Red Hat Satellite 6.2

Hammer CLI Guide

Using Hammer, the Satellite's CLI tool

Last Updated: 2018-05-01

Red Hat Satellite 6.2 Hammer CLI Guide

Using Hammer, the Satellite's CLI tool

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

Legal Notice

Abstract

This document describes how to use the Hammer CLI tool to configure and manage Red Hat Satellite.

Table of Contents

CHAPTER 1. INTRODUCTION TO HAMMER

Hammer is a powerful command-line tool provided with Red Hat Satellite 6. You can use Hammer to configure and manage a Red Hat Satellite Server either through CLI commands or automation in shell scripts. Hammer also provides an interactive shell.

Hammer compared to Satellite web UI

Compared to navigating the web UI, using Hammer can result in much faster interaction with the Satellite Server, as common shell features such as environment variables and aliases are at your disposal. You can also incorporate Hammer commands into reusable scripts for automating tasks of various complexity. Output from Hammer commands can be redirected to other tools, which allows for integration with your existing environment. You can issue Hammer commands directly on the base operating system running Red Hat Satellite.

Access to Satellite Server's base operating system is required to issue Hammer commands, which can limit the number of potential users compared to the web UI. Although the parity between Hammer and the web UI is almost complete, the web UI has development priority and can be ahead especially for newly introduced features.

Hammer compared to Satellite API

For many tasks, both Hammer and Satellite API are equally applicable. Hammer can be used as a human friendly interface to Satellite API, for example to test responses to API calls before applying them in a script (use the **`-d`** option to inspect API calls issued by Hammer, for example **`hammer -d organization list`**). Changes in the API are automatically reflected in Hammer, while scripts using the API directly have to be updated manually.

In the background, each Hammer command first establishes a binding to the API, then sends a request. This can have performance implications when executing a large number of Hammer commands in sequence. In contrast, a script communicating directly with the API establishes the binding only once. See the Red Hat Satellite API Guide for more information.

1.1. GETTING HELP

View the full list of **`hammer`** options and subcommands by executing:

```
$ hammer --help
```

Use **`--help`** to inspect any subcommand, for example:

```
$ hammer organization --help
```

You can search the help output using **`grep`**, or redirect it to a text viewer, for example:

```
$ hammer | less
```

1.2. AUTHENTICATION

By default, **`hammer`** prompts for your Satellite credentials each time you issue a command. You can specify your credentials when executing a command as follows:

```
$ hammer -u <username> -p <password> <subcommands>
```

As an alternative, follow these steps to use saved credentials:

1. Create the file **~/.hammer/cli_config.yml** and add the following contents to the file:

```
:foreman:
 :host: 'https://satellite.example.com/'
 :username: 'username'
 :password: 'password'
```

   Replace the example values with your own details. Do not use tabs in the file, always use indentation by spaces.

2. To protect your password, make sure the file is readable only by the current user:

```
$ chmod 600 ~/.hammer/cli_config.yml
```

3. Save and close the file. Now when you start hammer it will use the credentials in the **~/.hammer/cli_config.yml** file.

IMPORTANT

Use only spaces for indentation in Hammer configuration files. Do not use tabs for indentation in Hammer configuration files.

NOTE

Examples in this guide assume saved credentials.

1.3. STANDALONE USE OF HAMMER

It is possible to install **hammer** individually on a server where there is no Satellite installed, and use it to connect the server to a remote Satellite.

The **rhel-X-server-satellite-6.X-rpms** repository is required to install the package. Users desiring to use a workstation to connect should install the repository manually, see the Red Hat Satellite Installation Guide.

1. Install **hammer**.

```
# yum install tfm-rubygem-hammer_cli_katello
```

2. Edit **/etc/hammer/cli.modules.d/foreman.yml** to point to the desired Satellite.

1.4. SETTING A DEFAULT ORGANIZATION

Many **hammer** commands are organization specific. You can set a default organization, as well as location, for hammer commands so that you do not have to specify it every time using the **--organization-id** parameter. To do so, issue:

```
$ hammer defaults add --param-name organization_id --param-value <org_ID>
```

Find *<org_ID>* in the output of the **hammer organization list** command. Similarly, you can set the default location as follows:

```
$ hammer defaults add --param-name location_id --param-value <loc_ID>
```

To view the currently specified default settings, issue the following command:

```
$ hammer defaults list
```

Specifying a default organization is useful when you mostly manage a single organization, as it makes your commands shorter. However, when switching to a different organization, you still have to use a command-line option to specify it. Examples in this guide do not assume a saved default organization, instead they use the shell variable approach described in Note.

