example.com"

hammer --output csv --no-headers lifecycle-environment list --organization "${ORG}" | cut -d ',' -f 2 |
while read LC_ENV; do
  [[ ${LC_ENV} == "Library" ]] && continue

  hammer hostgroup create --name "rhel-${MAJOR}server-${ARCH}-${LC_ENV}" \
    --architecture "${ARCH}" \
    --partition-table "${PTABLE_NAME}" \
    --domain "${DOMAIN}" \
    --organizations "${ORG}" \
    --query-organization "${ORG}" \
    --locations "${LOCATIONS}" \
    --lifecycle-environment "${LC_ENV}"
done
```

2.6. CHANGING THE GROUP OF A HOST

Use this procedure to change the group of a host.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
2. Select the check box of the host you want to change.
3. From the **Select Action** list, select **Change Group**. A new option window opens.
4. From the **Host Group** list, select the group that you want for your host.
5. Click **Submit**.

2.7. CHANGING THE ENVIRONMENT OF A HOST

Use this procedure to change the environment of a host.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
2. Select the check box of the host you want to change.
3. From the **Select Action** list, select **Change Environment**. A new option window opens.
4. From the **Environment** list, select the new environment for your host.
5. Click **Submit**.

2.8. CHANGING THE MANAGED STATUS OF A HOST

Hosts provisioned by Satellite are Managed by default. When a host is set to Managed, you can configure additional host parameters from Satellite Server. These additional parameters are listed on the **Operating System** tab. If you change any settings on the **Operating System** tab, they will not take effect until you set the host to build and reboot it.

If you need to obtain reports about configuration management on systems using an operating system not supported by Satellite, set the host to Unmanaged.

Use this procedure to switch a host between Managed and Unmanaged status.

Procedure

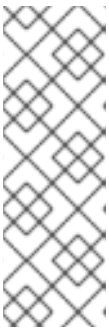
1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
2. Select the host.
3. Click **Edit**.
4. Click **Manage host** or **Unmanage host** to change the host's status.
5. Click **Submit**.

2.9. ASSIGNING A HOST TO A SPECIFIC ORGANIZATION

Use this procedure to assign a host to a specific organization. For general information about organizations and how to configure them, see Managing Organizations in the *Content Management Guide*.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
2. Select the check box of the host you want to change.
3. From the **Select Action** list, select **Assign Organization**. A new option window opens.
4. From the **Select Organization** list, select the organization that you want to assign your host to. Select the check box **Fix Organization on Mismatch**.



NOTE

A mismatch happens if there is a resource associated with a host, such as a domain or subnet, and at the same time not associated with the organization you want to assign the host to. The option **Fix Organization on Mismatch** will add such a resource to the organization, and is therefore the recommended choice. The option **Fail on Mismatch** will always result in an error message. For example, reassigning a host from one organization to another will fail, even if there is no actual mismatch in settings.

5. Click **Submit**.

2.10. ASSIGNING A HOST TO A SPECIFIC LOCATION

Use this procedure to assign a host to a specific location. For general information about locations and how to configure them, see Creating a Location in the *Provisioning Guide*.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
2. Select the check box of the host you want to change.
3. From the **Select Action** list, select **Assign Location**. A new option window opens.
4. Navigate to the **Select Location** list and choose the location that you want for your host. Select the check box **Fix Location on Mismatch**.

NOTE

A mismatch happens if there is a resource associated with a host, such as a domain or subnet, and at the same time not associated with the location you want to assign the host to. The option **Fix Location on Mismatch** will add such a resource to the location, and is therefore the recommended choice. The option **Fail on Mismatch** will always result in an error message. For example, reassigning a host from one location to another will fail, even if there is no actual mismatch in settings.

5. Click **Submit**.

2.11. REMOVING A HOST FROM RED HAT SATELLITE

Use this procedure to remove a host from Red Hat Satellite.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts** or **Hosts** > **Content Hosts**.
2. Select the hosts that you want to remove.
3. From the **Select Action** list, select **Delete Hosts**.
4. Click **Submit** to remove the host from Red Hat Satellite permanently.

WARNING

If a host record that is associated with a virtual machine is deleted, the virtual machine will be deleted as well. To avoid deleting the virtual machine in this situation, disassociate the virtual machine from Satellite without removing it from the hypervisor.

Disassociating A Virtual Machine from Satellite without Removing it from a Hypervisor

1. In the Satellite web UI, navigate to **Hosts** > **All Hosts** and select the check box to the left of the hosts to be disassociated.
2. From the **Select Action** list, select the **Disassociate Hosts** button.
3. Optionally, select the check box to keep the hosts for future action.
4. Click **Submit**.

CHAPTER 3. REGISTERING HOSTS

There are two main methods for registering a host to Satellite Server or Capsule Server:

- Download and install the consumer RPM (*server.example.com*/pub/katello-ca-consumer-latest.noarch.rpm) and then run Subscription Manager. This method is suited for freshly installed hosts.
- Download and run the bootstrap script (*server.example.com*/pub/bootstrap.py). This method is suited for both freshly installed hosts and hosts that have been previously registered, for example, to Satellite 5 or another Satellite 6.

You can also register Atomic Hosts to Satellite Server or Capsule Server.

Use one of the following procedures to register a host:

- Section 3.1, "Registering a Host to Red Hat Satellite"
- Section 3.2, "Registering an Atomic Host to Red Hat Satellite"
- Section 3.3, "Registering a Host to Red Hat Satellite Using The Bootstrap Script"

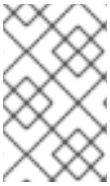
Use the following procedures to install and configure host tools:

- Section 3.4, "Installing the Katello Agent"
- Section 3.5, "Installing Tracer"
- Section 3.6, "Installing and Configuring the Puppet Agent"

Supported Host Operating Systems

Hosts must use one of the following Red Hat Enterprise Linux versions:

- 5.7 or later
- 6.1 or later*
- 7.0 or later
- 8.0 or later



NOTE

Red Hat Enterprise Linux versions 6.1, 6.2, and 6.3 require **subscription-manager** and related packages to be updated manually. For more information, see https://access.redhat.com/solutions/1256763.

Supported Architectures

All architectures of Red Hat Enterprise Linux are supported:

- i386
- x86_64
- s390x

- ppc_64

3.1. REGISTERING A HOST TO RED HAT SATELLITE

Use the following procedure to register a host to Red Hat Satellite 6.

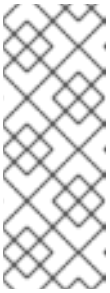
Prerequisites

- Satellite Server, any Capsule Servers, and all hosts must be synchronized with the same NTP server, and have a time synchronization tool enabled and running.
- The daemon **rhsmcertd** must be running on the hosts.
- An activation key must be available for the host. For more information, see Managing Activation Keys in the *Content Management Guide*.
- Subscription Manager must be version 1.10 or later. The package is available in the standard Red Hat Enterprise Linux repository.

Procedure

1. Red Hat Enterprise Linux hosts register to the Content Delivery Network by default. Update each host configuration so that they receive updates from the correct Satellite Server or Capsule Server:
   a. Take note of the fully qualified domain name (FQDN) of the Satellite Server or Capsule Server, for example *server.example.com*.
   b. Log in to the host as the **root** user and install the consumer RPM from the Satellite Server or Capsule Server to which the host is to be registered. The consumer RPM updates the content source location of the host and allows the host to download content from the content source specified in Red Hat Satellite.

```
# rpm -Uvh http://server.example.com/pub/katello-ca-consumer-latest.noarch.rpm
```



NOTE

The RPM package is not signed. If required, you can add the **--nosignature** option to install the package. The **katello-ca-consumer-*hostname*-1.0-1.noarch.rpm** package is an additional **katello-ca-consumer** RPM that contains the server's host name. The **katello-ca-consumer-latest.noarch.rpm** package always reflects the most recent version. Both serve the same purpose.

2. Clear any old host data related to Red Hat Subscription Manager (RHSM):

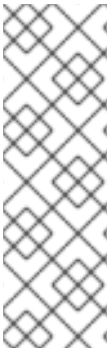
```
# subscription-manager clean
```

3. Register the host using RHSM:

```
# subscription-manager register --org=your_org_name \
--activationkey=your_activation_key
```

Example 3.1. Command Output after Registration:

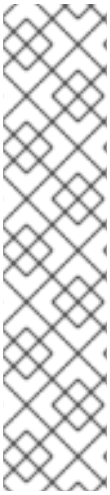
```
# subscription-manager register --org=MyOrg --activationkey=TestKey-1
The system has been registered with id: 62edc0f8-855b-4184-b1b8-72a9dc793b96
```



NOTE

You can use the **--environment** option to override the Content View and life cycle environment defined by the activation key. For example, to register a host to the Content View "MyView" in a "Development" life cycle environment:

```
# subscription-manager register --org=your_org_name \
--environment=Development/MyView \
--activationkey=your_activation_key
```



NOTE

For Red Hat Enterprise Linux 6.3 hosts, the release version defaults to Red Hat Enterprise Linux 6 Server and needs to be pointed to the 6.3 repository:

1. In the Satellite web UI, navigate to **Hosts** > **Content Hosts**.
2. Select the check box next to the host that needs to be changed.
3. From the **Select Action** list, select **Set Release Version**
4. From the **Release Version** list, select **6.3**.
5. Click **Done**.

3.2. REGISTERING AN ATOMIC HOST TO RED HAT SATELLITE

Use the following procedure to register an Atomic Host to Red Hat Satellite 6.

Procedure

1. Log in to the Atomic Host as the **root** user.
2. Retrieve **katello-rhsm-consumer** from Satellite Server:

   ```
   # wget http://satellite.example.com/pub/katello-rhsm-consumer
   ```

3. Change the mode of **katello-rhsm-consumer** to make it executable:

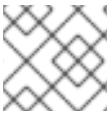
   ```
   # chmod +x katello-rhsm-consumer
   ```

4. Run **katello-rhsm-consumer**:

   ```
   # ./katello-rhsm-consumer
   ```

   Register with Red Hat Subscription Manager:

```
# subscription-manager register
```



NOTE

The Katello agent is not supported on Atomic Hosts.

3.3. REGISTERING A HOST TO RED HAT SATELLITE USING THE BOOTSTRAP SCRIPT

Use the bootstrap script to automate content registration and Puppet configuration. You can use the bootstrap script to register new hosts, or to migrate existing hosts to Red Hat Satellite 6 from Satellite 5, RHN, SAM, or RHSM.

The **katello-client-bootstrap** package is installed by default on Satellite Server's base operating system. The **bootstrap.py** script is installed in the **/var/www/html/pub/** directory to make it available to hosts at ***satellite6.example.com*/pub/bootstrap.py**. The script includes documentation in the **/usr/share/doc/katello-client-bootstrap-*version*/README.md** file.

To use the bootstrap script, you must install it on the host. As the script is only required once, and only for the **root** user, you can place it in **/root** or **/usr/local/sbin** and remove it after use. This procedure uses **/root**.

Prerequisites

- You have a Satellite user with the permissions required to run the bootstrap script. The examples in this procedure specify the **admin** user. If this is not acceptable to your security policy, create a new role with the minimum permissions required and add it to the user that will run the script. For more information, see Section 3.3.1, "Setting Permissions for the Bootstrap Script".
- You have an activation key for your hosts with the Satellite Tools repository enabled. For information on configuring activation keys, see Managing Activation Keys in the *Content Management Guide*.
- You have created a host group. For more information about creating host groups, see Section 2.4, "Creating a Host Group" .

Puppet Considerations

If a host group is associated with a Puppet environment created inside a **Production** environment, Puppet fails to retrieve the Puppet CA certificate while registering a host from that host group.

To create a suitable Puppet environment to be associated with a host group, follow these steps:

1. Manually create a directory and change the owner:

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

2. Navigate to **Configure** > **Environments** and click **Import environment from**. The button name includes the FQDN of the internal or external Capsule.
3. Choose the created directory and click **Update**.

Procedure

Procedure

1. Log in to the host as the **root** user.
2. Download the script:

```
# curl -O http://satellite6.example.com/pub/bootstrap.py
```

3. Make the script executable:

```
# chmod +x bootstrap.py
```

4. Confirm that the script is executable by viewing the help text:
   - On Red Hat Enterprise Linux 8:

```
# /usr/libexec/platform-python bootstrap.py -h
```

   - On other Red Hat Enterprise Linux versions:

```
# ./bootstrap.py -h
```

5. Enter the bootstrap command with values suitable for your environment.
   For the **--server** option, specify the FQDN of Satellite Server or a Capsule Server. For the **--location**, **--organization**, and **--hostgroup** options, use quoted names, not labels, as arguments to the options. See Section 3.3.2, "Advanced Bootstrap Script Configuration" for advanced use cases.
   - On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key
```

   - On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# ./bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key
```

6. Enter the password of the Satellite user you specified with the **--login** option.
   The script sends notices of progress to **stdout**.
7. When prompted by the script, approve the host's Puppet certificate. In the Satellite web UI, navigate to **Infrastructure** > **Capsules** and find the Satellite or Capsule Server you specified with the **--server** option.
8. From the list in the **Actions** column, select **Certificates**.

9. In the **Actions** column, click **Sign** to approve the host's Puppet certificate.
10. Return to the host to see the remainder of the bootstrap process completing.
11. In the Satellite web UI, navigate to **Hosts** > **All hosts** and ensure that the host is connected to the correct host group.
12. Optional: After the host registration is complete, remove the script:

```
# rm bootstrap.py
```

3.3.1. Setting Permissions for the Bootstrap Script

Use this procedure to configure a Satellite user with the permissions required to run the bootstrap script.

Procedure

1. In the Satellite web UI, navigate to **Administer** > **Users**.
2. Select an existing user by clicking the required **Username**. A new pane opens with tabs to modify information about the selected user. Alternatively, create a new user specifically for the purpose of running this script.
3. Click the **Roles** tab.
4. Select **Edit hosts** and **Viewer** from the **Roles** list.

IMPORTANT

The **Edit hosts** role allows the user to edit and delete hosts as well as being able to add hosts. If this is not acceptable to your security policy, create a new role with the following permissions and assign it to the user:

- **view_organizations**
- **view_locations**
- **view_domains**
- **view_hostgroups**
- **view_hosts**
- **view_architectures**
- **view_ptables**
- **view_operatingsystems**
- **create_hosts**

5. Click **Submit**.

For CLI Users

1. Create a role with the minimum permissions required by the bootstrap script. This example creates a role with the name *Bootstrap*:

```
# ROLE='Bootstrap'
hammer role create --name "$ROLE"
hammer filter create --role "$ROLE" --permissions view_organizations
hammer filter create --role "$ROLE" --permissions view_locations
hammer filter create --role "$ROLE" --permissions view_domains
hammer filter create --role "$ROLE" --permissions view_hostgroups
hammer filter create --role "$ROLE" --permissions view_hosts
hammer filter create --role "$ROLE" --permissions view_architectures
hammer filter create --role "$ROLE" --permissions view_ptables
hammer filter create --role "$ROLE" --permissions view_operatingsystems
hammer filter create --role "$ROLE" --permissions create_hosts
```

2. Assign the new role to an existing user:

```
# hammer user add-role --id user_id --role Bootstrap
```

Alternatively, you can create a new user and assign this new role to them. For more information on creating users with Hammer, see Creating Users in the *Hammer CLI Guide*.

3.3.2. Advanced Bootstrap Script Configuration

This section has more examples for using the bootstrap script to register or migrate a host.

WARNING

These examples specify the **admin** Satellite user. If this is not acceptable to your security policy, create a new role with the minimum permissions required by the bootstrap script. For more information, see Section 3.3.1, "Setting Permissions for the Bootstrap Script".

Migrating a host from one Satellite 6 to another Satellite 6

Use the script with **--force** to remove the **katello-ca-consumer-*** packages from the old Satellite and install the **katello-ca-consumer-*** packages on the new Satellite.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--force
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--force
```

Migrating a host from Red Hat Network (RHN) or Satellite 5 to Satellite 6

The bootstrap script detects the presence of **/etc/syconfig/rhn/systemid** and a valid connection to RHN as an indicator that the system is registered to a legacy platform. The script then calls **rhn-classic-migrate-to-rhsm** to migrate the system from RHN. By default, the script does not delete the system's legacy profile due to auditing reasons. To remove the legacy profile, use **--legacy-purge**, and use **--legacy-login** to supply a user account that has appropriate permissions to remove a profile. Enter the user account password when prompted.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--legacy-purge \
--legacy-login rhn-user
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--legacy-purge \
--legacy-login rhn-user
```

Registering a host to Satellite 6, omitting Puppet setup

By default, the bootstrap script configures the host for content management and configuration management. If you have an existing configuration management system and do not want to install Puppet on the host, use **--skip-puppet**.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
```

```
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--skip-puppet
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--skip-puppet
```

Registering a host to Satellite 6 for content management only

To register a system as a content host, and omit the provisioning and configuration management functions, use **--skip-foreman**.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--server satellite6.example.com \
--organization="Example Organization" \
--activationkey=activation_key \
--skip-foreman
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --server satellite6.example.com \
--organization="Example Organization" \
--activationkey=activation_key \
--skip-foreman
```

Changing the method the bootstrap script uses to download the consumer RPM

By default, the bootstrap script uses HTTP to download the consumer RPM (*server.example.com*/pub/katello-ca-consumer-latest.noarch.rpm). In some environments, you might want to allow HTTPS only between the host and Satellite. Use **--download-method** to change the download method from HTTP to HTTPS.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--download-method https
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
```

```
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--download-method https
```

Providing the host's IP address to Satellite

On hosts with multiple interfaces or multiple IP addresses on one interface, you might need to override the auto-detection of the IP address and provide a specific IP address to Satellite. Use **--ip**.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--ip 192.x.x.x
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--ip 192.x.x.x
```

Enabling remote execution on the host

Use **--rex** and **--rex-user** to enable remote execution and add the required SSH keys for the specified user.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--rex \
--rex-user root
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
```

```
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--rex \
--rex-user root
```

Creating a domain for a host during registration

To create a host record, the DNS domain of a host needs to exist in Satellite prior to running the script. If the domain does not exist, add it using **--add-domain**.