1.5. CONFIGURING HAMMER

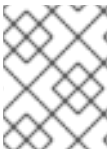
The default location for global **hammer** configuration is:

- **/etc/hammer/cli_config.yml** for general **hammer** settings.
- **/etc/hammer/cli.modules.d/** for CLI module configuration files.

You can set user specific directives for **hammer** (in ~/**.hammer/cli_config.yml**) as well as for CLI modules (in respective **.yml** files under ~/**.hammer/cli.modules.d/**).

To see the order in which configuration files are loaded, as well as versions of loaded modules, issue:

```
$ hammer -d --version
```



NOTE

Loading configuration for many CLI modules can slow down the execution of **hammer** commands. In such a case, consider disabling CLI modules that are not regularly used.

Apart from saving credentials as described in Section 1.2, “Authentication”, you can set several other options in the ~/**.hammer/** configuration directory. For example, you can change the default log level and set log rotation with the following directives in ~/**.hammer/cli_config.yml**. Note that these directives affect only the current user and are not applied globally.

```
:log_level: 'warning'
:log_size: 5 #in MB
```

Similarly, you can set the number of lines displayed at once in the **hammer** output (equivalent of the **--per-page** option):

```
:per-page: 30
```

1.6. CONFIGURING HAMMER LOGGING

You can set **hammer** to log debugging information for various Satellite components.

You can set debug or normal configuration options for all Satellite components.

NOTE

After changing hammer's logging behavior, you must restart Satellite services.

```
# katello-service restart
```

- To set debug level for all components, use the following command:

```
# hammer admin logging --all --level-debug
# katello-service restart
```

- To set production level logging, use the following command:

```
# hammer admin logging --all --level-production
# katello-service restart
```

- To list the currently recognized components, that you can set logging for:

```
# hammer admin logging --list
```

- To list all the available options of this tool:

```
# hammer admin logging --help

Usage:
    hammer admin logging [OPTIONS]
```

1.7. INVOKING THE HAMMER SHELL

You can issue **hammer** commands through the interactive shell. To invoke the shell, issue the following command:

```
$ hammer shell
```

In the shell, you can enter subcommands directly without typing "hammer", which can be useful for testing commands before using them in a script. To exit the shell, type **exit** or press [Ctrl + D].

1.8. GENERATING FORMATTED OUTPUT

You can modify the default formatting of the output of **hammer** commands to simplify the processing of this output by other command line tools and applications. For example, to list organizations in a CSV format with a custom separator (in this case a semicolon), issue the following command:

```
$ hammer --csv --csv-separator ";" organization list
```

Output in CSV format is useful for example when you need to parse IDs and use them in a **for** loop (see Example 2.6, "Synchronizing All Repositories in ACME Organization" or Example 2.8, "Assigning a Synchronization Plan to Multiple Products").

Several other formatting options are available with the **--output** option:

```
$ hammer --output <output_format> organization list
```

Replace *<output_format>* with one of:

- **table** — generates output in the form of a human readable table (default).
- **base** — generates output in the form of key-value pairs.
- **yaml** — generates output in the YAML format.
- **csv** — generates output in the Comma Separated Values format. To define a custom separator, use the **--csv** and **--csv-separator** options instead.
- **json** — generates output in the JavaScript Object Notation format.
- **silent** — suppresses the output.

1.9. TROUBLESHOOTING WITH HAMMER

You can use the **hammer ping** command to check the status of core Satellite services. Together with the **katello-service status** command, this can help you to diagnose and troubleshoot Satellite issues. If all services are running as expected, the output looks as follows:

```
$ hammer ping
candlepin:
    Status:          ok
    Server Response: Duration: 22ms
candlepin_auth:
    Status:          ok
    Server Response: Duration: 17ms
pulp:
    Status:          ok
    Server Response: Duration: 41ms
pulp_auth:
    Status:          ok
    Server Response: Duration: 23ms
foreman_tasks:
    Status:          ok
    Server Response: Duration: 33ms
```

CHAPTER 2. MANAGING ORGANIZATIONS, LOCATIONS, AND REPOSITORIES

You can use **hammer** to create, edit, and manage organizations, locations, and repositories. For web UI equivalents of the following procedures see Configuring Organizations, Locations and Life Cycle Environments in the *Red Hat Satellite Server Administration Guide*.

2.1. ORGANIZATIONS

Organization in Red Hat Satellite is an isolated collection of systems, content, and other functionality within a Satellite deployment. This section shows how to create and modify organizations using **hammer**.