- On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py \
--login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--add-domain
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--add-domain
```

Providing an alternative FQDN for the host

If the host's host name is not an FQDN, or is not RFC-compliant (containing a character such as an underscore), the script will fail at the host name validation stage. If you cannot update the host to use an FQDN that is accepted by Satellite, you can use the bootstrap script to specify an alternative FQDN.

1. Set **create_new_host_when_facts_are_uploaded** and **create_new_host_when_report_is_uploaded** to false using Hammer:

```
# hammer settings set \
--name  create_new_host_when_facts_are_uploaded \
--value false
# hammer settings set \
--name  create_new_host_when_report_is_uploaded \
--value false
```

2. Use **--fqdn** to specify the FQDN that will be reported to Satellite:

   - On Red Hat Enterprise Linux 8, enter the following command:

```
# /usr/libexec/platform-python bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
```

```
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--fqdn node100.example.com
```

- On Red Hat Enterprise Linux 5, 6, or 7, enter the following command:

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key \
--fqdn node100.example.com
```

3.4. INSTALLING THE KATELLO AGENT

To remotely update Satellite clients, you must install the Katello agent.

The **katello-agent** package depends on the **gofer** package that provides the **goferd** service. This service must be enabled so that Satellite Server or Capsule Server can provide information about errata that are applicable for content hosts.

Prerequisites

Before installing the Katello agent, ensure the following conditions are met:

- You have enabled the Satellite Tools repository on Satellite Server. For more information, see Enabling the Satellite Tools Repository in *Installing Satellite Server from a Connected Network* .
- You have synchronized the Satellite Tools repository on Satellite Server. For more information, see Synchronizing the Satellite Tools Repository in *Installing Satellite Server from a Connected Network*.
- You have enabled the Satellite Tools repository on the client. For example, to ensure that the repository is enabled on the Red Hat Enterprise Linux 7 client, enter the following command on the client:

```
# subscription-manager repos --enable rhel-7-server-satellite-tools-6.6-rpms
```

Procedure

To install the Katello agent, complete the following steps:

1. Install the **katello-agent** package:

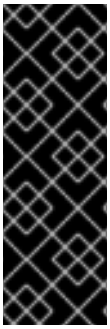
```
# yum install katello-agent
```

2. Start the **goferd** service :

```
# systemctl start goferd
```

3.5. INSTALLING TRACER

Use this procedure to install Tracer on Red Hat Satellite 6.6, and access Traces. Tracer displays a list of services and applications that are outdated and need to be restarted. Traces is the output generated by Tracer in the Satellite web UI.



IMPORTANT

The integration of Tracer with Satellite Server is a Technology Preview feature. Technology Preview features are not fully supported under Red Hat Subscription Service Level Agreements (SLAs), may not be functionally complete, and are not intended for production use. These features provide early access to upcoming product innovations, enabling customers to test functionality and provide feedback during the development process. For more information see Red Hat Technology Preview Features Support Scope.

Prerequisites

- The host must be registered to Red Hat Satellite.
- The Red Hat Satellite Tools 6.6 repository must be enabled and synchronized on Satellite Server, and enabled on the host.

Procedure

1. On the content host, install the **katello-host-tools-tracer** RPM package:

```
# yum install katello-host-tools-tracer
```

2. Enter the following command:

```
# katello-tracer-upload
```

3. In the Satellite web UI, navigate to **Hosts** > **All hosts**, then click the required host name.
4. In the **Properties** tab, examine the **Properties** table and find the Traces item. If you cannot find a Traces item in the **Properties** table, then Tracer is not installed.
5. Navigate to **Hosts** > **Content Hosts**, then click the required host name.
6. Click the **Traces** tab to view Traces.

3.6. INSTALLING AND CONFIGURING THE PUPPET AGENT

Use this procedure to install and configure the Puppet agent on a host. For more information about Puppet, see the *Puppet Guide*.

Prerequisites

- The host must be registered to Red Hat Satellite.
- The host must have a Puppet environment assigned to it.
- The Red Hat Satellite Tools 6.6 repository must be enabled and synchronized on Satellite Server, and enabled on the host.

Procedure

1. Log in to the host as the **root** user.
2. Install the Puppet agent package:

```
# yum install puppet-agent
```

3. Configure the Puppet agent to start on boot:
   - On Red Hat Enterprise Linux 6:

```
# chkconfig puppet on
```

   - On Red Hat Enterprise Linux 7:

```
# systemctl enable puppet
```

4. Append the following server and environment settings to the **/etc/puppetlabs/puppet/puppet.conf** file. Set the **environment** parameter to the name of the Puppet environment to which the host belongs:

```
environment = My_Example_Org_Library
server = satellite.example.com
ca_server = satellite.example.com
```

5. Run the Puppet agent on the host:

```
# puppet agent -t
```

6. In the Satellite web UI, navigate to **Infrastructure** > **Capsules**.
7. From the list in the **Actions** column for the required Capsule Server, select **Certificates**.
8. Click **Sign** to the right of the required host to sign the SSL certificate for the Puppet client.
9. Enter the **puppet agent** command again:

```
# puppet agent -t
```

CHAPTER 4. ADDING NETWORK INTERFACES

Red Hat Satellite supports specifying multiple network interfaces for a single host. You can configure these interfaces when creating a new host as described in Section 2.1, “Creating a Host in Red Hat Satellite” or when editing an existing host.

There are several types of network interfaces that you can attach to a host. When adding a new interface, select one of:

- **Interface**: Allows you to specify an additional physical or virtual interface. There are two types of virtual interfaces you can create. Use **VLAN** when the host needs to communicate with several (virtual) networks using a single interface, while these networks are not accessible to each other. Use **alias** to add an additional IP address to an existing interface.
  For more information about adding a physical interface, see Section 4.1, “Adding a Physical Interface”.

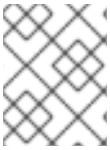
  For more information about adding a virtual interface, see Section 4.2, “Adding a Virtual Interface”.

- **Bond**: Creates a bonded interface. NIC bonding is a way to bind multiple network interfaces together into a single interface that appears as a single device and has a single MAC address. This enables two or more network interfaces to act as one, increasing the bandwidth and providing redundancy. See Section 4.3, “Adding a Bonded Interface” for details.

- **BMC**: Baseboard Management Controller (BMC) allows you to remotely monitor and manage the physical state of machines. For more information about BMC, see Enabling Power Management on Managed Hosts in *Installing Satellite Server from a Connected Network* . For more information about configuring BMC interfaces, see Section 4.4, “Adding a Baseboard Management Controller (BMC) Interface”.

NOTE

Additional interfaces have the **Managed** flag enabled by default, which means the new interface is configured automatically during provisioning by the DNS and DHCP Capsule Servers associated with the selected subnet. This requires a subnet with correctly configured DNS and DHCP Capsule Servers. If you use a Kickstart method for host provisioning, configuration files are automatically created for managed interfaces in the post-installation phase at **/etc/sysconfig/network-scripts/ifcfg-*interface_id***.



NOTE

Virtual and bonded interfaces currently require a MAC address of a physical device. Therefore, the configuration of these interfaces works only on bare-metal hosts.

4.1. ADDING A PHYSICAL INTERFACE

Use this procedure to add an additional physical interface to a host.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
2. Click **Edit** next to the host you want to edit.
3. On the **Interfaces** tab, click **Add Interface**.

4. Keep the **Interface** option selected in the **Type** list.
5. Specify a **MAC address**. This setting is required.
6. Specify the **Device Identifier**, for example **eth0**. The identifier is used to specify this physical interface when creating bonded interfaces, VLANs, and aliases.
7. Specify the **DNS name** associated with the host's IP address. Satellite saves this name in the Capsule Server associated with the selected domain (the "DNS A" field) and the Capsule Server associated with the selected subnet (the "DNS PTR" field). A single host can therefore have several DNS entries.
8. Select a domain from the **Domain** list. To create and manage domains, navigate to **Infrastructure** > **Domains**.
9. Select a subnet from the **Subnet** list. To create and manage subnets, navigate to **Infrastructure** > **Subnets**.
10. Specify the **IP address**. Managed interfaces with an assigned DHCP Capsule Server require this setting for creating a DHCP lease. DHCP-enabled managed interfaces are automatically provided with a suggested IP address.
11. Select whether the interface is **Managed**. If the interface is managed, configuration is pulled from the associated Capsule Server during provisioning, and DNS and DHCP entries are created. If using kickstart provisioning, a configuration file is automatically created for the interface.
12. Select whether this is the **Primary** interface for the host. The DNS name from the primary interface is used as the host portion of the FQDN.
13. Select whether this is the **Provision** interface for the host. TFTP boot takes place using the provisioning interface. For image-based provisioning, the script to complete the provisioning is executed through the provisioning interface.
14. Select whether to use the interface for **Remote execution**.
15. Leave the **Virtual NIC** check box clear.
16. Click **OK** to save the interface configuration.
17. Click **Submit** to apply the changes to the host.

4.2. ADDING A VIRTUAL INTERFACE

Use this procedure to configure a virtual interface for a host. This can be either a VLAN or an alias interface.

An alias interface is an additional IP address attached to an existing interface. An alias interface automatically inherits a MAC address from the interface it is attached to; therefore, you can create an alias without specifying a MAC address. The interface must be specified in a subnet with boot mode set to **static**.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.

2. Click **Edit** next to the host you want to edit.
3. On the **Interfaces** tab, click **Add Interface**.
4. Keep the **Interface** option selected in the **Type** list.
5. Specify the general interface settings. The applicable configuration options are the same as for the physical interfaces described in Section 4.1, "Adding a Physical Interface" .
   Specify a **MAC address** for managed virtual interfaces so that the configuration files for provisioning are generated correctly. However, a **MAC address** is not required for virtual interfaces that are not managed.

   If creating a VLAN, specify ID in the form of **eth1.10** in the **Device Identifier** field. If creating an alias, use ID in the form of **eth1:10**.
6. Select the **Virtual NIC** check box. Additional configuration options specific to virtual interfaces are appended to the form:
   - **Tag**: Optionally set a VLAN tag to trunk a network segment from the physical network through to the virtual interface. If you do not specify a tag, managed interfaces inherit the VLAN tag of the associated subnet. User-specified entries from this field are not applied to alias interfaces.
   - **Attached to**: Specify the identifier of the physical interface to which the virtual interface belongs, for example **eth1**. This setting is required.
7. Click **OK** to save the interface configuration.
8. Click **Submit** to apply the changes to the host.

4.3. ADDING A BONDED INTERFACE

Use this procedure to configure a bonded interface for a host.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
2. Click **Edit** next to the host you want to edit.
3. On the **Interfaces** tab, click **Add Interface**.
4. Select **Bond** from the **Type** list. Additional type-specific configuration options are appended to the form.
5. Specify the general interface settings. The applicable configuration options are the same as for the physical interfaces described in Section 4.1, "Adding a Physical Interface" .
   Bonded interfaces use IDs in the form of **bond0** in the **Device Identifier** field.

   A single **MAC address** is sufficient.
6. Specify the configuration options specific to bonded interfaces:
   - **Mode**: Select the bonding mode that defines a policy for fault tolerance and load balancing. See Table 4.1, "Bonding Modes Available in Red Hat Satellite" for a brief description of each bonding mode.

- **Attached devices**: Specify a comma-separated list of identifiers of attached devices. These can be physical interfaces or VLANs.
- **Bond options**: Specify a space-separated list of configuration options, for example **miimon=100**. See the Red Hat Enterprise Linux 7 Networking Guide for details of the configuration options you can specify for the bonded interface.

7. Click **OK** to save the interface configuration.
8. Click **Submit** to apply the changes to the host.

For CLI Users

To create a host with a bonded interface, enter the following command:

```
# hammer host create --name bonded_interface \
--hostgroup-id 1 \
--ip=192.168.100.123 \
--mac=52:54:00:14:92:2a \
--subnet-id=1 \
--managed true \
  --interface="identifier=eth1, \
              mac=52:54:00:62:43:06, \
              managed=true, \
              type=Nic::Managed, \
              domain_id=1, \
              subnet_id=1" \
  --interface="identifier=eth2, \
              mac=52:54:00:d3:87:8f, \
              managed=true, \
              type=Nic::Managed, \
              domain_id=1, \
              subnet_id=1" \
  --interface="identifier=bond0, \
              ip=172.25.18.123, \
              type=Nic::Bond, \
              mode=active-backup, \
              attached_devices=[eth1,eth2], \
              managed=true, \
              domain_id=1, \
              subnet_id=1" \
--organization "Your_Organization" \
--location "Your_Location" \
--ask-root-password yes
```

Table 4.1. Bonding Modes Available in Red Hat Satellite

Bonding Mode	Description
balance-rr	Transmissions are received and sent sequentially on each bonded interface.

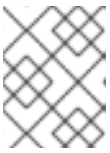
Bonding Mode	Description
active-backup	Transmissions are received and sent through the first available bonded interface. Another bonded interface is only used if the active bonded interface fails.
balance-xor	Transmissions are based on the selected hash policy. In this mode, traffic destined for specific peers is always sent over the same interface.
broadcast	All transmissions are sent on all bonded interfaces.
802.a3	Creates aggregation groups that share the same settings. Transmits and receives on all interfaces in the active group.
balance-tlb	The outgoing traffic is distributed according to the current load on each bonded interface.
balance-alb	Receive load balancing is achieved through Address Resolution Protocol (ARP) negotiation.

4.4. ADDING A BASEBOARD MANAGEMENT CONTROLLER (BMC) INTERFACE

Use this procedure to configure a baseboard management controller (BMC) interface for a host that supports this feature.

Prerequisites

- The **ipmitool** package is installed.
- You know the MAC address, IP address, and other details of the BMC interface on the host, and the appropriate credentials for that interface.



NOTE

You only need the MAC address for the BMC interface if the BMC interface is managed, so that it can create a DHCP reservation.

Procedure

1. Enable BMC on the Capsule server if it is not already enabled:
   a. Configure BMC power management on the Capsule Server by running the **satellite-installer** script with the following options:

```
# satellite-installer --foreman-proxy-bmc=true \
--foreman-proxy-bmc-default-provider=ipmitool
```

b. In the Satellite web UI, navigate to **Infrastructure** > **Capsules**.

c. From the list in the **Actions** column, click **Refresh**. The list in the **Features** column should now include BMC.

2. In the Satellite web UI, navigate to **Hosts** > **All hosts**.
3. Click **Edit** next to the host you want to edit.
4. On the **Interfaces** tab, click **Add Interface**.
5. Select **BMC** from the **Type** list. Type-specific configuration options are appended to the form.
6. Specify the general interface settings. The applicable configuration options are the same as for the physical interfaces described in Section 4.1, “Adding a Physical Interface” .
7. Specify the configuration options specific to BMC interfaces:
   - **Username** and **Password**: Specify any authentication credentials required by BMC.
   - **Provider**: Specify the BMC provider.
8. Click **OK** to save the interface configuration.
9. Click **Submit** to apply the changes to the host.

CHAPTER 5. MONITORING HOSTS USING RED HAT INSIGHTS

In this chapter, you can find information about creating host monitoring reports and monitoring your hosts using Red Hat Insights and creating an Insights plan.

5.1. USING RED HAT INSIGHTS WITH HOSTS IN SATELLITE

You can use Red Hat Insights to diagnose systems and downtime related to security exploits, performance degradation and stability failures. You can use the dashboard to quickly identify key risks to stability, security, and performance. You can sort by category, view details of the impact and resolution, and then determine what systems are affected.

To use Red Hat Insights to monitor hosts that you manage with Satellite, you must first install Red Hat Insights on your hosts and register your hosts with Red Hat Insights.

To install and register your host using Puppet, or manually, see Red Hat Insights Getting Started.

Deploying Red Hat Insights using the Ansible Role

You can automate the installation and registration of hosts with Red Hat Insights using the **RedHatInsights.insights-client** Ansible role. For more information about adding this role to your Satellite, see Managing Ansible Roles.

1. Add the **RedHatInsights.insights-client** role to the hosts.
   For new hosts, see Section 2.1, “Creating a Host in Red Hat Satellite” .

   For existing hosts, see Chapter 8, *Using Ansible Roles*.

2. To run the **RedHatInsights.insights-client** role on your host, navigate to **Hosts** > **All Hosts** and click the name of the host that you want to use.

3. Click the **Run Ansible roles** button.

When the role completes, you can view and work with the host that you add on the **Insights** > **Overview** page of the Satellite web UI.