2.1.1. Creating an Organization

Use the following command to create an organization:

```
$ hammer organization create \
--name "<org_name>" \
--label "<org_label>" \
--description "<org_description>"
```

Where:

- *<org_name>* is the name of the organization. This parameter is required.
- *<org_label>* is the organization label used in command-line applications such as **subscription-manager**. Labels cannot contain white space and you cannot change them later. If not specified, label is generated automatically from the organization name (white space is replaced with underscores).
- *<org_description>* is a short description of the organization. This parameter is not required, but it can help you to manage a large number of organizations.

You can fully configure an organization while creating it (issue **hammer organization create --help** to see the options). Also, you can modify an existing organization using the **hammer organization update** command.

Example 2.1. Creating and Updating ACME Organization

The following example shows how to create an organization named ACME:

```
$ ORG="ACME"
$ hammer organization create \
--name $ORG \
--description "Example organization"
```

This command assigns a compute resource to the organization:

```
$ hammer organization update \
--name $ORG \
--compute-resource-ids 1
```

NOTE

Many tasks you can perform in the Satellite Server are specific to an organization. Hammer commands provide three ways to identify an organization: by using the **`organization`**, **`organization-label`**, or **`organization-id`** option. To find the organization ID, use the following command:

```
$ hammer organization list
```

If your organization name is long, consider storing it in a shell variable. You can use this variable in Hammer commands. For example:

```
$ ORG = "Red Hat Enterprise Linux Developer Team"
$ hammer product list --organization $ORG
```

This approach is used in examples in this guide.

If you mostly manage a single organization, store its ID as a default parameter, for example:

```
$ hammer defaults add --param-name organization_id --param-
value 1
```

With the above setting, organization specific commands will assume **`--organization-id 1`** is specified, so you no longer have to type it.

2.1.2. Uploading a Manifest

A Subscription Manifest transfers subscriptions from the Red Hat Customer Portal to Satellite Server. First, generate the manifest on the Red Hat Customer portal as described in the Red Hat Satellite Content Management Guide. Then upload the manifest to the organization as follows:

```
$ hammer subscription upload \
--organization-label <org_label> \
--file <path_to_manifest>
```

Example 2.2. Uploading a Manifest to ACME Organization

The following example shows how to upload a Subscription Manifest file to the ACME organization (assuming the organization name is stored in a shell variable):

```
$ hammer subscription upload --organization $ORG --file
/tmp/manifest.zip
```

To view the subscriptions imported with the manifest, issue:

```
$ hammer subscription list --organization $ORG
```

2.2. LOCATIONS

Location in Red Hat Satellite is collection of default settings that represent a physical place. This section shows how to create locations using **hammer**.

2.2.1. Creating a Location

Use the following command to create a location:

```
$ hammer location create --name <location_name>
```

Example 2.3. Creating Multiple Locations Using a Script

The following Bash script creates three locations (london, munich, boston), and assigns them to the ACME organization.

```
ORG="ACME"
LOCATIONS="london munich boston"

for LOC in ${LOCATIONS}
do
  hammer location create --name "${LOC}"
  hammer location add-organization --name "${LOC}" --organization
"${ORG}"
done
```

Run **hammer location --help** to view all possible location related operations.

2.3. REPOSITORIES

Repository provides storage for a collection of content. This section shows how to enable and synchronize repositories using **hammer**.

2.3.1. Enabling a Repository

Before enabling a Red Hat repository, you need to know its name, the name of the product it provides, the base architecture, and the release version. Use the following command to enable a repository:

```
$ hammer repository-set enable \
--organization-label <org_label> \
--product "<product_name>" \
--basearch "<base_architecture>" \
--releasever "<release_version>" \
--name "<repository_name>"
```

Example 2.4. Enabling a Red Hat Enterprise Linux Repository

The following command enables the Red Hat Enterprise Linux 7 Server repository for the organization:

```
$ hammer repository-set enable \
--organization $ORG \
--product "Red Hat Enterprise Linux Server" \
--basearch "x86_64" \
```

```
--releasever "7Server" \
--name "Red Hat Enterprise Linux 7 Server (RPMs)"
```

Run **`hammer repository-set --help`** to view all possible repository related operations. Also see **`hammer repository --help`**.

2.3.2. Synchronizing a Repository

By synchronizing a repository you pull its content from Red Hat Customer Portal to the Satellite Server. To synchronize a repository you need to specify its name and a product name:

```
$ hammer repository synchronize \
--product "<product_name>" \
--name "<repo_name>" \
--organization-label <org_label> \
--async
```

Note that if you have created Content Views, multiple repositories with the same name can exist within a single organization. In such a case, use the **`--id`** option to identify the repository you want to synchronize (issue **`hammer repository list`** to find repository IDs).