Additional Information

- To apply any system updates to the Red Hat Insights plug-in, enter **httpd restart** after updating.
- To view the logs for Red Hat Insights and all plug-ins, go to **/var/log/foreman/production.log**.
- If you have problems connecting to Red Hat Insights, ensure that your certificates are up-to-date. Refresh your subscription manifest to update your certificates.
- You can change the default schedule for running **insights-client** by configuring **insights-client.timer** on a host. For more information, see Changing the insights-client schedule in the *Client Configuration Guide for Red Hat Insights* .

5.2. CREATING AN INSIGHTS PLAN FOR HOSTS

With Satellite 6, you can create a Red Hat Insights remediation plan and run the plan on Satellite hosts.

Procedure

To create the plan, complete the following steps:

1. In the Satellite web UI, navigate to **Insights** > **Inventory**, and select the hosts that you want to include in an Insights plan.
2. From the **Actions** list, select **Create a new Plan/Playbook**
3. From the Plan/Playbook Builder window, select **Create new plan** and enter a name for the plan.
4. Select whether you want the rule to apply to a specific system, group, or all systems.
5. Select one or more rules that you want to add to the plan. Use the **Filter** field to search for specific keywords.
6. Click **Save**.
7. In the Planner window, select **Run Playbook**.

In the Jobs window, you can view the progress of your plan as the playbook runs.

You can view the Insights plan by navigating to **Insights** > **Planner**.

CHAPTER 6. USING REPORT TEMPLATES TO MONITOR HOSTS

You can use report templates to query Satellite data to obtain information about, for example, host status, registered hosts, applicable errata, applied errata, subscription details, and user activity. You can use the report templates that ship with Satellite or write your own custom report templates to suit your requirements. The reporting engine uses the embedded Ruby (ERB) syntax. For more information about writing templates and ERB syntax, see Appendix A, *Template Writing Reference*.

You can create a template, or clone a template and edit the clone. For help with the template syntax, click a template and click the **Help** tab.

6.1. GENERATING HOST MONITORING REPORTS

To view the report templates, in the Satellite web UI, navigate to **Monitor** > **Report Templates**.

To schedule reports, you can configure a cron job or use the Satellite web UI.

Procedure

1. In the Satellite web UI, navigate to **Monitor** > **Report Templates**.
2. To the right of the report template that you want to use, click **Generate**.
3. Optional: To schedule a report, to the right of the **Generate at** field, click the icon to select the date and time you want to generate the report at.
4. Optional: To send a report to an e-mail address, select the **Send report via e-mail** check box, and in the **Deliver to e-mail addresses** field, enter the required e-mail address.
5. Optional: Apply search query filters. To view all available results, do not populate the filter field with any values.
6. Click **Submit**. A CSV file that contains the report is downloaded. If you have selected the **Send report via e-mail** check box, the host monitoring report is sent to your e-mail address.

For CLI Users

To generate a report, complete the following steps:

1. List all available report templates:

```
# hammer report-template list
```

2. Generate a report:

```
# hammer report-template generate --id template ID
```

This command waits until the report fully generates before completing. If you want to generate the report as a background task, you can use the **hammer report-template schedule** command.

6.2. CREATING A REPORT TEMPLATE

In Satellite, you can create a report template and customize the template to suit your requirements. You can import existing report templates and further customize them with snippets and template macros.

Report templates use Embedded Ruby (ERB) syntax. To view information about working with ERB syntax and macros, in the Satellite web UI, navigate to **Monitor** > **Report Templates**, and click **Create Template**, and then click the **Help** tab.

When you create a report template in Satellite, safe mode is enabled by default. For more information about safe mode, see Section 6.7, "Report Template Safe Mode" .

For more information about writing templates, see the Appendix A, *Template Writing Reference*.

For more information about macros you can use in report templates, see Section A.4, "Templates Macros".

Procedure

1. In the Satellite web UI, navigate to **Monitor** > **Report Templates**, and click **Create Template**.
2. In the **Name** field, enter a unique name for your report template.
3. If you want the template to be available to all locations and organizations, select **Default**.
4. Create the template directly in the template editor or import a template from a text file by clicking **Import**. For more information about importing templates, see Section 6.5, "Importing Report Templates".
5. Optional: In the **Audit Comment** field, you can add any useful information about this template.
6. Click the **Input** tab, and in the **Name** field, enter a name for the input that you can reference in the template in the following format: **input('name')**. Note that you must save the template before you can reference this input value in the template body.
7. Select whether the input value is mandatory. If the input value is mandatory, select the **Required** check box.
8. From the **Value Type** list, select the type of input value that the user must input.
9. Optional: If you want to use facts for template input, select the **Advanced** check box.
10. Optional: In the **Options** field, define the options that the user can select from. If this field remains undefined, the users receive a free-text field in which they can enter the value they want.
11. Optional: In the **Default** field, enter a value, for example, a host name, that you want to set as the default template input.
12. Optional: In the **Description** field, you can enter information that you want to display as inline help about the input when you generate the report.
13. Optional: Click the **Type** tab, and select whether this template is a snippet to be included in other templates,
14. Click the **Location** tab and add the locations where you want to use the template.
15. Click the **Organizations** tab and add the organizations where you want to use the template.

16. Click **Submit** to save your changes.

6.3. EXPORTING REPORT TEMPLATES

You can export report templates that you create in Satellite.

Procedure

1. In the Satellite web UI, navigate to **Monitor** > **Report Templates**.
2. Locate the template that you want to export, and from the list in the **Actions** column, select **Export**.
3. Repeat this action for every report template that you want to download.

An **.erb** file that contains the template downloads.

For CLI Users

1. To view the report templates available for export, enter the following command:

```
# hammer report-template list
```

Note the template ID of the template that you want to export in the output of this command.

2. To export a report template, enter the following command:

```
# hammer report-template dump --id template_ID > example_export.erb
```

6.4. EXPORTING REPORT TEMPLATES USING THE SATELLITE API

You can use the Satellite **report_templates** API to export report templates from Satellite. For more information about using the Satellite API, see the API Guide.

Procedure

1. Use the following request to retrieve a list of available report templates:

Example request:

```
$ curl --user admin:redhat \
--request GET \
--config https://satellite.example.com/api/report_templates \
| json_reformat
```

In this example, the **json_reformat** tool is used to format the JSON output.

Example response:

```
{
  "total": 6,
  "subtotal": 6,
  "page": 1,
```

```
    "per_page": 20,
    "search": null,
    "sort": {
        "by": null,
        "order": null
    },
    "results": [
        {
            "created_at": "2019-11-20 17:49:52 UTC",
            "updated_at": "2019-11-20 17:49:52 UTC",
            "name": "Applicable errata",
            "id": 112
        },
        {
            "created_at": "2019-11-20 17:49:52 UTC",
            "updated_at": "2019-11-20 17:49:52 UTC",
            "name": "Applied Errata",
            "id": 113
        },
        {
            "created_at": "2019-11-30 16:15:24 UTC",
            "updated_at": "2019-11-30 16:15:24 UTC",
            "name": "Hosts - complete list",
            "id": 158
        },
        {
            "created_at": "2019-11-20 17:49:52 UTC",
            "updated_at": "2019-11-20 17:49:52 UTC",
            "name": "Host statuses",
            "id": 114
        },
        {
            "created_at": "2019-11-20 17:49:52 UTC",
            "updated_at": "2019-11-20 17:49:52 UTC",
            "name": "Registered hosts",
            "id": 115
        },
        {
            "created_at": "2019-11-20 17:49:52 UTC",
            "updated_at": "2019-11-20 17:49:52 UTC",
            "name": "Subscriptions",
            "id": 116
        }
    ]
}
```


2. Note the **id** of the template that you want to export, and use the following request to export the template:

Example request:

```
% curl --output /tmp/_Example_Export_Template.erb_ \
--user admin:password --request GET --config \
https://satellite.example.com/api/report_templates/158/export
```

Note that **158** is an example ID of the template to export.

In this example, the exported template is redirected to **host_complete_list.erb**.

6.5. IMPORTING REPORT TEMPLATES

You can import a report template into the body of a new template that you want to create. Note that using the web UI, you can only import templates individually. For bulk actions, use the Satellite API. For more information, see Section 6.6, "Importing Report Templates Using the Satellite API" .

Prerequisite

- You must have exported templates from Satellite to import them to use in new templates. For more information see Section 6.3, "Exporting Report Templates".

Procedure

1. In the Satellite web UI, navigate to **Monitor** > **Report Templates**, and in the upper right of the Report Templates window, click **Create Template**.
2. In the **Template** area, click **Import**, and select the **.erb** file that you want to import.
3. Edit the template to suit your requirements, and click **Submit**.

For more information about customizing your new template, see Appendix A, *Template Writing Reference*.

6.6. IMPORTING REPORT TEMPLATES USING THE SATELLITE API

You can use the Satellite API to import report templates into Satellite. Importing report templates using the Satellite API automatically parses the report template metadata and assigns organizations and locations. For more information about using the Satellite API, see the API Guide.

Prerequisite

- Create a template using **.erb** syntax or export a template from another Satellite.
  For more information about writing templates, see Appendix A, *Template Writing Reference*.

  For more information about exporting templates from Satellite, see Section 6.4, "Exporting Report Templates Using the Satellite API".

Procedure

1. Use the following example to format the template that you want to import to a **.json** file:

```
# cat Example_Template.json
{
   "name": "Example Template Name",
   "template": "
   Enter ERB Code Here
"
}
```

Example JSON File with ERB Template:

```
{
    "name": "Hosts - complete list",
    "template": "
<%#
name: Hosts - complete list
snippet: false
template_inputs:
- name: host
  required: false
  input_type: user
  advanced: false
  value_type: plain
  resource_type: Katello::ActivationKey
model: ReportTemplate
-%>
<% load_hosts(search: input('host')).each_record do |host| -%>
<%
      report_row(
         'Server FQND': host.name
      )
-%>
<%  end -%>
<%= report_render %>
"
}
```

2. Use the following request to import the template:

```
$ curl --user admin:redhat \
--data @Example_Template.json --header "Content-Type:application/json" \
--request POST --config https://satellite.example.com/api/report_templates/import
```

3. Use the following request to retrieve a list of report templates and validate that you can view the template in Satellite:

```
$ curl --user admin:redhat \
 --request GET --config https://satellite.example.com/api/report_templates | json_reformat
```

6.7. REPORT TEMPLATE SAFE MODE

When you create report templates in Satellite, safe mode is enabled by default. Safe mode limits the macros and variables that you can use in the report template. Safe mode prevents rendering problems and enforces best practices in report templates. The list of supported macros and variables is available in the Satellite web UI.

To view the macros and variables that are available, in the Satellite web UI, navigate to **Monitor** > **Report Templates** and click **Create Template**. In the Create Template window, click the **Help** tab and expand **Safe mode methods**.

While safe mode is enabled, if you try to use a macro or variable that is not listed in **Safe mode methods**, the template editor displays an error message.

To view the status of safe mode in Satellite, in the Satellite web UI, navigate to **Administer** > **Settings** and click the **Provisioning** tab. Locate the **Safemode rendering** row to check the value.

CHAPTER 7. CONFIGURING HOST COLLECTIONS

A host collection is a group of multiple content hosts. This feature enables you to perform the same action on multiple hosts at once. These actions can include the installation, removal, and update of packages and errata, change of assigned life cycle environment, and change of Content View. You can create host collections to suit your requirements, and those of your company. For example, group hosts in host collections by function, department, or business unit.

7.1. CREATING A HOST COLLECTION

The following procedure shows how to create host collections.

To Create a Host Collection:

1. Click **Hosts** > **Host Collections**.
2. Click **New Host Collection**.
3. Add the Name of the host collection.
4. Clear **Unlimited Content Hosts**, and enter the desired maximum number of hosts in the **Limit** field.
5. Add the Description of the host collection.
6. Click **Save**.

For CLI Users

To create a host collection, enter the following command:

```
# hammer host-collection create \
--organization "Your_Organization" \
--name hc_name
```

7.2. CLONING A HOST COLLECTION

The following procedure shows how to clone a host collection.

To Clone a Host Collection:

1. Click **Hosts** > **Host Collections**.
2. On the left hand panel, click the host collection you want to clone.
3. Click **Copy Collection**.
4. Specify a name for the cloned collection.
5. Click **Create**.

7.3. REMOVING A HOST COLLECTION

The following procedure shows how to remove a host collection.

To Remove a Host Collection:

1. Click **Hosts** > **Host Collections**.
2. Choose the host collection to be removed.
3. Click **Remove**. An alert box appears:

   > Are you sure you want to remove host collection *Host Collection Name*?

4. Click **Remove**.

7.4. ADDING A HOST TO A HOST COLLECTION

The following procedure shows how to add hosts to host collections.

Prerequisites

A host must be registered to Red Hat Satellite in order to add it to a Host Collection. For more information about registering hosts, Chapter 3, *Registering Hosts*.

To Add Hosts to a Host Collection:

1. Click **Hosts** > **Host Collections**.
2. Click the host collection where the host should be added.
3. On the **Hosts** tab, select the **Add** subtab.
4. Select the hosts to be added from the table and click **Add Selected**.

For CLI Users

To add hosts to a host collection, enter the following command:

```
# hammer host-collection add-host \
--id hc_ID \
--host-ids host_ID1,host_ID2...
```

7.5. REMOVING A HOST FROM A HOST COLLECTION

The following procedure shows how to remove hosts from host collections.

To Remove Hosts from a Host Collection:

1. Click **Hosts** > **Host Collections**.
2. Choose the desired host collection.
3. On the **Hosts** tab, select the **List/Remove** subtab.
4. Select the hosts you want to remove from the host collection and click **Remove Selected**.

7.6. ADDING CONTENT TO A HOST COLLECTION

These steps show how to add content to host collections in Red Hat Satellite.

7.6.1. Adding Packages to a Host Collection

The following procedure shows how to add packages to host collections.

Prerequisites

- The content to be added should be available in one of the existing repositories or added prior to this procedure.
- Content should be promoted to the environment where the hosts are assigned.

To Add Packages to Host Collections:

1. Click **Hosts** > **Host Collections**.
2. Click the host collection where the package should be added.
3. On the **Collection Actions** tab, click **Package Installation, Removal, and Update**
4. To update all packages, click the **Update All Packages** button to use the default method. Alternatively, select the drop-down icon to the right of the button to select a method to use. Selecting the **via remote execution - customize first** menu entry will take you to the **Job invocation** page where you can customize the action.
5. Select the **Package** or **Package Group** radio button as required.
6. In the field provided, specify the package or package group name. Then click:
   - **Install** – to install a new package using the default method. Alternatively, select the drop-down icon to the right of the button and select a method to use. Selecting the **via remote execution - customize first** menu entry will take you to the **Job invocation** page where you can customize the action.
   - **Update** – to update an existing package in the host collection using the default method. Alternatively, select the drop-down icon to the right of the button and select a method to use. Selecting the **via remote execution - customize first** menu entry will take you to the **Job invocation** page where you can customize the action.

7.6.2. Adding Errata to a Host Collection

The following procedure shows how to add errata to host collections.

Prerequisites

- The errata to be added should be available in one of the existing repositories or added prior to this procedure.
- Errata should be promoted to the environment where the hosts are assigned.

To Add Errata to a Host Collection:

1. Click **Hosts** > **Host Collections**.
2. Select the host collection where the errata should be added.

3. On the **Collection Actions** tab, click **Errata Installation**.

4. Select the errata you want to add to the host collection and click the **Install Selected** button to use the default method. Alternatively, select the drop-down icon to the right of the button to select a method to use. Selecting the **via remote execution - customize first** menu entry will take you to the **Job invocation** page where you can customize the action.

7.7. REMOVING CONTENT FROM A HOST COLLECTION

The following procedure shows how to remove packages from host collections.

To Remove Content from a Host Collection:

1. Click **Hosts** > **Host Collections**.

2. Click the host collection where the package should be removed.

3. On the **Collection Actions** tab, click **Package Installation, Removal, and Update**

4. Select the **Package** or **Package Group** radio button as required.

5. In the field provided, specify the package or package group name.

6. Click the **Remove** button to remove the package or package group using the default method. Alternatively, select the drop-down icon to the right of the button and select a method to use. Selecting the **via remote execution - customize first** menu entry will take you to the **Job invocation** page where you can customize the action.

7.8. CHANGING THE LIFE CYCLE ENVIRONMENT OR CONTENT VIEW OF A HOST COLLECTION

The following procedure shows how to change the assigned life cycle environment or Content View of host collections.

To Change the Life Cycle Environment or Content View of a Host Collection:

1. Click **Hosts** > **Host Collection**.

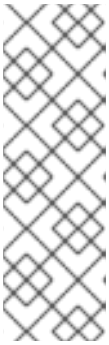
2. Selection the host collection where the life cycle environment or Content View should be changed.

3. On the **Collection Actions** tab, click **Change assigned Life Cycle Environment or Content View**.

4. Select the life cycle environment to be assigned to the host collection.

5. Select the required Content View from the list.

6. Click **Assign**.



NOTE

The changes take effect in approximately 4 hours. To make the changes take effect immediately, on the host, enter the following command:

```
# subscription-manager refresh
```

You can use remote execution to run this command on multiple hosts at the same time.

CHAPTER 8. USING ANSIBLE ROLES

8.1. ASSIGNING ANSIBLE ROLES TO AN EXISTING HOST

You can use Ansible roles for remote management of Red Hat Enterprise Linux versions 8, 7, and 6.9 or later.

Prerequisites

You must import the roles to Satellite before you can assign them to a host. For more information, see Adding Red Hat Enterprise Linux System Roles in *Administering Red Hat Satellite* .

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All Hosts**.
2. On the host you want to assign an Ansible role to, click **Edit**.
3. Select the **Ansible Roles** tab, and in the **All items** list, search for the roles that you want to add.
4. Select the roles that you want to add, and click the arrow icon to move the roles to the **Selected items** list.
5. Click **Submit**.

After you assign Ansible roles to hosts, you can use Ansible for remote execution. For more information, see Section 9.1, “Establishing a Secure Connection for Remote Commands” .

Overiding Parameter Variables

On the **Parameters** tab, click **Add Parameter** to add any parameter variables that you want to pass to job templates at run time. This includes all Ansible playbook parameters and host parameters that you want to associate with the host. To use a parameter variable with an Ansible job template, you must add a **Host Parameter**.

8.2. RUNNING ANSIBLE ROLES ON A HOST

You can run Ansible roles on a host through the Satellite web UI.

Prerequisites

- You must have imported the Ansible roles to Satellite.
- You must have assigned the Ansible roles to the host.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All Hosts**.
2. Select the check box of the host that contains the Ansible role you want to run.
3. From the **Select Action** list, select **Play Ansible roles**.

You can view the status of your Ansible job on the **Run Ansible roles** page. To rerun a job, click the **Rerun** button.

8.3. ASSIGNING AN ANSIBLE ROLE TO A HOST GROUP

You can use Ansible roles for remote management of Red Hat Enterprise Linux versions 8, 7, and 6.9 or later.

Prerequisites

You must import the roles to Satellite before you can assign them to a host group. For more information, see Adding Red Hat Enterprise Linux System Roles in *Administering Red Hat Satellite* .

Procedure

1. In the Satellite web UI, navigate to **Configure** > **Host Groups**.
2. From the list of host groups, click the host group name that you want to add an Ansible Role to.
3. Select the **Ansible Roles** tab, and in the **All items** list, search for the roles that you want to add.
4. Select the roles that you want to add, and click the arrow icon to move the roles to the **Selected items** list.
5. Click **Submit**.

8.4. RUNNING ANSIBLE ROLES ON A HOST GROUP

You can run Ansible roles on a host group through the Satellite web UI.

Prerequisites

- You must have imported the Ansible roles to Satellite.
- You must have assigned the Ansible roles to the host group.
- You must have at least one host in your host group.

Procedure

1. In the Satellite web UI, navigate to **Configure** > **Host Groups**.
2. From the list in the **Actions** column for the host group, select **Play Roles**.

You can view the status of your Ansible job on the **Run Ansible roles** page. To rerun a job, click the **Rerun** button.

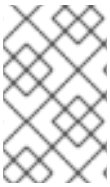
CHAPTER 9. RUNNING JOBS ON HOSTS

Red Hat Satellite supports the ability to run jobs with arbitrary commands on hosts using shell scripts and to run Ansible tasks and playbooks. This is referred to as remote execution.

Remote execution is enabled by default on the Satellite Server, but must be enabled manually on Capsule Servers. For custom Ansible roles that you create, or roles that you download, the package containing the roles must be installed on the Capsule's base system where a task is executed, or a playbook is run. The roles must also be imported into Satellite from the Capsule where they are installed before Satellite can use them.

Communication occurs through the Capsule Server which means that Satellite Server does not require direct access to the target host, and can scale to control many hosts. Remote execution uses the SSH service which must be enabled and running on the target host. Ensure the Capsule has access to port 22 on the target hosts.

Satellite uses ERB syntax job templates. Several job templates for shell scripts and Ansible are included by default. See Section 9.4.1, "Setting up Job Templates".



NOTE

Any Capsule Server's base system is a client of Satellite Server's internal Capsule, and therefore this section applies to any type of host connected to Satellite Server, including Capsule Servers.

You can run jobs on multiple hosts at once, and you can use variables in your commands to suit your deployment. Variable values can be filled by host fact, Smart Class Parameter, Smart Variable, or even host parameter. In addition, you can specify custom values for templates when you run the command. See Section 9.4.2, "Executing Jobs".

By default, each Capsule is installed with the remote execution feature disabled. To enable remote execution, enter the following command:

```
# satellite-installer --scenario capsule \
--enable-foreman-proxy-plugin-remote-execution-ssh
```

To verify that remote execution is enabled on Capsule Server, in the web UI navigate to **Infrastructure** > **Capsules**. To the right of the edit icon, in the **Actions** column, select **Refresh** and confirm the **SSH** feature is listed in the **Features** column.

By default, Satellite Server is configured to use remote execution rather than Katello Agent. These settings can be changed by first creating custom job templates and then selecting these new templates in the web UI by going to **Administer** > **Remote Execution Features**. For each action you want to change, select the label and then select the job template to use.

9.1. ESTABLISHING A SECURE CONNECTION FOR REMOTE COMMANDS

The SSH keys used for remote execution are created automatically when installing a Capsule and the settings are in the **/etc/foreman-proxy/settings.d/remote_execution_ssh.yml** file. They include the following options:

ssh_identity_file

File to load the SSH key from. By default, set to **/usr/share/foreman-proxy/.ssh/id_rsa_foreman_proxy**.

local_working_dir

Directory used on the Satellite or Capsule to run the scripts necessary for remote execution. By default, set to **/var/tmp**.

remote_working_dir

Directory on the client system that is used to execute the remote execution jobs. By default, set to **/var/tmp**.

NOTE

If the client system has **noexec** set for the **/var/** volume or file system, change the **remote_working_dir** as otherwise the remote execution job will fail since the script cannot be run.

If you must use an alternative directory, create the new directory, for example *new_place*, and then copy the SELinux context from the default directory. For example:

```
# chcon --reference=/var new_place
```

See Maintaining SELinux Labels in the *SELinux User's and Administrator's Guide* for more information on working with SELinux labels.

Distributing SSH Keys for Remote Execution

To use SSH keys for authenticating remote execution connections, distribute the public SSH key from a Capsule to its attached hosts that you want to manage. Ensure the SSH service is enabled and running on the hosts. Configure any network or host-based firewalls to enable access to port 22.

There are three ways to distribute the public key from a Capsule to target hosts:

- To distribute SSH keys manually, enter the following command on the Capsule:

  ```
  # ssh-copy-id -i ~foreman-proxy/.ssh/id_rsa_foreman_proxy.pub root@target.example.com
  ```

  Where *target.example.com* is the host name of the target host. Repeat for each target host you want to manage.

  To confirm the key was successfully copied to the target host, enter the following command on the Capsule:

  ```
  # ssh -i ~foreman-proxy/.ssh/id_rsa_foreman_proxy root@target.example.com
  ```

- To use the Satellite API to download the public key directly from the Capsule, enter the following command on each target host:

  ```
  # curl https://capsule.example.com:9090/ssh/pubkey >> ~/.ssh/authorized_keys
  ```

  Where *capsule.example.com* is the host name of the Capsule that the host is attached to.

- To include the public key in newly-provisioned hosts, modify for example the **Kickstart default finish** template to include the following line:

```
<%= snippet 'remote_execution_ssh_keys' %>
```

9.2. CONFIGURING A KEYTAB FOR KERBEROS TICKET GRANTING TICKETS

Use this procedure to configure Satellite to use a keytab to obtain Kerberos ticket granting tickets. If you do not set up a keytab, you must manually retrieve tickets.

To ensure that the **foreman-proxy** user on Satellite can obtain Kerberos ticket granting tickets, complete the following steps:

1. Find the ID of the **foreman-proxy** user:

```
# id -u foreman-proxy
```

2. Modify the **umask** value so that new files have the permissions **600**:

```
# umask 077
```

3. Create the directory for the keytab:

```
# mkdir -p "/var/kerberos/krb5/user/USER_ID"
```

4. Create a keytab or copy an existing keytab to the directory:

```
# cp your_client.keytab /var/kerberos/krb5/user/USER_ID/client.keytab
```

5. Change the directory owner to the **foreman-proxy** user:

```
# chown -R foreman-proxy:foreman-proxy "/var/kerberos/krb5/user/USER_ID"
```

6. Ensure that the keytab file is read-only:

```
# chmod -wx "/var/kerberos/krb5/user/USER_ID/client.keytab"
```

7. Restore the SELinux context:

```
# restorecon -RvF /var/kerberos/krb5
```

9.3. SETTING UP KERBEROS AUTHENTICATION FOR REMOTE EXECUTION

From Satellite 6.6, you can use Kerberos authentication to establish an SSH connection for remote execution on Satellite hosts.

Prerequisites

Before you can use Kerberos authentication for remote execution on Red Hat Satellite, you must set up a Kerberos server for identity management and ensure that you complete the following prerequisites:

- Enroll Satellite Server on the Kerberos server

- Enroll the Satellite target host on the Kerberos server
- Configure and initialize a Kerberos user account for remote execution
- Ensure that the foreman-proxy user on Satellite has a valid Kerberos ticket granting ticket

To set up Satellite to use Kerberos authentication for remote execution on hosts, complete the following steps:

1. Before installing the **tfm-rubygem-net-ssh-krb** package, you must temporarily set SELinux to **permissive** until Red Hat bug 1541481 is resolved:

```
# setenforce 0
```

2. To install the **tfm-rubygem-net-ssh-krb** package, enter the following command:

```
# yum install tfm-rubygem-net-ssh-krb
```

3. To install and enable Kerberos authentication for remote execution, enter the following command:

```
# satellite-installer --scenario satellite \
 --foreman-proxy-plugin-remote-execution-ssh-ssh-kerberos-auth true
```

4. Set SELinux to **enforcing**:

```
# setenforce 1
```

5. To edit the default user for remote execution, in the Satellite web UI, navigate to **Administer** > **Settings** and click the **RemoteExecution** tab. In the **remote_execution_ssh_user** row, edit the second column and add the user name for the Kerberos account.

6. Navigate to **remote_execution_effective_user** and edit the second column to add the user name for the Kerberos account.
   Note that until BZ#1728612 is resolved, **sudo** is the only **become_method** that works with Ansible.

To confirm that Kerberos authentication is ready to use, run a remote job on the host.

9.4. CONFIGURING AND RUNNING REMOTE JOBS

Any command that you want to apply to a remote host must be defined as a job template. After you have defined a job template you can execute it multiple times.

9.4.1. Setting up Job Templates

Satellite provides default job templates that you can use for executing jobs. To view the list of job templates, navigate to **Hosts** > **Job templates**. If want to use a template without making changes, proceed to Section 9.4.2, “Executing Jobs”.

You can use default templates as a base for developing your own. Default job templates are locked for editing. Clone the template and edit the clone.

1. To clone a template, in the **Actions** column, select **Clone**.

2. Enter a unique name for the clone and click **Submit** to save the changes.

Job templates use the Embedded Ruby (ERB) syntax. For more information about writing templates, see the Appendix A, *Template Writing Reference*.

Ansible Considerations

To create an Ansible job template, use the following procedure and instead of ERB syntax, use YAML syntax. Begin the template with **---** and to the first line, you must add **- hosts: all**. You can embed an Ansible playbook YAML file into the job template body. You can also add ERB syntax to customize your YAML Ansible template. You can also import Ansible playbooks in Satellite. For more information, see Synchronizing Templates with Git in the *Content Management Guide*.

Ansible become_method Limitation

Until BZ#1728612 is resolved, for the **remote_execution_effective_user** setting, **sudo** is the only **become_method** that works with Ansible.

Parameter Variables

At run time, job templates can accept parameter variables that you define for a host. Note that only the parameters visible on the **Parameters** tab at the host's edit page can be used as input parameters for job templates. If you do not want your Ansible job template to accept parameter variables at run time, in the Satellite web UI, navigate to **Administer** > **Settings** and click the **Ansible** tab. In the **Top level Ansible variables** row, change the **Value** parameter to **No**.

To Create a Job Template:

1. Navigate to **Hosts** > **Job templates**.
2. Click **New Job Template**.
3. Click the **Template** tab, and in the **Name** field, enter a unique name for your job template.
4. Select **Default** to make the template available for all organizations and locations.
5. Create the template directly in the template editor or upload it from a text file by clicking **Import**.
6. Optional: In the **Audit Comment** field, add information about the change.
7. Click the **Job** tab, and in the **Job category** field, enter your own category or select from the default categories listed in Table 9.1, "Default Job Template Categories".
8. Optional: In the **Description Format** field, enter a description template. For example, **Install package %{package_name}**. You can also use **%{template_name}** and **%{job_category}** in your template.
9. From the **Provider Type** list, select **SSH** for shell scripts and **Ansible** for Ansible tasks or playbooks.
10. Optional: In the **Timeout to kill** field, enter a timeout value to terminate the job if it does not complete.
11. Optional: Click **Add Input** to define an input parameter. Parameters are requested when executing the job and do not have to be defined in the template. For examples, see the **Help** tab.

12. Optional: Click **Foreign input set** to include other templates in this job.
13. Optional: In the **Effective user** area, configure a user if the command cannot use the default **remote_execution_effective_user** setting.
14. Optional: If this template is a snippet to be included in other templates, click the **Type** tab and select **Snippet**.
15. Click the **Location** tab and add the locations where you want to use the template.
16. Click the **Organizations** tab and add the organizations where you want to use the template.
17. Click **Submit** to save your changes.

You can create advanced templates by including other templates in the template syntax, see Section 9.4.4, "Creating Advanced Templates" for more information.

An advanced template is required, for example, for executing jobs that perform power actions; see Example 9.4, "Including Power Actions in Templates" for information on how to include the **Power Action - SSH Default** template in a custom template.

For CLI USers

To create a job template using a template-definition file, enter the following command:

```
# hammer job-template create \
--file "path_to_template_file" \
--name "template_name" \
--provider-type SSH \
--job-category "category_name"
```

Table 9.1. Default Job Template Categories

Job template category	Description
Packages	Templates for performing package related actions. Install, update, and remove actions are included by default.
Puppet	Templates for executing Puppet runs on target hosts.
Power	Templates for performing power related actions. Restart and shutdown actions are included by default.
Commands	Templates for executing custom commands on remote hosts.
Services	Templates for performing service related actions. Start, stop, restart, and status actions are included by default.

Job template category	Description
Katello	Templates for performing content related actions. These templates are used mainly from different parts of the Satellite web UI (for example bulk actions UI for content hosts), but can be used separately to perform operations such as errata installation.

Example 9.1. Creating a restorecon Template

This example shows how to create a template called **Run Command - restorecon** that will restore the default **SELinux** context for all files in the selected directory on target hosts.

1. Navigate to **Hosts** > **Job templates**. Click **New Job Template**.
2. Enter **Run Command - restorecon** in the **Name** field. Select **Default** to make the template available to all organizations. Add the following text to the template editor:

   ```
   restorecon -RvF <%= input("directory") %>
   ```

   The **<%= input("directory") %>** string will be replaced by a user-defined directory during job invocation.
3. On the **Job** tab, set **Job category** to **Commands**.
4. Click **Add Input** to allow job customization. Enter **directory** to the **Name** field. The input name must match the value specified in the template editor.
5. Click **Required** so that the command cannot be executed without the user specified parameter.
6. Select **User input** from the **Input type** list. Enter a description to be shown during job invocation, for example **Target directory for restorecon**.
7. Click **Submit**.

See Example 9.2, "Executing a restorecon Template on Multiple Hosts" for information on how to execute a job based on this template.

9.4.2. Executing Jobs

This section shows how to run a job based on a job template against one or more hosts.

To Execute a Remote Job:

1. Navigate to **Hosts** > **All hosts** and select the target hosts for your job. You can use the search field to filter the host list.
2. From the **Select Action** list, select **Schedule Remote Job**.

3. On the **Job invocation** page, define the main job settings:

   a. Select the **Job category** and the job template you want to use.

   b. Optionally, select a stored search string in the **Bookmark** list to specify the target hosts.

   c. Optionally, further limit the targeted hosts by entering a **Search query**. The **Resolves to** line displays the number of hosts affected by your query. Use the refresh button to recalculate the number after changing the query. The preview icon will list the targeted hosts.

   d. The remaining settings depend on the selected job template. See To Create a Job Template: for information on adding custom parameters to a template.

4. Clicking **Display advanced fields** will show advanced setting for the job. Some of the advanced settings depend on the job template, the following settings are general:

   - **Effective user** defines the user for executing the job, by default it is the SSH user.
   - **Concurrency level** defines maximum number of jobs executed at once, which can prevent overload of systems' resources in a case of executing the job on a large number of hosts.
   - **Time span** defines time interval in seconds after which the job should be killed, if it is not finished already. A task which could not be started during the defined interval, for example, if the previous task took too long to finish, is canceled.
   - **Type of query** defines when the search query is evaluated. This helps to keep the query up to date for scheduled tasks.
     Concurrency level and **Time span** settings enable you to tailor job execution to fit your infrastructure hardware and needs.

5. If you want to run the job immediately, ensure that **Schedule** is set to **Execute now**. You can also define a one-time future job, or set up a recurring job. For recurring tasks, you can define start and end dates, number and frequency of runs. You can also use cron syntax to define repetition. For more information about cron, see the Automating System Tasks section of the Red Hat Enterprise Linux 7 *System Administrator's Guide*.

6. Click **Submit**. This displays the **Job Overview** page, and when the job completes, also displays the status of the job.