Example 2.5. Synchronizing a Red Hat Enterprise Linux Repository

The following command performs a single synchronization of the Red Hat Enterprise Linux 7 Server repository in the organization:

```
$ hammer repository synchronize \
--product "Red Hat Enterprise Linux Server" \
--name "Red Hat Enterprise Linux 7 Server (RPMs)" \
--organization $ORG \
--async
```

The task ID is displayed after executing the above command:

```
Repository is being synchronized in task 640bb71f-0ce5-40a3-a675-
425a4acacceb
```

To view the progress of the task, issue:

```
$ hammer task progress --id 640bb71f-0ce5-40a3-a675-425a4acacceb
```

After finishing the first synchronization, the repository is added to the list of repositories mirrored on Satellite Server. Execute the following command to see the list:

```
$ hammer repository list --organization $ORG
```

You can also synchronize all repositories within a product as follows:

```
$ hammer product synchronize \
--organization-label <org_label> \
```

```
--name "<product_name>" \
--async
```

With the **--async** option, the repository synchronization runs in the background, which for example allows you to enable and synchronize several repositories in parallel.

Example 2.6. Synchronizing All Repositories in ACME Organization

The following Bash script synchronizes all repositories within the ACME organization.

```
ORG="ACME"

for i in $(hammer --csv repository list --organization $ORG | grep -vi
'^ID' | awk -F, {'print $1'})
do
  hammer repository synchronize --id ${i} --organization $ORG --async
done
```

2.3.3. Creating a Synchronization Plan

Product in Red Hat Satellite is a collection of repositories that acts as the smallest unit of the synchronization process. You can create a synchronization plan to automatically update repositories of a selected product in a given time interval.

To define a synchronization plan, issue the following command:

```
$ hammer sync-plan create \
--name "<sync_plan_name>" \
--enabled=true \
--interval <repetition_interval> \
--organization-label <org_label> \
--sync-date "<initial_sync>"
```

Replace *<repetition_interval>* with **hourly**, **daily**, or **weekly**. Replace *<initial_sync>* with the date and time of the initial synchronization in the form of "YYYY-MM-DD HH:MM:SS".

Example 2.7. Creating a Synchronization Plan

The following command creates a daily synchronization schedule for the ACME organization, that runs at 3 a.m., starting from 15 January 2016:

```
$ hammer sync-plan create \
--name "daily sync at 3 a.m." \
--enabled=true \
--interval daily \
--organization $ORG \
--sync-date "2016-01-15 03:00:00"
```

To associate the synchronization plan with a product, issue the following command:

```
$ hammer product set-sync-plan \
```

```
--organization-label <org_label> \
--name "<product_name>" \
--sync-plan "<sync_plan_name>"
```

Example 2.8. Assigning a Synchronization Plan to Multiple Products

The following Bash script selects the products in the ACME organization that have been synchronized at least once and contain at least one repository and assigns them a synchronization plan named "daily sync at 3 a.m.".

```
ORG="ACME"
SYNC_PLAN="daily sync at 3 a.m."

for i in $(hammer --csv product list --organization $ORG --per-page 999
| grep -vi 'ID' | grep -vi not_synced | awk -F, {'{ if ($5!=0) print
$1}'})
do
  hammer product set-sync-plan --sync-plan $SYNC_PLAN --organization
$ORG --id $i
done
```

After executing the script, issue the following command to see which products have been assigned the synchronization plan:

```
$ hammer product list --organization $ORG --sync-plan "daily sync at 3
a.m."
```

To view synchronization plans available for a selected organization, issue the following command:

```
$ hammer sync-plan list --organization-label <org_label>
```

For more details on working with products and synchronization plans see the output of **`hammer sync-plan --help`** and **`hammer product --help`**.

2.3.4. Creating a Custom Repository

After enabling a Red Hat repository, the corresponding product is created automatically. To enable a repository with custom packages, you first need to manually create a product for this repository.

Use the following command to create a custom product:

```
$ hammer product create --name "<product_name>" --organization-label
<org_label>
```

The following command creates a new repository under the custom product:

```
$ hammer repository create \
--name "<repo_name>" \
--organization-label <org_label> \
--product "<product_name>" \
--content-type <cont_type> --publish-via-http true \
--url "<repo_url>"
```

Replace the example values with your own details, in particular:

- *<cont_type>* specifies the type of content in the repository, choose one of **`yum`**, **`puppet`**, or **`docker`**.
- *<repo_url>* specifies the URL on which the repository will be available, valid only if ` --publish-via-http` is enabled.

To upload packages to a custom repository, issue the following command:

```
$ hammer repository upload-content \
--product "<product_name>" \
 --organization-label <org_label> \
 --id "<repo_id>" \
--path <path_to_dir>
```

Replace *<path_to_dir>* with the path to the directory with content (RPM packages, Puppet modules, or Docker images) to be added to the custom repository.

CHAPTER 3. MANAGING CONTENT LIFE CYCLE

This section shows how to use **hammer** to create Content Views and to promote them through life cycle environments.

3.1. CREATING A LIFE CYCLE ENVIRONMENT

Life cycle environments represent stages of the content life cycle. This section shows how to view and create life cycle environments with **hammer**. By default, the Library environment is present for each organization. Use the following syntax to create a new life cycle environment:

```
$ hammer lifecycle-environment create \
--name <env_name> \
--description "<env_description>" \
--organization-label <org_label> \
--prior <prior_env_name>
```

Example 3.1. Creating a Life Cycle Environment

This example shows how to create a new environment based on Library for the ACME organization (assuming the organization name is stored in a shell variable):

```
$ hammer lifecycle-environment create \
--name Development \
--description "Initial testing" \
--organization $ORG \
--prior Library
```

You can create another life cycle environment based on **Development** using the **--prior** option.

To view existing life cycle environments, issue the following command:

```
$ hammer lifecycle-environment list --organization-label <org_label>
```

The output of the above command can look as follows:

```
---|-------------|------------
ID | NAME        | PRIOR
---|-------------|------------
2  | Library     |
5  | Development | Library
6  | Testing     | Development
---|-------------|------------
```

For more information on commands related to life cycle environments, see the output of **hammer lifecycle-environment --help**.

3.2. CREATING A CONTENT VIEW

Content Views are subsets of content from the Library created by intelligent filtering. You can publish and promote Content Views into life cycle environments that make content available for different uses (typically Dev, QA, and Production). To create a Content View, issue the following command:

```
$ hammer content-view create \
--name <cv_name> \
--repository-ids <repo_ID1>,<repo_ID2>,<repo_ID3> \
--description "<cv_description>" \
--organization-label <org_label>
```

The **--repository-ids** option adds the selected repositories to the Content View, use the **hammer repository list** command to find the IDs. It is also possible to omit this option to create an empty Content View that you can modify later using the **update** or **add-repository** subcommands.

Example 3.2. Creating a Content View

The following example creates a Content View under the ACME organization and assigns it three repositories:

```
$ hammer content-view create \
--name cv-rhel7-server \
--repository-ids 1,2,3 \
--description "Initial CV for RHEL 7" \
--organization $ORG
```

Example 3.3. Creating a Composite Content View

A Composite Content View is comprised of one or more Content Views. This example shows how to create a Composite Content View from two existing Content Views:

```
$ hammer content-view create \
--name ccv-rhel7-server-scl \
--description "CCV for RHEL7 and Software Collections" \
--organization $ORG \
--composite --component-ids 2,6
```

Find the IDs for the **--component-ids** option by executing **hammer content-view list**.

There are three content types you can add to the Content View: RPM packages, Puppet modules, and Docker images.

3.2.1. Adding Repositories to a Content View

Use the following command to add repositories to an existing Content View:

```
$ hammer content-view update \
--repository-ids <repo_ID1>,<repo_ID2>... \
--name <cv_name> \
--organization-label <org_label>
```

The above command is useful for populating an empty Content View with repositories. Note that it will overwrite any existing repositories, therefore to increase the number of repositories in a Content View, use:

```
$ hammer content-view add-repository \
--organization-label <org_label> \
--name <cv_name> \
--repository-id <repo_ID>
```

Similarly, you can use the **remove-repository** subcommand to remove a repository from the Content View. Use **hammer content-view info** to inspect repositories in a Content View.

Example 3.4. Filtering Packages for a Content View – Excluding a Package

Filters allow you to select a subset of packages from a repository (either by including or excluding) to create customized Content Views. This example shows how to create a filter to exclude the **emacs** package from the **cv-rhel7-server** Content View.

First create a filter for the Content View in the organization:

```
$ hammer content-view filter create \
--type rpm \
--name exclude-emacs \
--description "Excluding emacs package" \
--inclusion false \
--organization $ORG \
--repository-ids 1,2,3 \
--content-view cv-rhel7-server
```

Find the repository IDs by executing **hammer repository list**. Create a rule to exclude packages with name starting with "emacs" and add it to the filter as follows:

```
$ hammer content-view filter rule create \
--name "emacs*" \
--organization $ORG \
--content-view cv-rhel7-server \
--content-view-filter exclude-emacs
```

As a result, hosts using the cv-rhel7-server Content View will not have access to the emacs package. You can add multiple rules to a filter, see the output **hammer content-view rule create --help** for the full list of filtering parameters. To inspect rules present in a filter, issue the following command:

```
$ hammer content-view filter rule list \
--content-view cv-rhel7-server \
--content-view-filter exclude-emacs \
--organization $ORG
```