For CLI Users

To execute a remote job with custom parameters, complete the following steps:

1. Find the ID of the job template you want to use:

```
# hammer job-template list
```

2. Show the template details to see parameters required by your template:

```
# hammer job-template info --id template_ID
```

3. Execute a remote job with custom parameters:

```
# hammer job-invocation create \
--job-template "template_name" \
--inputs key1="value",key2="value",... \
```

```
--search-query "query"
```

Replace *query* with the filter expression defining which hosts will be affected, for example **"name ~ rex01"**. For more information about executing remote commands with hammer, enter **hammer job-template --help** and **hammer job-invocation --help**.

Example 9.2. Executing a restorecon Template on Multiple Hosts

This example shows how to run a job based on the template created in Example 9.1, "Creating a restorecon Template" on multiple hosts. The job will restore the SELinux context in all files under the **/home/** directory.

1. Navigate to **Hosts** > **All hosts** and select target hosts. Select **Schedule Remote Job** from the **Select Action** list.
2. In the **Job invocation** page, select the **Commands** job category and the **Run Command - restorecon** job template.
3. Type /**home** in the **directory** field.
4. Set **Schedule** to **Execute now**.
5. Click **Submit**. You are taken to the **Job invocation** page where you can monitor the status of job execution.

9.4.3. Monitoring Jobs

You can monitor the progress of the job while it is running. This can help in any troubleshooting that may be required.

To Monitor a Job:

1. Navigate to the Job page. This page is automatically displayed if you triggered the job with the **Execute now** setting. To monitor scheduled jobs, navigate to **Monitor** > **Jobs** and select the job run you wish to inspect.
2. On the Job page, click the **Hosts** tab. This displays the list of hosts on which the job is running.
3. In the **Host** column, click the name of the host that you want to inspect. This displays the **Detail of Commands** page where you can monitor the job execution in real time.
4. Click **Back to Job** at any time to return to the **Job Details** page.

For CLI Users

To monitor the progress of a job while it is running, complete the following steps:

1. Find the ID of a job:

```
# hammer job-invocation list
```

2. Monitor the job output:

```
# hammer job-invocation output \
--id job_ID \
--host host_name
```

3. Optional: to cancel a job, enter the following command:

```
# hammer job-invocation cancel \
--id job_ID
```

9.4.4. Creating Advanced Templates

When creating a job template, you can include an existing template in the template editor field. This way you can combine templates, or create more specific templates from the general ones.

The following template combines default templates to install and start the **httpd** service on Red Hat Enterprise Linux systems:

```
<%= render_template 'Package Action - SSH Default', :action => 'install', :package => 'httpd' %>
<%= render_template 'Service Action - SSH Default', :action => 'start', :service_name => 'httpd' %>
```

The above template specifies parameter values for the rendered template directly. It is also possible to use the **input()** method to allow users to define input for the rendered template on job execution. For example, you can use the following syntax:

```
<%= render_template 'Package Action - SSH Default', :action => 'install', :package =>
input("package") %>
```

With the above template, you have to import the parameter definition from the rendered template. To do so, navigate to the **Jobs** tab, click **Add Foreign Input Set**, and select the rendered template from the **Target template** list. You can import all parameters or specify a comma separated list.

Example 9.3. Rendering a restorecon Template

This example shows how to create a template derived from the **Run command - restorecon** template created in Example 9.1, “Creating a restorecon Template” . This template does not require user input on job execution, it will restore the SELinux context in all files under the **/home/** directory on target hosts.

Create a new template as described in Section 9.4.1, “Setting up Job Templates”, and specify the following string in the template editor:

```
<%= render_template("Run Command - restorecon", :directory => "/home") %>
```

Example 9.4. Including Power Actions in Templates

This example shows how to set up a job template for performing power actions, such as reboot. This procedure prevents Satellite from interpreting the disconnect exception upon reboot as an error, and consequently, remote execution of the job works correctly.

Create a new template as described in Section 9.4.1, “Setting up Job Templates”, and specify the following string in the template editor:

```
<%= render_template("Power Action - SSH Default", :action => "restart") %>
```

9.5. CONFIGURING GLOBAL SETTINGS

The Satellite remote execution feature provides numerous global settings that you can use to configure its behavior. These are listed in Table 9.2, "Global Settings for Remote Execution". To review and update these settings, navigate to **Administer** > **Settings** and click the **Remote Execution** tab.

Table 9.2. Global Settings for Remote Execution

Parameter Name	Description
Default SSH key passphrase	Defines the default key passphrase to use for SSH. You can override this for each host by setting the **remote_execution_ssh_key_passphrase** parameter.
Enable Global Capsule	When enabled, searches for a remote execution Capsule outside of the Capsules assigned to the host. If the location or organization is configured, the search will be limited to the host's organization or location.
Fallback Without Capsule	When enabled, the remote execution will try to run the commands directly, when no Capsule with remote execution feature is configured for the host.
Fallback to Any Capsule	When enabled, searches the host for any Capsule with remote execution configured. This is useful when the host has no subnet configured or if the subnet does not have a Capsule with remote execution enabled.
Effective User Method	Defines which method to use to set the effective user on the target host. One of **sudo**, **dzdo**, **su**.
Effective User	Defines the default effective user for any job. When the job is executed, the effective user of the process is changed accordingly. You can override this for each job template and job invocation.
Connect by IP	When enabled, the IP addresses on host interfaces is preferred over the FQDN. It is useful when DNS is not resolving the FQDNs properly. You can override this for each host by setting the **remote_execution_connect_by_ip** parameter.
SSH User	Defines the default user to use while the Capsule connects to the target using SSH. You can override this for each host by setting the **remote_execution_ssh_user** parameter. You can set this by host, host group, operating system, domain, location, or organization. This can also be a different user from the effective user.
Sudo password	Defines the **sudo** password.

Parameter Name	Description
Default SSH password	Defines the default password to use for SSH. You can override this for each host by setting the **remote_execution_ssh_password** parameter.
SSH Port	Defines the port to use for SSH communication. The default port is 22. You can override this for each host by setting the **remote_execution_ssh_port** parameter.
Sync Job Templates	When enabled, job templates are synchronized from disk when seeding a database.
Cleanup working directories	When enabled, working directories will be removed after task completion. You can override this for each host by setting a **remote_execution_cleanup_working_dirs** parameter.
Workers pool size	Defines the number of workers in the pool to handle the execution of the remote execution jobs. Restart of the **dynflowd/foreman-tasks** service is required.

IMPORTANT

Modify these parameters in the web UI, because any manual changes that you make in the **/etc/foreman/settings.yaml** file are overwritten the next time you run **satellite-installer**. Alternatively, use the **foreman-rake config** command from a console.

9.6. REMOTE EXECUTION AND CAPSULES

By default, Capsules have the **remote execution provider** feature enabled. You can use any Capsule that you add to the host's organization and location to perform remote execution.

In more complex environments, where not all Capsules can be used because of possible network isolation, you can set the **remote_execution_global_proxy** variable to **false** to disable remote execution on Capsules. In this configuration, you can assign a pool of Capsules to each subnet, and jobs are load balanced across them. To use remote execution for a host, any Capsule Server that you want to use for remote execution must reside within the host's subnet. Ensure that the host's interface for that subnet is enabled for remote execution.

Alternatively, you can set the **remote_execution_fallback_proxy** variable to **true** to enable fallback mode. In this configuration, remote execution will use any Capsule associated with the host, such as its Puppet Master, provided that Capsule also has remote execution configured.

9.7. DELEGATING PERMISSIONS FOR REMOTE EXECUTION

You can control which users can run which jobs within your infrastructure, including which hosts they can target. The remote execution feature provides two built-in roles:

- **Remote Execution Manager**: This role allows access to all remote execution features and functionality.

- **Remote Execution User**: This role only allows running jobs; it does not provide permission to modify job templates.

You can clone the Remote Execution User role and customize its filter for increased granularity. If you adjust the filter with the **view_job_templates** permission, the user can only see and trigger jobs based on matching job templates. You can use the **view_hosts** and **view_smart_proxies** permissions to limit which hosts or Capsules are visible to the role.

The **execute_template_invocation** permission is a special permission that is checked immediately before execution of a job begins. This permission defines which job template you can run on a particular host. This allows for even more granularity when specifying permissions. For more information on working with roles and permissions see Creating and Managing Roles in the *Administering Red Hat Satellite*.

The following example shows filters for the **execute_template_invocation** permission:

```
name = Reboot and host.name = staging.example.com
name = Reboot and host.name ~ *.staging.example.com
name = "Restart service" and host_group.name = webservers
```

The first line in the above example permits the user to apply the **Reboot** template to one selected host. The second line defines a pool of hosts with names ending with **.staging.example.com**. The third line binds the template with a host group.

NOTE

Permissions assigned to users can change over time. If a user has already scheduled some jobs to run in the future, and the permissions have changed, this can result in execution failure because the permissions are checked immediately before job execution.

9.8. CONFIGURING ANSIBLE RUNNER

You can use Ansible Runner to improve performance when running Ansible playbooks on multiple hosts at once.

Because Ansible Runner automates Ansible playbook runs in batches of 100 hosts, you cannot cancel a job running on a specific host. A job completes only after the Ansible playbook runs on all hosts in the batch.

Prerequisite

You must enable remote execution in Satellite. For more information, see Chapter 9, *Running Jobs on Hosts*.

Procedure

To install and configure Satellite to use Ansible Runner, complete the following steps:

1. On Satellite Server, enter the following command to install the **ansible-runner** package on the integrated Capsule:

```
# foreman-maintain packages install ansible-runner
```

2. In the Satellite web UI, navigate to **Administer** > **Settings** and click the **Ansible** tab.

3. In the **Implementation for running Ansible** row, change the **Value** parameter to **ansible-runner**.

4. Optional: if you want to install Ansible Runner on an external Capsule, enter the following command on Capsule Server:

```
# yum install ansible-runner
```

CHAPTER 10. DISCOVERING BARE-METAL HOSTS ON SATELLITE

Red Hat Satellite 6.6 includes the Discovery plug-in. The Discovery plug-in enables automatic bare-metal discovery of unknown hosts on the provisioning network. These new hosts are registered to the Satellite Server and the Puppet agent on the client uploads system facts collected by Facter, such as serial ID, network interface, memory, and disk information. After registration, the hosts are displayed on the **Discovered Hosts** page in the Satellite web UI. You can then initiate provisioning either manually (using the web UI, CLI, or API) or automatically, using predefined discovery rules.

The Discovery plug-in communicates through the Satellite Capsule Server, which has direct access both to the provisioning network and the Satellite Server instance. It is possible to discover hosts directly from the Satellite Server, but Red Hat recommends the following scheme be used:

> Satellite Server (Satellite Server Discovery plug-in) <--> Satellite Capsule (Satellite Capsule Discovery plug-in) <--> Discovered Host (Satellite Discovery image)

The Satellite Discovery plug-in consists of three different components:

The Satellite Server Discovery plug-in

This runs on the Satellite Server and provides API and UI functionality for working with discovered hosts. The **tfm-rubygem-foreman_discovery** package contains this plug-in.

The Satellite Capsule Server Discovery plug-in

This is a communication proxy between discovered hosts on a provisioning network and the Satellite Server. The **rubygem-smart_proxy_discovery** package contains this plug-in.

The Satellite Discovery image

This is the minimal operating system based on Red Hat Enterprise Linux that is PXE-booted on hosts to acquire initial hardware information and to check in to the Satellite Server. Discovered hosts keep running the Satellite Discovery image until they are rebooted into Anaconda, which then initiates the provisioning process. The **foreman-discovery-image** package contains this image. It must be installed on the Satellite Capsule Server that provides TFTP services.

10.1. NETWORK CONFIGURATION FOR PXE-BASED DISCOVERY

The discovery process is based on PXE: Systems must boot from the network using a single Ethernet connection to the LAN or VLAN. All other network interface configurations are not supported (bonding, teaming, bridging, DSL, Wi-Fi and others).

You must have a separate LAN or VLAN for discovery and PXE provisioning. You can configure systems to use VLAN trunks, but you must also configure the provisioning interface with the correct VLAN tag for the provisioning VLAN, and then change the tag to the production VLAN using a post-installation script.

Although using special network configurations is technically possible in PXE-less mode, where discovered systems use kexec to load a new kernel with Anaconda which avoids PXE booting completely, the discovery image currently does not allow such a configuration. While it is possible to use discovery extensions or a script to re-configure the network, Satellite 6 discovery plug-in cannot work with such a configuration.

Because the discovery process currently has limited possibilities for configuring network interfaces, and because the provisioning interface is also the primary interface, to simplify the configuration, have a separate primary interface from the interface used in production. Satellite 6 template features can be

used to deploy post-installation scripts to configure interfaces if required.

10.2. CONFIGURING THE SATELLITE DISCOVERY PLUG-IN

The following sections describe how to configure the Satellite Discovery plug-in and how to prepare the PXE-boot template on the Satellite Server.

10.2.1. Deploying the Satellite Discovery Image

Install the package containing the Satellite Discovery image on the Satellite Capsule Server that provides TFTP services (not on the Satellite Server itself):

```
# yum install foreman-discovery-image
```

This package contains the Linux kernel and initial RAM disk image as a bootable ISO file which is used for PXE-booting discovered hosts. You can run the following command to investigate the contents of the package. This produces output similar to the following:

```
$ rpm -ql foreman-discovery-image
/usr/share/foreman-discovery-image
/usr/share/foreman-discovery-image/fdi-image-rhel_7-2.1.0-20150212.1.iso
```

When you install this package, it extracts the kernel and image from the ISO file into the TFTP directory and creates symbolic links to the latest versions of the image and kernel. Use the symbolic links in the PXE-boot provisioning template to make sure that you do not need to change the version in the template every time the **foreman-discovery-image** package is upgraded. For example:

```
$ find /var/lib/tftpboot/boot
/var/lib/tftpboot/boot
/var/lib/tftpboot/boot/fdi-image-rhel_7-2.1.0-20150212.1-img
/var/lib/tftpboot/boot/fdi-image-rhel_7-2.1.0-20150212.1-vmlinuz
/var/lib/tftpboot/boot/fdi-image-rhel_7-img
/var/lib/tftpboot/boot/fdi-image-rhel_7-vmlinuz
```

NOTE

Currently, only Red Hat Enterprise Linux 7 Discovery images are provided, even for Satellite 6 installations on Red Hat Enterprise Linux 6. If there are discovered hosts running during the upgrade of the **foreman-discovery-image** package, reboot them all to load the updated version of the image as soon as possible. This can be done using the Satellite 6 web UI, CLI, or API.

10.2.2. Configuring PXE-booting

When an unknown host is booted on the provisioning network, Satellite Server provides a PXELinux boot menu with a single option: to boot from the local hard drive. You can use following procedure to build a default PXE template in Satellite to enable hardware discovery.

To Configure PXE-booting for host discovery:

1. In the Satellite web UI, navigate to **Hosts** > **Provisioning Templates**.
2. In the upper-right of the **Provisioning Templates** page, click **Build PXE Default**, and click **OK**.

The template becomes the default template on all TFTP servers. Every new unknown host that is in the provisioning subnet uses this configuration and uses the Foreman Discovery Image as the default.

10.2.3. Reviewing Global Discovery Settings

You can review global settings related to the Discovery plug-in in the Satellite web UI. Navigate to **Administer** > **Settings** and open the **Discovered** tab. Notable settings are:

Discovery organization, Discovery location

These variables specify where to place the discovered hosts. By default, the discovered hosts are automatically placed under the first organization and location created.

Interface fact

This variable specifies which incoming fact to use to determine the MAC address of the discovered host. By default, the PXELinux BOOTIF kernel command line option is used.

Hostname facts

This variable allows you to list facts to use for the host name. These are separated by commas, and the first fact in the list takes precedence.

Auto provisioning

This variable enables automatic provisioning according to specified rules. Set to false by default. Red Hat recommends that you test the configuration with manual provisioning before enabling Auto provisioning. See Section 10.4, “Provisioning Discovered Hosts” for more information.

Reboot

This variable enables automatic reboot of a host discovered by PXE, or the use of kexec for a host booted from local media, during provisioning. This is set to true by default.

Hostname prefix

This variable specifies the default prefix to use for the host name. Set to "mac" by default. The variable must start with a letter.

Fact columns

This variable allows you to add any fact reported by Facter as an additional column in discovered host lists.

Highlighted facts

This variable uses regular expressions to organize facts for the highlights section.

Storage facts

This variable uses regular expressions to organize facts for the storage section.

Hardware facts

This variable uses regular expressions to organize facts for the hardware section.

Network facts

This variable uses regular expressions to organize facts for the network section.

IPMI facts

This variable uses regular expressions to organize facts for the IPMI section.

10.3. CONFIGURING THE SATELLITE CAPSULE SERVER DISCOVERY PLUG-IN

Ensure the foreman_url setting exists in the Satellite Capsule Server configuration file. The setting can appear as follows:

```
# grep foreman_url /etc/foreman-proxy/settings.yml
:foreman_url: https://satellite.example.com
```

The **satellite-installer** command configures this variable automatically, but Red Hat recommends that you check that the host responds correctly and there are no firewall rules blocking communication.

10.3.1. Configuring Discovery Subnets

You need to configure all subnets with discovered hosts to communicate to the Satellite Server or a Capsule Server for host discovery and provisioning. You must first enable a Capsule to provide a proxy service for provisioning templates for the subnet. The template Capsule for a subnet does not have to be the same Capsule that is set as the TFTP Capsule.

If you want to configure a new subnet, see Adding a Subnet to the Satellite Server in the *Provisioning Guide*.