Example 3.5. Filtering Packages for a Content View – Limiting Errata by Date

This example shows how to create a filter to exclude errata released before a specific date from the **cv-rhel7-server** Content View. For more information on errata management see Chapter 8, *Managing Errata*. Create a filter for the Content View as follows:

```
$ hammer content-view filter create \
--type erratum \
--name limit-errata-by-date \
--description "Excluding errata by date" \
--inclusion false \
--organization $ORG \
--repository-ids 1,2,3 \
--content-view cv-rhel7-server
```

Create a rule to exclude errata with a name starting with "emacs" and add it to the filter as follows:

```
$ hammer content-view filter rule create \
--end-date <YYYY-MM-DD> \
--organization $ORG \
--content-view cv-rhel7-server \
--content-view-filter limit-errata-by-date \
--types enhancement,bugfix,security
```

3.2.2. Adding Puppet Modules to a Content View

To add a Puppet module to a Content View, first upload this module to a Puppet repository within a custom product. Use the commands from Section 2.3.4, “Creating a Custom Repository” to create a product with a repository, and to upload a Puppet module to the repository.

To add a Puppet module to a Content View, issue the following command:

```
$ hammer content-view puppet-module add \
--content-view <cv_name> \
--name <module_name>
```

Example 3.6. Adding a Puppet Module to a Content View

This example shows how to add a Puppet module from an external source to the cv-rhel7-server Content View.

1. Download the **concat** module (that constructs files from multiple text fragments) from Puppet Forge:

```
$ wget -O /tmp/puppetlabs-concat-1.2.5.tar.gz
https://forgeapi.puppetlabs.com
/v3/files/puppetlabs-concat-1.2.3.tar.gz
```

2. Create a Puppet repository under the ACME-puppet product and upload the module to this repository (the example assumes repository ID is 6):

```
$ hammer product create \
--name "ACME-puppet" \
--organization $ORG
```

```
$ hammer repository create \
--organization $ORG \
--product ACME-puppet \
--name "ACME Puppet Repository" \
--content-type puppet \
--url "https://forge.puppetlabs.com/"
```

```
$ hammer repository upload-content \
--organization $ORG \
--product ACME-puppet \
--id 6 \
--path /tmp/puppetlabs-concat-1.2.5.tar.gz
```

3. Add the module to the Content View using the **id**, **name**, or **author** parameters. To find the exact values, enter:

```
$ hammer puppet-module list --organization $ORG
---|--------|------------|--------
ID | NAME   | AUTHOR     | VERSION
---|--------|------------|--------
1  | concat | puppetlabs | 1.2.3
---|--------|------------|--------
```

To add the module to the Content View, issue:

```
$ hammer content-view puppet-module add \
--name concat \
--content-view cv-rhel7-server \
--organization $ORG
```

To verify if the module has been added successfully, issue the following command:

```
$ hammer content-view puppet-module list \
--content-view cv-rhel7-server \
--organization $ORG
```

3.2.3. Adding Docker Images to a Content View

You can upload Docker images directly to the dedicated repository as follows:

```
$ hammer repository upload-content --path <image_archive> --id <repo_id>
```

Replace *<image_archive>* with a path to the archive containing the Docker image. Use *<repo_id>* to identify the repository of docker content type. Then you can add this repository to the Content View.

3.3. PUBLISHING A CONTENT VIEW

By publishing a Content View you make it visible and usable by hosts. Use the following command to publish a selected Content View:

```
$ hammer content-view publish \
```

```
--id <cv_ID> \
--organization-label <org_label> \
--async
```

Find the *<cv_ID>* of the Content View to be published in the output of the **`hammer content-view list`** command. Published Content Views become available in the Library environment. To verify the Content View status, issue the following command:

```
$ hammer content-view info --id <cv_ID>
```

3.4. PROMOTING A CONTENT VIEW

Promoting is the act of moving a Content View from one life cycle environment to another. To do so, issue the following command.

```
$ hammer content-view version promote \
--content-view <cv_name> \
--organization-label <org_label> \
--to-lifecycle-environment <env_name>
```

Here, *<env_name>* stands for the name of target life cycle environment.