To enable a Template Capsule:

You can enable the Satellite's integrated Capsule and any external Capsule Servers to provide a proxy service for provisioning templates with the following command:

```
# satellite-installer --foreman-proxy-templates=1
```

To verify that a Capsule Server provides a template proxy service:

1. Navigate to **Infrastructure** > **Capsules**.
2. From the **Actions** list, select **Refresh** to ensure that the list is up-to-date.
3. In the **Features** column, search for the word "Templates".

To Configure a Subnet with a TFTP Capsule, Template Capsule, and Discovery Capsule:

1. In the Satellite web UI, navigate to **Infrastructure** > **Subnets**.
2. Select the subnet you want to configure.
3. On the **Capsules** tab, select the **TFTP Capsule**, **Template Capsule**, and **Discovery Capsule** for this subnet.

10.3.2. Using Hammer with the Discovery Plug-in

To use the **hammer** command with the Discovery plug-in, you need to enable the Discovery plug-in in **/etc/hammer/cli.modules.d/foreman_discovery.yml** as follows:

```
:foreman_discovery:
  :enable_module: true
```

See hammer configuration directories for more information about the files and directories that **hammer** uses.

10.3.3. Reviewing User Permissions

When it first starts, the Satellite Capsule Server Discovery plug-in creates a role called **Discovery**. You can assign this role to non-administrative users to allow them to use the Discovery plug-in. Alternatively, assign the **perform_discovery** permission to an existing role. For more information on roles and permissions, see Creating and Managing Users in *Administering Red Hat Satellite* .

10.4. PROVISIONING DISCOVERED HOSTS

After you have correctly configured Discovery plug-ins on both Satellite Server and Capsule Server, you can automatically detect bare-metal hosts. To do so, boot a machine in any provisioning network that was configured with the PXE configuration template described in Section 10.2.2, "Configuring PXE-booting". The machine is automatically registered with the Satellite Server and appears in the **Hosts** > **Discovered Hosts** list in the Satellite web UI.

You can either provision the discovered host manually, or you can configure automatic provisioning.

10.4.1. Manually Provisioning Hosts

The following procedure describes how to manually provision discovered hosts from the Satellite web UI.

To Manually Provision a Discovered Host:

1. Navigate to **Hosts** > **Discovered Hosts**.
2. Select the host you want to provision and click **Provision**.
3. On the host's **Edit** page, complete the necessary details, and then click **Save**.

When the host configuration is saved, Satellite modifies the host's PXELinux file on the TFTP server and reboots the discovered host. It then boots into an installer for the chosen operating system, and finally into the installed operating system.

If you decide to re-provision an existing discovered host, delete the operating system from the machine and reboot it. The host then reappears on the **Discovered Hosts** page.

10.4.2. Decommissioning Discovered Hosts

If you no longer require Red Hat Satellite to manage a specific host, you need to decommission that host to prevent it from being discovered.

To Decommission a Discovered Host:

1. Shut down the host.
2. Navigate to **Hosts** > **Discovered Hosts**.
3. In the **Name** column find the host you want to decommission and then select **Delete** from the corresponding **Edit** list.

10.4.3. Automatically Provisioning Hosts

With Satellite 6.6, it is possible to define provisioning rules that will assign a host group to provisioned hosts and trigger provisioning automatically.

To Create a Provisioning Rule:

1. Navigate to **Configure** > **Discovery rules**.
2. Click **New Rule**. Specify the following parameters of the provisioning rule:
   - **Name** is the name of the rule displayed in the list of rules. This name must not contain spaces or non-alphanumeric characters.
   - **Search** is the search statement used to match discovered hosts for the particular rule. You can use scoped search syntax to define it. See Section 10.4.4, "Scoped Search Syntax" for examples of using scoped search.
   - **Host Group** is the host group to be assigned to a matching host before starting the provisioning process. Make sure that the selected host group has all the required parameters set; required parameters are marked with an asterisk (*).
   - **Hostname** defines a pattern for assigning human-readable host names to the matching hosts. When left blank, the host name is assigned in the format "macMACADDRESS" by default. The same syntax used for provisioning templates is used in this instance. See Section 10.4.5, "Host Name Patterns" for more information and examples.
   - **Hosts limit** is the maximum number of provisioned hosts per rule. If the limit is reached, the rule will not take effect until one or more hosts are deleted. Typical use cases are rules per server rack or row when it is necessary to change provisioning parameters such as host name or host group per entry. You can set this value to zero (0) to specify no limit.
   - **Priority** specifies the order of execution of rules. The value must be greater than or equal to zero. A lower value indicates a higher priority. If two rules have the same priority, the first rule encountered is applied.
   - **Enabled** provides the option to temporarily enable or disable rules.
3. Click **Submit** to save the rule.

By default, Satellite does not enable automatic discovery of hosts. The following procedure describes how to enable the Auto provisioning variable to provide automatic provisioning according to specified rules.

To Enable Automatic Provisioning:

1. Navigate to **Administer** > **Settings** > **Discovered** in the Satellite web UI.
2. Locate Auto provisioning in the **Name** column, and set its value to **true**.
3. Click **Save**.

After you have defined some rules, Red Hat recommends that you discover a host and apply the rules using the **Auto discover** button on the host. This triggers auto-provisioning without the need to enable the global option.

10.4.4. Scoped Search Syntax

This section shows how to use scoped search syntax to filter the discovered hosts according to selected parameters. This is useful when creating a rule for automatic provisioning (see Section 10.4.3, "Automatically Provisioning Hosts").

The search fields in the Satellite web UI support automatic completion to make building search strings easier. For example, you can test search patterns on the **Hosts** > **Discovered Hosts** page. The following are examples of typical search queries:

- facts.architecture = x86_64
- facts.bios_vendor ~ 'Dell*'
- facts.macaddress = "aa:bb:cc:dd:ee:ff"
- facts.macaddress_eth0 = "aa:bb:cc:dd:ee:ff"
- facts.ipaddress_eth1 ~ "192.168.*"
- facts.architecture ^ (x86_64,i386)

NOTE

The caret symbol (^) in scoped searches means "in" (the same usage as in SQL) and not "starts with" as it is used in regular expressions. You can review the full list of scoped search operators at https://github.com/wvanbergen/scoped_search/blob/master/lib/scoped_search/query_lar

In Satellite 6.6, all facts are strings, so it is not possible to do numeric comparisons. However, three important facts are extracted and converted to numbers. These are described in Table 10.1, "Facts that Allow Numerical Comparison".

Table 10.1. Facts that Allow Numerical Comparison

Search Parameter	Description	Example Usage
cpu_count	The number of CPUs	cpu_count >= 8
disk_count	The number of disks attached	disk_count < 10
disks_size	The total amount of disk space (in MiB)	disks_size > 1000000

10.4.5. Host Name Patterns

This section lists the host name patterns that you can use when creating a rule for automatic provisioning (see Section 10.4.3, "Automatically Provisioning Hosts").

The target host name template pattern has the same syntax as the provisioning templates (ERB). The domain is appended automatically. In addition to the **@host** attribute, the **rand()** function for random integers is available. For example:

- application-server-<%= rand(99999) %>
- load-balancer-<%= @host.facts['bios_vendor'] + '-' + rand(99999) %>

- wwwsrv-<%= @host.hostgroup.name %>
- minion-<%= @host.discovery_rule.name %>
- db-server-<%= @host.ip.gsub('.','-') + '-' + @host.hostgroup.subnet.name %>

IMPORTANT

When creating host name patterns, ensure the resulting host names are unique. Host names must not start with numbers. A good approach is to use unique information provided by Facter (for example, the MAC address, BIOS or serial ID) or to otherwise randomize the host name.

10.4.6. Using the Discovery Plug-in on the Command Line

You can use the **hammer** command to perform certain tasks related to discovery. Run the **hammer -h** command to verify your configuration:

```
$ hammer -h | grep discovery
discovery                Manipulate discovered hosts.
discovery_rule             Manipulate discovered rules.
```

Use the **hammer discovery -h** command to view the available options. For example, you can use the following command to reboot a discovered host (assuming its ID is 130):

```
$ hammer discovery reboot -id 130
Host reboot started
```

10.5. EXTENDING THE DISCOVERY IMAGE

It is possible to extend the Satellite Discovery image with custom facts, software, or device drivers. You can also provide a compressed archive file containing extra code for the image to use.

First, create the following directory structure:

```
.
├── autostart.d
│   └── 01_zip.sh
├── bin
│   └── ntpdate
├── facts
│   └── test.rb
└── lib
    ├── libcrypto.so.1.0.0
    └── ruby
        └── test.rb
```

Where:

- The **autostart.d** directory contains scripts that are executed in POSIX order by the image when it starts, but before the host is registered to Satellite.
- The **bin** directory is added to the $PATH variable; you can place binary files here and use them in the autostart scripts.

- The **facts** directory is added to the FACTERLIB variable so that custom facts can be configured and sent to Satellite.
- The **lib** directory is added to the LD_LIBRARY_PATH variable and **lib/ruby** is added to the RUBYLIB variable, so that binary files in **/bin** can be executed correctly.

New directives and options are appended to the existing environment variables (PATH, LD_LIBRARY_PATH, RUBYLIB and FACTERLIB). If you need to specify the path to something explicitly in your scripts, the zip contents are extracted to the **/opt/extension** directory on the image.

After creating the above directory structure, package it into a zip archive with the following command:

```
zip -r my_extension.zip .
```

You can create multiple zip files but be aware they will be extracted to the same place on the Discovery image, so files in later zips will overwrite earlier ones if they have the same file name.

To inform the Discovery image of the extensions it should use, place your zip files on your TFTP server with the Discovery image, and then update the APPEND line of the PXELinux template with the **fdi.zips** option where the paths are relative to the TFTP root. For example, if you have two archives at **$TFTP/zip1.zip** and **$TFTP/boot/zip2.zip**, use the following syntax:

```
fdi.zips=zip1.zip,boot/zip2.zip
```

See Section 10.2.2, “Configuring PXE-booting” for more information on updating the PXE template.

10.6. TROUBLESHOOTING SATELLITE DISCOVERY

If a machine is not listed in the Satellite web UI under **Hosts** > **Discovered Hosts**, inspect the following configuration areas to help isolate the error:

- Navigate to **Hosts** > **Provisioning Templates** and redeploy the default PXELinux template using the **Build PXE Default** button.
- Verify the **pxelinux.cfg/default** configuration file on the TFTP Capsule Server.
- Ensure adequate network connectivity between hosts, Capsule Server, and Satellite Server.
- Check the PXELinux template in use and determine the PXE discovery snippet it includes. Snippets are named as follows: **pxelinux_discovery**, **pxegrub_discovery**, or **pxegrub2_discovery**. Verify the **proxy.url** and **proxy.type** options in the PXE discovery snippet.
- Ensure that DNS is working correctly for the discovered nodes, or use an IP address in the **proxy.url** option in the PXE discovery snippet included in the PXELinux template you are using.
- Ensure that the DHCP server is delivering IP addresses to the booted image correctly.
- Ensure the discovered host or virtual machine has at least 1200 MB of memory. Less memory can lead to various random kernel panic errors as the image needs to be extracted in-memory.

For gathering important system facts, use the **discovery-debug** command. It prints out system logs, network configuration, list of facts, and other information on the standard output. The typical use case is to redirect this output and copy it with the **scp** command for further investigation.

The first virtual console on the discovered host is reserved for **systemd** logs. Particularly useful system logs are tagged as follows:

- discover-host – initial facts upload
- foreman-discovery – facts refresh, reboot remote commands
- nm-prepare – boot script which pre-configures NetworkManager
- NetworkManager – networking information

Use TTY2 or higher to log in to a discovered host. The root account and SSH access are disabled by default, but you can enable SSH and set the root password using the following kernel command-line options in the Default PXELinux template on the APPEND line:

```
fdi.ssh=1 fdi.rootpw=redhat
```

CHAPTER 11. INTEGRATING RED HAT SATELLITE AND ANSIBLE TOWER

You can integrate Red Hat Satellite 6.6 and Ansible Tower to use Satellite Server as a dynamic inventory source for Ansible Tower.

You can also use the provisioning callback function to run playbooks on hosts managed by Satellite, from either the host or Ansible Tower. When provisioning new hosts from Satellite Server, you can use the provisioning callback function to trigger playbook runs from Ansible Tower. The playbook configures the host following Kickstart deployment.

11.1. ADDING SATELLITE SERVER TO ANSIBLE TOWER AS A DYNAMIC INVENTORY ITEM

To add Satellite Server to Ansible Tower as a dynamic inventory item, you must create a credential for a Satellite Server user on Ansible Tower, add an Ansible Tower user to the credential, and then configure an inventory source.

Prerequisites

- If your Satellite deployment is large, for example, managing tens of thousands of hosts, using a non-admin user can negatively impact performance because of time penalties that accrue during authorization checks. For large deployments, consider using an admin user.
- For non-admin users, you must assign the **Ansible Tower Inventory Reader** role to your Satellite Server user. For more information about managing users, roles, and permission filters, see Creating and Managing Roles in *Administering Red Hat Satellite*.
- You must host your Satellite Server and Ansible Tower on the same network or subnet.

Procedure

To add Satellite Server to Ansible Tower as a Dynamic Inventory Item, complete the following procedure:

1. In the Ansible Tower web UI, create a credential for your Satellite. For more information about creating credentials, see Add a New Credential and Red Hat Satellite 6 Credentials in the *Ansible Tower User Guide*.

   Table 11.1. Satellite Credentials

Credential Type:	Red Hat Satellite 6
Satellite 6 URL:	https://*satellite.example.com*
Username:	The username of the Satellite user with the integration role.
Password:	The password of the Satellite user.

2. Add an Ansible Tower user to the new credential. For more information about adding a user to a credential, see Getting Started with Credentials in the *Ansible Tower User Guide*.
3. Add a new inventory. For more information, see Add a new inventory in the *Ansible Tower User Guide*.

4. In the new inventory, add Satellite Server as the inventory source, specifying the following inventory source options. For more information, see Add Source in the *Ansible Tower User Guide*.

Table 11.2. Inventory Source Options

Source	Red Hat Satellite 6
Credential	The credential you create for Satellite Server.
Overwrite	Select
Overwrite Variables	Select
Update on Launch	Select
Cache Timeout	*90*

5. Ensure that you synchronize the source that you add.

11.2. CONFIGURING PROVISIONING CALLBACK FOR A HOST

When you create hosts in Satellite, you can use Ansible Tower to run playbooks to configure your newly created hosts. This is called *provisioning callback* in Ansible Tower.

The provisioning callback function triggers a playbook run from Ansible Tower as part of the provisioning process. The playbook configures the host after Kickstart deployment.

For more information about provisioning callbacks, see Provisioning Callbacks in the *Ansible Tower User Guide*.

In Satellite Server, the **Kickstart Default** and **Kickstart Default Finish** templates include three snippets:

1. **ansible_provisioning_callback**
2. **ansible_tower_callback_script**
3. **ansible_tower_callback_service**

You can add parameters to hosts or host groups to provide the credentials that these snippets can use to run Ansible playbooks on your newly created hosts.

Prerequisites

Before you can configure provisioning callbacks, you must add Satellite as a dynamic inventory in Ansible Tower. For more information, see Integrating Satellite and Ansible Tower .

In the Ansible Tower web UI, you must complete the following tasks:

1. Create a machine credential for your new host. Ensure that you enter the same password in the credential that you plan to assign to the host that you create in Satellite. For more information, see Add a New Credential in the *Ansible Tower User Guide*.

2. Create a project. For more information, see Projects in the *Ansible Tower User Guide*.
3. Add a job template to your project. For more information, see Job Templates in the *Ansible Tower User Guide*.
4. In your job template, you must enable provisioning callbacks, generate the host configuration key, and note the *template_ID* of your job template. For more information about job templates, see Job Templates in the *Ansible Tower User Guide*.

Procedure

To configure provisioning callback for a new host in Satellite, complete the following steps:

1. In the Red Hat Satellite web UI, navigate to **Configure** > **Host Group**.
2. Create a host group or edit an existing host group.
3. In the Host Group window, click the **Parameters** tab.
4. Click **Add Parameter**.
5. Enter the following information for each new parameter:

Table 11.3. Host Parameters

Name	Value	Description
ansible_tower_provisioning	true	Enables Provisioning Callback.
ansible_tower_fqdn	*tower.example.com*	The fully qualified domain name (FQDN) of your Ansible Tower. Do not add **https** because this is appended by Ansible Tower.
ansible_job_template_id	*template_ID*	The ID of your provisioning template that you can find in the URL of the template: **/templates/job_template/*5***.
ansible_host_config_key	*config_KEY*	The host configuration key that your job template generates in Ansible Tower.

6. Click **Submit**.
7. Create a host using the host group.
8. On the new host, enter the following command to start the **ansible-callback** service:

```
# systemctl start ansible-callback
```

9. On the new host, enter the following command to output the status of the **ansible-callback** service:

```
# systemctl status ansible-callback
```

Provisioning callback is configured correctly if the command returns the following output:

```
SAT_host systemd[1]: Started Provisioning callback to Ansible Tower...
```

Manual Provisioning Callback

You can use the provisioning callback URL and the host configuration key from a host to call Ansible Tower. For example:

```
# curl -k -s --data curl --insecure --data host_config_key=my_config_key \
 https://tower.example.com/api/v2/job_templates/8/callback/
```

Ensure that you use **https** when you enter the provisioning callback URL.

This triggers the playbook run specified in the template against the host.

CHAPTER 12. SYNCHRONIZING TEMPLATE REPOSITORIES

In Satellite, you can synchronize repositories of job templates, provisioning templates, report templates, and partition table templates between Satellite Server and a version control system or local directory. In this chapter, a Git repository is used for demonstration purposes.

This section details the workflow for:

- installing and configuring the TemplateSync plug-in
- performing exporting and importing tasks

12.1. ENABLING THE TEMPLATESYNC PLUG-IN

1. To enable the plug-in on your Satellite Server, enter the following command:

```
# satellite-installer --enable-foreman-plugin-templates
```

2. To verify that the plug-in is installed correctly, ensure **Administer** > **Settings** includes the **TemplateSync** menu.

12.2. CONFIGURING THE TEMPLATESYNC PLUG-IN

In the Satellite web UI, navigate to **Administer** > **Settings** > **TemplateSync** to configure the plug-in. The following table explains the attributes behavior. Note that some attributes are used only for importing or exporting tasks.

Table 12.1. Synchronizing Templates Plug-in configuration

Parameter	API parameter name	Meaning on importing	Meaning on exporting
Associate	**associate** Accepted values: **always**, **new**, **never**	Associates templates with OS, Organization, and Location based on metadata.	N/A
Branch	**branch**	Specifies the default branch in Git repository to read from.	Specifies the default branch in Git repository to write to.
Dirname	**dirname**	Specifies the subdirectory under the repository to read from.	Specifies the subdirectory under the repository to write to.
Filter	**filter**	Imports only templates with names that match this regular expression.	Exports only templates with names that match this regular expression.
Force import	**force**	Imported templates overwrite locked templates with the same name.	N/A

Parameter	API parameter name	Meaning on importing	Meaning on exporting
Metadata export mode	**metadata_export_mode** Accepted values: **refresh**, **keep**, **remove**	N/A	Defines how metadata is handled when exporting: • **Refresh** – remove existing metadata from the template content and generate new metadata based on current assignments and attributes. • **Keep** – retain the existing metadata. • **Remove** – export template without metadata. Useful if you want to add metadata manually.
Negate	**negate** Accepted values: **true**, **false**	Imports templates ignoring the filter attribute.	Exports templates ignoring the filter attribute.
Prefix	**prefix**	Adds specified string to the beginning of the template if the template name does not start with the prefix already.	N/A
Repo	**repo**	Defines the path to the repository to synchronize from.	Defines the path to a repository to export to.
Verbosity	**verbose** Accepted values: **true**, **false**	Enables writing verbose messages to the logs for this action.	N/A

12.3. IMPORTING AND EXPORTING TEMPLATES

You can use the Satellite API or Hammer CLI to import and export templates. The Satellite API calls use the role-based access control system, which enables the tasks to be executed as any user. You can synchronize templates with a version control system, such as Git, or a local directory.

Prerequisites

For imported templates to appear in the Satellite web UI, each template must contain the location and organization that the template belongs to. This applies to all template types. Before you import a template, ensure that you add the following section to the template:

```
<%#
kind: provision
name: My Kickstart File
oses:
- RedHat 7
- RedHat 6
locations:
- First Location
- Second Location
organizations:
- Default Organization
- Extra Organization
%>
```

12.4. SYNCHRONIZING TEMPLATES USING THE SATELLITE API

1. Configure a version control system that uses SSH authorization, for example gitosis, gitolite, or git daemon.
2. Configure the TemplateSync plug-in settings on a **TemplateSync** tab.
   a. Change the **Branch** setting to match the target branch on a Git server.
   b. Change the **Repo** setting to match the Git repository. For example, for the repository located in **git@git.example.com/templates.git** set the setting into **ssh://git@git.example.com/templates.git**.
3. Accept Git SSH host key as the **foreman** user:

   ```
   # sudo -u foreman ssh git.example.com
   ```

   You can see the **Permission denied, please try again.** message in the output, which is expected, because the SSH connection cannot succeed yet.
4. Create an SSH key pair if you do not already have it. Do not specify a passphrase.

   ```
   # sudo -u foreman ssh-keygen
   ```
5. Configure your version control server with the public key from your Satellite, which resides in **/usr/share/foreman/.ssh/id_rsa.pub**.
6. Export templates from your Satellite Server to the version control repository specified in the **TemplateSync** menu:

   ```
   $ curl -H "Accept:application/json,version=2" \
   ```

```
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/export \
-X POST

{"message":"Success"}
```

7. Import templates to Satellite Server after their content was changed:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/import \
-X POST

{“message”:”Success”}
```

Note that templates provided by Satellite are locked and you cannot import them by default. To overwrite this behavior, change the **Force import** setting in the **TemplateSync** menu to **yes** or add the **force** parameter **-d '{ "force": "true" }'** to the import command.

12.5. SYNCHRONIZING TEMPLATES WITH A LOCAL DIRECTORY USING THE SATELLITE API

Synchronizing templates with a local directory is useful if you have configured a version control repository in the local directory. That way, you can edit templates and track the history of edits in the directory. You can also synchronize changes to Satellite Server after editing the templates.

1. Create the directory where templates are stored and apply appropriate permissions and SELinux context:

```
# mkdir -p /usr/share/templates_dir/
# chown foreman /usr/share/templates_dir/
# chcon -t httpd_sys_rw_content_t /usr/share/templates_dir/ -R
```

2. Change the **Repo** setting on the **TemplateSync** tab to match the export directory **/usr/share/templates_dir/**.

3. Export templates from your Satellite Server to a local directory:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/export \
-X POST \

{"message":"Success"}
```

4. Import templates to Satellite Server after their content was changed:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/import \
```

```
-X POST

{“message”:”Success”}
```

Note that templates provided by Satellite are locked and you cannot import them by default. To overwrite this behavior, change the **Force import** setting in the **TemplateSync** menu to **yes** or add the **force** parameter **-d '{ "force": "true" }'** to the import command.

NOTE

You can override default API settings by specifying them in the request with the **-d** parameter. The following example exports templates to the **git.example.com/templates** repository:

```
$ curl -H "Accept:application/json,version=2" \
-H "Content-Type:application/json" \
-u login:password \
-k https://satellite.example.com/api/v2/templates/export \
-X POST \
-d "{\"repo\":\"git.example.com/templates\"}"
```

12.6. IMPORTING TEMPLATES USING THE HAMMER CLI

You can use the **hammer import-templates** command to import templates from a repository of your choice. You can use different protocols to point to your repository, for example **/tmp/dir**, **git://example.com**, **https://example.com**, and **ssh://example.com**.

For better indexing and management of your templates, use **--prefix** to set a category for your templates. To select certain templates from a large repository, use **--filter** to define the title of the templates that you want to import. For example **--filter '.*Ansible Default$'** imports various Ansible Default templates.

Procedure

To import a template from a repository, enter the following command:

```
$ hammer import-templates \
  --prefix '[Custom Index] ' \
  --filter '.*Template Name$' \
  --repo https://github.com/examplerepo/exampledirectory \
  --branch my_branch \
  --organization 'Default Organization'
```

12.7. EXPORTING TEMPLATES USING THE HAMMER CLI

You can use the **hammer export-templates** command to export templates to a version control server, such as a Git repository.

Procedure

1. Clone a local copy of your Git repository:

```
$ git clone https://github.com/theforeman/community-templates /custom/templates
```

2. Change the owner of your local directory to the **foreman** user, and change the SELinux context with the following commands:

```
# chown -R foreman:foreman /custom/templates
# chcon -R -t httpd_sys_rw_content_t /custom/templates
```

3. To export the templates to your local repository, enter the following command:

```
hammer export-templates --organization 'Default Organization' --repo /custom/templates
```

12.8. ADVANCED GIT CONFIGURATION

You can perform additional Git configuration for the TemplateSync plug-in using the command line or editing the **.gitconfig** file.

Accepting a self-signed Git certificate

If you are using a self-signed certificate authentification on your Git server, validate the certificate with the **git config http.sslCAPath** command.

For example, the following command verifies a self-signed certificate stored in **/cert/cert.pem**:

```
# sudo -u foreman git config --global http.sslCAPath cert/cert.pem
```

For a complete list of advanced options, see the **git-config** manual page.

12.9. UNINSTALLING THE PLUG-IN

To avoid errors after removing the foreman_templates plugin:

1. Disable the plug-in using the Satellite installer:

```
# satellite-installer --no-enable-foreman-plugin-templates
```

2. Clean custom data of the plug-in. The command does not affect any templates that you created.

```
# foreman-rake templates:cleanup
```

3. Uninstall the plug-in:

```
# yum remove tfm-rubygem-foreman_templates
```

CHAPTER 13. SAMPLE SCENARIOS

13.1. SIMPLE SCENARIO

The simple scenario shows how to add a single host, register it, set up, and run a job on it.

13.1.1. Creating the Host

The following procedure provides an example on how to create a host in Red Hat Satellite.

To Create a Host:

1. Click **Hosts** > **Create Host**.
2. On the **Host** tab, enter the required details:
   a. In the **Name** field, enter the host name, for example *host1.example.com*.
   b. In the **Organization** field, enter the organization name, for example *MyOrg*.
   c. In the **Location** field, enter the location name, for example *MyLoc*.
3. Optionally, on the **Puppet Classes** tab, select the Puppet classes you want to include.
4. On the **Interfaces** tab, edit the primary interface:
   a. Click the **Edit** button in the **Actions** column.
   b. Select a type from the **Type** list, for example *Interface*.
   c. In the **MAC address** field, enter the MAC address of your host.
   d. In the **Device Identifier** field, specify the device identifier for this interface, for example *eth0*.
   e. In the **DNS name** field, specify the DNS name of your host. For the primary interface, this host name is used with the domain name to form the FQDN.
   f. Select a domain from the **Domain** list, for example *satellite.example.com*. This automatically updates the **IPv4 Subnet** and **IPv6 Subnet** lists with a selection of available subnets. Optionally, select the subnets.
   g. In the **IPv4 address** field, specify the IPv4 address of your host.
   h. Click **Ok**.
5. On the **Operating System** tab, enter the required details:
   a. Select an architecture from the **Architecture** list, for example *x86_64*.
   b. Select an operating system from the **Operating system** list, for example *RHEL Server 7.4*.
   c. Select a partition table from the **Partition table** list, for example *Kickstart default*.
   d. In the **Root password** field, enter the root password for your host.

6. Optionally, on the **Parameters** tab, select the parameters you want to supply to the Puppet master to override the default values.
7. Optionally, on the **Additional Information** tab, enter additional information about the host.
8. Click **Submit**.

13.1.2. Registering the Host

After you have created *host1.example.com*, you must register it so that it can receive updates. The following procedure assumes the host is running Red Hat Enterprise Linux 7.

To Register the Host:

1. In a terminal, connect to the host as the root user.
2. Ensure that a time synchronization tool is enabled and running on the host:

```
# systemctl start chronyd; systemctl enable chronyd
```

3. Install the consumer RPM from the Satellite Server or Capsule Server to which the host is to be registered. The consumer RPM updates the content source location of the host and allows the host to download content from the content source specified in Red Hat Satellite.

```
# rpm -Uvh http://satellite.example.com/pub/katello-ca-consumer-latest.noarch.rpm
```

4. Ensure that an activation key associated with the appropriate Content View and environment exists for the host. If not, see Managing Activation Keys in the *Content Management Guide* for more information.
5. Clear any old host data related to Red Hat Subscription Manager (RHSM):

```
# subscription-manager clean
```

6. Register the host using RHSM:

```
# subscription-manager register --org=MyOrg \
--activationkey=my_activation_key
```

Command output after registration:

```
# subscription-manager register --org=MyOrg --activationkey=my_activation_key
The system has been registered with id: 62edc0f8-855b-4184-b1b8-72a9dc793b96
```

7. Enable the **Red Hat Satellite Tools 6**repository:

```
# subscription-manager repos --enable=rhel-version-server-satellite-tools-6-rpms
```

8. Install the **katello-agent**:

```
# yum install katello-agent
```

9. Ensure the **goferd** service is running:

```
# systemctl start goferd
```

10. Install and configure the Puppet agent:
    a. Install the Puppet agent:

    ```
    # yum install puppet
    ```

    b. Configure the Puppet agent to start at boot:

    ```
    # systemctl enable puppet
    ```

11. Append the following server and environment settings to the **/etc/puppetlabs/puppet/puppet.conf** file. Set the **environment** parameter to the name of the Puppet environment to which the host belongs:

    ```
    environment = My_Example_Org_Library
    server = satellite.example.com
    ca_server = satellite.example.com
    ```

12. Run the Puppet agent on the host:

    ```
    # puppet agent -t
    ```

13. Sign the SSL certificate for the Puppet client in the Satellite Server web UI:
    a. Log in to the Satellite Server web UI.
    b. Select **Infrastructure** > **Capsules**.
    c. Select **Certificates** from the list to the right of the required Capsule.
    d. Click **Sign** to the right of the required host.
    e. Enter the **puppet agent** command again:

    ```
    # puppet agent -t
    ```

13.1.3. Running a Job on the Host

The following procedure shows how to run a job template on the previously created and registered host *host1.example.com*.

To Execute a Remote Job:

1. Navigate to **Hosts** > **All hosts** and select the target host, in our example *host1.example.com*.
2. From the **Select Action** list at the upper right of the screen select **Schedule Remote Job**.
3. On the **Job invocation** page, define the main job settings:
    a. Select a job category from the **Job category** list, for example *Commands*.
    b. Select a job template from the **Job template** list, for example *Run Command - SSH Default*.

c. In the **command** field, enter the command you want to run on the host. For example, *timedatectl set-timezone Europe/Prague* will set the time zone to Prague in Europe.

4. Click **Submit**.

APPENDIX A. TEMPLATE WRITING REFERENCE

Embedded Ruby (ERB) is a tool for generating text files based on templates that combine plain text with Ruby code. Red Hat Satellite uses ERB syntax in the following cases:

Provisioning templates

For more information, see Creating Provisioning Templates in the *Provisioning Guide*.

Remote execution job templates

For more information, see Chapter 9, *Running Jobs on Hosts*.

Report templates

For more information, see Chapter 6, *Using Report Templates to Monitor Hosts*.

Templates for partition tables

For more information, see Creating Partition Tables in the *Provisioning Guide*.

Smart Variables

For more information, see Configuring Smart Variables in the *Puppet Guide*.

Smart Class Parameters

For more information, see Configuring Smart Class Parameters in the *Puppet Guide*.

This section provides an overview of Satellite-specific macros and variables that can be used in ERB templates along with some usage examples. Note that the default templates provided by Red Hat Satellite (**Hosts** > **Provisioning templates**, **Hosts** > **Job templates**, **Monitor** > **Report Templates**) also provide a good source of ERB syntax examples.

When provisioning a host or running a remote job, the code in the ERB is executed and the variables are replaced with the host specific values. This process is referred to as **rendering**. The Satellite Server has the safemode rendering option enabled by default, which prevents any harmful code being executed from templates.

A.1. WRITING ERB TEMPLATES

The following tags are the most important and commonly used in ERB templates:

<% %>

All Ruby code is enclosed within **<% %>** in an ERB template. The code is executed when the template is rendered. It can contain Ruby control flow structures as well as Satellite-specific macros and variables. For example:

```
<% if @host.operatingsystem.family == "Redhat" && @host.operatingsystem.major.to_i > 6 %>
systemctl <%= input("action") %> <%= input("service") %>
<% else %>
service <%= input("service") %> <%= input("action") %>
<% end -%>
```

Note that there is no output when the template is rendered.

<%= %>

This provides the same functionality as **<% %>** but when the template is executed, the code output is inserted into the template. This is useful for variable substitution, for example:

Example input:

```
echo <%= @host.name %>
```

Example rendering:

```
host.example.com
```

Example input:

```
<% server_name = @host.fqdn %>
<%= server_name %>
```

Example rendering:

```
host.example.com
```

Note that if you enter an incorrect variable, no output is returned. However, if you try to call a method on an incorrect variable, the following error message returns:

Example input:

```
<%= @example_incorrect_variable.fqdn -%>
```

Example rendering:

```
undefined method `fqdn' for nil:NilClass
```

<% -%>, <%= -%>

By default, a newline character is inserted after a Ruby block if it is closed at the end of a line:

Example input:

```
<%= "line1" %>
<%= "line2" %>
```

Example rendering:

```
line1
line2
```

To change the default behavior, modify the enclosing mark with **-%>**:

Example input:

```
<%= "line1" -%>
<%= "line2" %>
```

Example rendering:

```
line1line2
```

This is used to reduce the number of lines, where Ruby syntax permits, in rendered templates. White spaces in ERB tags are ignored.

An example of how this would be used in a report template to remove unnecessary newlines between a FQDN and IP address:

Example input:

```
<%= @host.fqdn -%>
<%= @host.ip -%>
```

Example rendering:

```
host.example.com10.10.181.216
```

<%# %>

Encloses a comment that is ignored during template rendering:

Example input:

```
<%# A comment %>
```

This generates no output.

Indentation in ERB templates

Because of the varying lengths of the ERB tags, indenting the ERB syntax might seem messy. ERB syntax ignore white space. One method of handling the indentation is to declare the ERB tag at the beginning of each new line and then use white space within the ERB tag to outline the relationships within the syntax, for example:

```
<%- load_hosts.each do |host| -%>
<%-   if host.build? %>
<%=     host.name %> build is in progress
<%-   end %>
<%- end %>
```

A.2. TROUBLESHOOTING ERB TEMPLATES

The Satellite web UI provides two ways to verify the template rendering for a specific host:

- **Directly in the template editor** – when editing a template (under **Hosts** > **Partition tables**, **Hosts** > **Provisioning templates**, or **Hosts** > **Job templates**), on the **Template** tab click **Preview** and select a host from the list. The template then renders in the text field using the selected host's parameters. Preview failures can help to identify issues in your template.
- **At the host's details page** – select a host at **Hosts** > **All hosts** and click the **Templates** tab to list templates associated with the host. Select **Review** from the list next to the selected template to view it's rendered version.

A.3. GENERIC SATELLITE-SPECIFIC MACROS

This section lists Satellite-specific macros for ERB templates.

You can use the macros listed in the following table across all kinds of templates.

Table A.1. Generic Macros

Name	Description
indent(n)	Indents the block of code by n spaces, useful when using a snippet template that is not indented.
foreman_url(kind)	Returns the full URL to host-rendered templates of the given kind. For example, templates of the "provision" type usually reside at http://HOST/unattended/provision.
snippet(name)	Renders the specified snippet template. Useful for nesting provisioning templates.
snippets(file)	Renders the specified snippet found in the Foreman database, attempts to load it from the **unattended/snippets/** directory if it is not found in the database.
snippet_if_exists(name)	Renders the specified snippet, skips if no snippet with the specified name is found.

A.4. TEMPLATES MACROS

If you want to write custom templates, you can use some of the following macros.

Depending on the template type, some of the following macros have different requirements.

For more information about the available macros for report templates, in the Satellite web UI, navigate to **Monitor** > **Report Templates**, and click **Create Template**. In the Create Template window, click the **Help** tab.

For more information about the available macros for job templates, in the Satellite web UI, navigate to **Hosts** > **Job Templates**, and click the **New Job Template**. In the New Job Template window, click the **Help** tab.

input

Using the **input** macro, you can customize the input data that the template can work with. You can define the input name, type, and the options that are available for users. For report templates, you can only use user inputs. When you define a new input and save the template, you can then reference the input in the ERB syntax of the template body.