Example 3.7. Promoting a Content View Through the Life Cycle Environment Path

The following Bash script promotes the selected Content View from Library through all life cycle environments in the ACME organization:

```
ORG="ACME"
CV_ID=1

for i in $(hammer --csv lifecycle-environment list --organization $ORG |
grep -vi '^ID' | awk -F, {'print $1'} | sort -n)
do
   hammer content-view version promote --organization $ORG --to-
lifecycle-environment-id $i --id $CV_ID
done
```

To verify if the Content View has been promoted correctly, issue the following command:

```
$ hammer content-view version info --id 1
```

3.5. PERFORMING AN INCREMENTAL UPDATE OF A CONTENT VIEW

Incremental updates enable modifying a published Content View without the need to promote a new Content View version through the life cycle environment. As a result of the incremental update, a new minor Content View version is created. Incremental updates are useful for fast emergency updates, you can use them to add errata, packages, or Puppet modules.

To create an incremental update adding new packages to a Content View, issue:

```
$ hammer content-view version incremental-update \
```

```
--content-view-version-id <cv_ID> \
--packages <pkg_name1>,<pkg_name2> \
--lifecycle-environment-ids <env_ID1>, <env_ID2>,...
```

Find the Content View version ID in the output of **`hammer Content View version list`**. Instead of supplying packages with the **`--packages`** option, you can add Puppet modules with **`--puppet-modules`**, or errata with **`--errata-ids`** (see Example 3.8, "Adding Errata to a Content View using an Incremental Update"). For more information on working with incremental updates issue **`hammer content-view version incremental-update --help`**.

Example 3.8. Adding Errata to a Content View using an Incremental Update

This example shows how to apply an erratum to a host (named **auth01.example.com**) by creating an incremental update of its Content View:

```
$ hammer content-view version incremental-update \
--content-view-version-id 4 \
--errata-ids 8c3801f6-12a7-4a62-83f4-addbb1f34ce6 \
--lifecycle-environments Infrastructure
```

To find the required information for the above command, perform the following steps:

1. Find the Content View your host is registered to as well as its life cycle environment by executing:

   ```
   $ hammer content-host info --name auth01.example.com --
   organization $ORG
   ```

2. Then find the current version of the Content View (assuming Content View name **RHEL7_Infra**):

   ```
   $ hammer content-view info --name "RHEL7_Infra" --organization
   $ORG
   ```

3. Find the IDs of errata you want to apply in the list of applicable errata in **Library**:

   ```
   $ hammer erratum list --content-view "RHEL7_Infra" --organization
   $ORG
   $ hammer host errata list --host auth01.example.com
   ```

CHAPTER 4. MANAGING ACTIVATION KEYS

Activation keys define the subscription properties of a host. Using an activation key improves the speed of host registration. For web UI equivalents of the following procedures see the Red Hat Satellite Host Configuration Guide.

There are three possible use cases for activation keys:

- **Activation key with no subscriptions specified** – hosts using the activation key search for the best fitting subscription. This is akin to running **`subscription-manager --auto-attach`**. See Example 4.1, “Creating an Empty Activation Key”.
- **Activation key providing a custom subscription pool for auto attach** – hosts using the activation key select the best fitting subscription from the list specified in the activation key. See Example 4.2, “Creating an Activation Key with Custom Subscription Pool”.
- **Activation key with the exact set of subscriptions** – hosts using the activation key are associated with all subscriptions specified in the activation key. See Example 4.3, “Creating an Activation Key with Mandatory Subscription List”.

NOTE

Activation keys are only used when hosts are registered. If changes are made to an activation key, it is only applicable to hosts that are registered with the amended activation key in the future. The changes are not made to existing hosts.

To create an activation key, issue the following command:

```
$ hammer activation-key create --name <ak_name> \
--organization-label <org_label> \
--content-view <cv_name> \
--lifecycle-environment <lc_name>
```

Note that the Content View has to be published. To see the full list of operations related to activation keys, use the **`hammer activation-key --help`** command.

To add a subscription to the activation key, issue the following command:

```
$ hammer activation-key add-subscription \
--id <ak_ID> \
--subscription-id <sub_ID>
```

To find the activation key ID, use **`hammer activation-key list`**; to find the subscription ID, use **`hammer subscription list`**.

Example 4.1. Creating an Empty Activation Key

This example shows how to create an activation key that directs the associated hosts to automatically attach a best fitting subscription:

```
$ hammer activation-key create \
--name "automatically attach key" \
--organization $ORG \
```

```
--content-view cv-rhel7-server \
--lifecycle-environment Testing
```

As a result, hosts registered in the cv-rhel7-server Content View are associated with this activation key.

Example 4.2. Creating an Activation Key with Custom Subscription Pool

This example shows how to create an activation key that will direct the associated hosts to automatically attach a best fitting subscription from the list specified in the activation key.

First create an empty activation key:

```
$ hammer activation-key create \
--name "custom pool key" \
--organization $ORG \
--content-view cv-rhel7-server \
--lifecycle-environment Testing
```

Add a subscription to the activation key:

```
$ hammer activation-key add-subscription \
--name "custom pool key" \
--subscription-id 1
```

Keep repeating this step to add all required subscriptions to the activation key.