```
<%= input('cpus') %>
```

This loads the value from user input **cpus**.

load_hosts

Using the **load_hosts** macro, you can generate a complete list of hosts.

```
<%- load_hosts().each_record do |host| -%>
<%=     host.name %>
```

Use the **load_hosts** macro with the **each_record** macro to load records in batches of 1000 to reduce memory consumption.

If you want to filter the list of hosts for the report, you can add the option **search: input('*Example_Host*')**:

```
<% load_hosts(search: input('Example_Host')).each_record do |host| -%>
<%=  host.name %>
<% end -%>
```

In this example, you first create an input that you then use to refine the search criteria that the **load_hosts** macro retrieves.

report_row

Using the **report_row** macro, you can create a formatted report for ease of analysis. The **report_row** macro requires the **report_render** macro to generate the output.

Example input:

```
<%- load_hosts(search: input('Example_Host')).each_record do |host| -%>
<%-   report_row(
        'Server FQDN': host.name
      ) -%>
<%- end -%>
<%= report_render -%>
```

Example rendering:

```
Server FQDN
host1.example.com
host2.example.com
host3.example.com
host4.example.com
host5.example.com
host6.example.com
```

You can add extra columns to the report by adding another header. The following example adds IP addresses to the report:

Example input:

```
<%- load_hosts(search: input('host')).each_record do |host| -%>
<%-   report_row(
      'Server FQDN': host.name,
          'IP': host.ip
      ) -%>
<%- end -%>
<%= report_render -%>
```

Example rendering:

```
Server FQDN,IP
host1.example.com,10.8.30.228
host2.example.com,10.8.30.227
host3.example.com,10.8.30.226
host4.example.com,10.8.30.225
host5.example.com,10.8.30.224
host6.example.com,10.8.30.223
```

report_render

This macro is available only for report templates.
Using the **report_render** macro, you create the output for the report. During the template rendering process, you can select the format that you want for the report. YAML and CSV formats are supported.

```
<%= report_render format: :yaml -%>
```

render_template()

This macro is available only for job templates.
Using this macro, you can render a specific template. You can also enable and define arguments that you want to pass to the template.

A.5. HOST-SPECIFIC VARIABLES

The following variables enable using host data within templates. Note that job templates accept only **@host** variables.

Table A.2. Host Specific Variables and Macros

Name	Description
@host.architecture	The architecture of the host.
@host.bond_interfaces	Returns an array of all bonded interfaces. See Section A.8, "Parsing Arrays".
@host.capabilities	The method of system provisioning, can be either build (for example kickstart) or image.
@host.certname	The SSL certificate name of the host.
@host.diskLayout	The disk layout of the host. Can be inherited from the operating system.
@host.domain	The domain of the host.
@host.environment	The Puppet environment of the host.

Name	Description
@host.facts	Returns a Ruby hash of facts from Facter. For example to access the 'ipaddress' fact from the output, specify @host.facts['ipaddress'].
@host.grub_pass	Returns the host's GRUB password.
@host.hostgroup	The host group of the host.
host_enc['parameters']	Returns a Ruby hash containing information on host parameters. For example, use host_enc['parameters']['lifecycle_environment'] to get the life cycle environment of a host.
@host.image_build?	Returns **true** if the host is provisioned using an image.
@host.interfaces	Contains an array of all available host interfaces including the primary interface. See Section A.8, "Parsing Arrays".
@host.interfaces_with_identifier('IDs')	Returns array of interfaces with given identifier. You can pass an array of multiple identifiers as an input, for example @host.interfaces_with_identifier(['eth0', 'eth1']). See Section A.8, "Parsing Arrays".
@host.ip	The IP address of the host.
@host.location	The location of the host.
@host.mac	The MAC address of the host.
@host.managed_interfaces	Returns an array of managed interfaces (excluding BMC and bonded interfaces). See Section A.8, "Parsing Arrays".
@host.medium	The assigned operating system installation medium.
@host.name	The full name of the host.
@host.operatingsystem.family	The operating system family.
@host.operatingsystem.major	The major version number of the assigned operating system.
@host.operatingsystem.minor	The minor version number of the assigned operating system.

Name	Description
@host.operatingsystem.name	The assigned operating system name.
@host.operatingsystem.boot_files_uri(medium_provider)	Full path to the kernel and initrd, returns an array.
@host.os.medium_uri(@host)	The URI used for provisioning (path configured in installation media).
host_param('parameter_name')	Returns the value of the specified host parameter.
host_param_false?('parameter_name')	Returns **false** if the specified host parameter evaluates to false.
host_param_true?('parameter_name')	Returns **true** if the specified host parameter evaluates to true.
@host.primary_interface	Returns the primary interface of the host.
@host.provider	The compute resource provider.
@host.provision_interface	Returns the provisioning interface of the host. Returns an interface object.
@host.ptable	The partition table name.
@host.puppetmaster	The Puppet master the host should use.
@host.pxe_build?	Returns **true** if the host is provisioned using the network or PXE.
@host.shortname	The short name of the host.
@host.sp_ip	The IP address of the BMC interface.
@host.sp_mac	The MAC address of the BMC interface.
@host.sp_name	The name of the BMC interface.
@host.sp_subnet	The subnet of the BMC network.
@host.subnet.dhcp	Returns **true** if a DHCP proxy is configured for this host.
@host.subnet.dns_primary	The primary DNS server of the host.

Name	Description
@host.subnet.dns_secondary	The secondary DNS server of the host.
@host.subnet.gateway	The gateway of the host.
@host.subnet.mask	The subnet mask of the host.
@host.url_for_boot(:initrd)	Full path to the initrd image associated with this host. Not recommended, as it does not interpolate variables.
@host.url_for_boot(:kernel)	Full path to the kernel associated with this host. Not recommended, as it does not interpolate variables, prefer boot_files_uri.
@provisioning_type	Equals to 'host' or 'hostgroup' depending on type of provisioning.
@static	Returns **true** if the network configuration is static.
@template_name	Name of the template being rendered.
grub_pass	Returns the GRUB password wrapped in md5pass argument, that is **--md5pass=#{@host.grub_pass}**.
ks_console	Returns a string assembled using the port and the baud rate of the host which can be added to a kernel line. For example **console=ttyS1,9600**.
root_pass	Returns the root password configured for the system.

The majority of common Ruby methods can be applied on host-specific variables. For example, to extract the last segment of the host's IP address, you can use:

```
<% @host.ip.split('.').last %>
```

A.6. KICKSTART-SPECIFIC VARIABLES

The following variables are designed to be used within kickstart provisioning templates.

Table A.3. Kickstart Specific Variables

Name	Description
@arch	The host architecture name, same as @host.architecture.name.

Name	Description
@dynamic	Returns **true** if the partition table being used is a %pre script (has the #Dynamic option as the first line of the table).
@epel	A command which will automatically install the correct version of the epel-release rpm. Use in a %post script.
@mediapath	The full kickstart line to provide the URL command.
@osver	The operating system major version number, same as @host.operatingsystem.major.

A.7. CONDITIONAL STATEMENTS

In your templates, you might perform different actions depending on which value exists. To achieve this, you can use conditional statements in your ERB syntax.

In the following example, the ERB syntax searches for a specific host name and returns an output depending on the value it finds:

Example input:

```
<% load_hosts().each_record do |host| -%>
<% if @host.name == "host1.example.com" -%>
<%      result="positive" -%>
<%  else -%>
<%      result="negative" -%>
<%  end -%>
<%= result -%>
```

Example rendering:

```
host1.example.com
positive
```

A.8. PARSING ARRAYS

While writing or modifying templates, you might encounter variables that return arrays. For example, host variables related to network interfaces, such as **@host.interfaces** or **@host.bond_interfaces**, return interface data grouped in an array. To extract a parameter value of a specific interface, use Ruby methods to parse the array.

Finding the Correct Method to Parse an Array

The following procedure is an example that you can use to find the relevant methods to parse arrays in your template. In this example, a report template is used, but the steps are applicable to other templates.

1. To retrieve the NIC of a content host, in this example, using the **@host.interfaces** variable returns class values that you can then use to find methods to parse the array.

   Example input:

   ```
   <%= @host.interfaces -%>
   ```

   Example rendering:

   ```
   <Nic::Base::ActiveRecord_Associations_CollectionProxy:0x00007f734036fbe0>
   ```

2. In the Create Template window, click the **Help** tab and search for the **ActiveRecord_Associations_CollectionProxy** and **Nic::Base** classes.

3. For **ActiveRecord_Associations_CollectionProxy**, in the **Allowed methods or members** column, you can view the following methods to parse the array:

   ```
   [] each find_in_batches first map size to_a
   ```

4. For **Nic::Base**, in the **Allowed methods or members** column, you can view the following method to parse the array:

   ```
   alias? attached_devices attached_devices_identifiers attached_to bond_options
   children_mac_addresses domain fqdn identifier inheriting_mac ip ip6 link mac managed?
   mode mtu nic_delay physical? primary provision shortname subnet subnet6 tag virtual?
   vlanid
   ```

5. To iterate through an interface array, add the relevant methods to the ERB syntax:

   Example input:

   ```
   <% load_hosts().each_record do |host| -%>
   <%   host.interfaces.each do |iface| -%>
     iface.alias?: <%= iface.alias? %>
     iface.attached_to: <%= iface.attached_to %>
     iface.bond_options: <%= iface.bond_options %>
     iface.children_mac_addresses: <%= iface.children_mac_addresses %>
     iface.domain: <%= iface.domain %>
     iface.fqdn: <%= iface.fqdn %>
     iface.identifier: <%= iface.identifier %>
     iface.inheriting_mac: <%= iface.inheriting_mac %>
     iface.ip: <%= iface.ip %>
     iface.ip6: <%= iface.ip6 %>
     iface.link: <%= iface.link %>
     iface.mac: <%= iface.mac %>
     iface.managed?: <%= iface.managed? %>
     iface.mode: <%= iface.mode %>
     iface.mtu: <%= iface.mtu %>
     iface.physical?: <%= iface.physical? %>
     iface.primary: <%= iface.primary %>
     iface.provision: <%= iface.provision %>
     iface.shortname: <%= iface.shortname %>
     iface.subnet: <%= iface.subnet %>
     iface.subnet6: <%= iface.subnet6 %>
   ```

```
  iface.tag: <%= iface.tag %>
  iface.virtual?: <%= iface.virtual? %>
  iface.vlanid: <%= iface.vlanid %>
<%- end -%>
```

Example rendering:

```
host1.example.com
  iface.alias?: false
  iface.attached_to:
  iface.bond_options:
  iface.children_mac_addresses: []
  iface.domain:
  iface.fqdn: host1.example.com
  iface.identifier: ens192
  iface.inheriting_mac: 00:50:56:8d:4c:cf
  iface.ip: 10.10.181.13
  iface.ip6:
  iface.link: true
  iface.mac: 00:50:56:8d:4c:cf
  iface.managed?: true
  iface.mode: balance-rr
  iface.mtu:
  iface.physical?: true
  iface.primary: true
  iface.provision: true
  iface.shortname: host1.example.com
  iface.subnet:
  iface.subnet6:
  iface.tag:
  iface.virtual?: false
  iface.vlanid:
```

A.9. EXAMPLE TEMPLATE SNIPPETS

Checking if a Host Has Puppet and Puppetlabs Enabled

The following example checks if the host has the Puppet and Puppetlabs repositories enabled:

```
<%
pm_set = @host.puppetmaster.empty? ? false : true
puppet_enabled = pm_set || host_param_true?('force-puppet')
puppetlabs_enabled = host_param_true?('enable-puppetlabs-repo')
%>
```

Capturing Major and Minor Versions of a Host's Operating System

The following example shows how to capture the minor and major version of the host's operating system, which can be used for package related decisions:

```
<%
os_major = @host.operatingsystem.major.to_i
os_minor = @host.operatingsystem.minor.to_i
%>
```

```
<% if ((os_minor < 2) && (os_major < 14)) -%>
...
<% end -%>
```

Importing Snippets to a Template

The following example imports the **subscription_manager_registration** snippet to the template and indents it by four spaces:

```
<%= indent 4 do
snippet 'subscription_manager_registration'
end %>
```

Conditionally Importing a Kickstart Snippet

The following example imports the **kickstart_networking_setup** snippet if the host's subnet has the DHCP boot mode enabled:

```
<% subnet = @host.subnet %>
<% if subnet.respond_to?(:dhcp_boot_mode?) -%>
<%= snippet 'kickstart_networking_setup' %>
<% end -%>
```

Parsing Values from Host Custom Facts

You can use the **host.facts** variable to parse values from a host's facts and custom facts.

In this example **luks_stat** is a custom fact that you can parse in the same manner as **dmi::system::serial_number**, which is a host fact:

```
'Serial': host.facts['dmi::system::serial_number'],
'Encrypted': host.facts['luks_stat'],
```

In this example, you can customize the Applicable Errata report template to parse for custom information about the kernel version of each host:

```
<%-    report_row(
        'Host': host.name,
        'Operating System': host.operatingsystem,
        'Kernel': host.facts['uname::release'],
        'Environment': host.lifecycle_environment,
        'Erratum': erratum.errata_id,
        'Type': erratum.errata_type,
        'Published': erratum.issued,
        'Applicable since': erratum.created_at,
        'Severity': erratum.severity,
        'Packages': erratum.package_names,
        'CVEs': erratum.cves,
        'Reboot suggested': erratum.reboot_suggested,
      ) -%>
```

APPENDIX B. HOST MANAGEMENT WITHOUT GOFERD

From **Satellite 6.2.11** onward, errata and package management via remote execution is available using a **yum** plugin. This allows the goferd service daemon to be disabled, thereby reducing memory and CPU load on content hosts.

The **yum** plugin is included with the **katello-host-tools**. This shipped with the Satellite 6.2.11 client update.

B.1. PREREQUISITES

The following must be made on all content hosts to allow host management by remote execution.

- Ensure **katello-agent** is installed on the content host by following Section 3.4, "Installing the Katello Agent".
- Stop the goferd service:

```
# systemctl stop goferd.service
```

- Disable the goferd service:

```
# systemctl disable goferd.service
```

- Distribute the SSH keys to the content hosts by following Section 9.1, "Establishing a Secure Connection for Remote Commands".

B.2. CONFIGURING HOST MANAGEMENT WITHOUT GOFERD AS THE SYSTEM DEFAULT

These steps configure host management to use remote execution as the system default for future package deployments.

To Configure Host Management Without Goferd as the System Default:

1. Log in to the Satellite web UI.
2. Navigate to **Administer** > **Settings**.
3. Select the **Content** tab.
4. Set the **Use remote execution by default** parameter to **Yes**.

The Satellite server now uses host management by remote execution instead of goferd.

B.3. LIMITATIONS WITH HAMMER

The following applies if you are using the **hammer** command to push errata. The **hammer** command is dependent on goferd to manage errata on content hosts. As a workaround, use Satellite's remote execution feature to apply errata.

To Use Hammer Remote Execution Commands:

For example, perform a **yum -y update** on *host123.example.org*:

```
# hammer job-invocation create \
--job-template "Run Command - SSH Default" \
--inputs command="yum -y update" \
--search-query "name ~ host123"
Job invocation 24 created
[.....................................] [100%]
1 task(s), 1 success, 0 fail
```