Example 4.3. Creating an Activation Key with Mandatory Subscription List

This example shows how to create an activation key that will direct the associated hosts to attach all subscriptions specified in the activation key.

First, create an activation key and add all required subscriptions to it. Follow the steps from Example 4.2, “Creating an Activation Key with Custom Subscription Pool”.

Then disable the **`auto-attach`** property of the activation key:

```
$ hammer activation-key update \
--organization $ORG \
--name "mandatory subs key" \
--auto-attach false
```

IMPORTANT

You can assign several activation keys to a Content View. In case of conflicting settings, the values from the last specified key take precedence. You can specify the order of precedence by setting a host group parameter as follows:

```
$ hammer hostgroup set-parameter \
--name kt_activation_keys \
--value <name_of_first_key>, <name_of_second_key>,... \
--hostgroup <hostgroup_name>
```

CHAPTER 5. CONFIGURING PROVISIONING ENVIRONMENT

This section shows how to configure various stages of your provisioning environment using **hammer**. For web UI equivalents of the following procedures see the Red Hat Satellite Host Configuration Guide.

5.1. DOMAINS

Domains in Red Hat Satellite represent DNS zones. Satellite has the ability to assign domain names with Red Hat Satellite Capsule Server DNS. This provides users with a means to group and name hosts within a particular domain and associate them with parameters and Puppet variables.

To create a new domain, issue the following command:

```
$ hammer domain create --name <domain_name>
```

You can associate the newly created domain to organizations and locations using the **hammer organization add-domain** or **hammer location add-domain** commands. To view the status of a domain, issue the following command:

```
$ hammer domain info --name <domain_name>
```

5.2. SUBNETS

Subnets in Red Hat Satellite define networks specified for groups of systems. Subnets use standard IP-address settings to define the network and use the Red Hat Satellite Capsule Server's DHCP features to assign IP addresses to systems within the subnet. The following command contains the minimal set of options required for subnet creation:

```
$ hammer subnet create \
--name <subnet_name> \
--organization-ids <org_ID1>,<org_ID2>... \
--location-ids <loc_ID1>,<loc_ID2>... \
--domain-ids <dom_ID1>,<dom_ID2>... \
--boot-mode <boot_mode> \
--network <network_address> \
--mask <netmask> \
--ipam <ipam>
```

Here, *<boot_mode>* is one of **Static** or **DHCP**, *<ipam>* is one of **DHCP**, **Internal DB**, or **None**. If using DHCP, you can set the IP range with the **--from** and **--to** options. For the full list of configurable options, see the output of the **hammer subnet create --help** command.

5.3. ARCHITECTURES

Architecture in Satellite represents a logical grouping of hosts and operating systems. To view the architectures, issue the following command:

```
$ hammer architecture list
```

Architectures are created by Satellite automatically when hosts are registered in Puppet, therefore it is rarely needed to create them manually (even though **hammer** provides this option).

5.4. COMPUTE RESOURCES

Compute resources are hardware abstractions from virtualization and cloud providers. Satellite uses compute resources to provision virtual machines and containers. Run the following command to create a compute resource:

```
$ hammer compute-resource create \
--name <cr_name> \
--organization-ids <org_ID1>,<org_ID2>... \
--location-ids <loc_ID1>,<loc_ID2>... \
--provider <provider>
```

Here, *<provider>* is one of: **RHEV**, **RHEL OpenStack Platform**, **Libvirt**, **Docker**, **Rackspace**, **Google**, **EC2**, or **VMware**. Depending on the provider type, other options such as **--url**, or **--user** may be required. See the output of the **hammer compute-resource create --help** command for details.

5.5. INSTALLATION MEDIA

Installation media (ISO images) provide content for kickstart trees and new host installations in Red Hat Satellite. To list the media, issue the following command:

```
$ hammer medium list
```

To add a new medium, issue the following command:

```
$ hammer medium create --name <medium_name> --path <path_to_medium>
```

You can make the medium available to organizations and locations directly when adding it (see the output of the **hammer medium create --help** command), or later by using the **hammer organization add-medium** or **hammer location add-medium** commands.

5.6. PARTITION TABLES

Partition tables define the partitions and file system layout for new installations when provisioning systems. Red Hat Satellite provides default partition tables associated with operating system families, to view them, issue the following command:

```
$ hammer partition-table list
```

To create a new partition table, issue the following command:

```
$ hammer partition-table create \
--name <table_name> \
--file <path_to_layout_file> \
--os-family <os_family>
```

See the output of the **hammer partition-table --help** command for other subcommands.

5.7. PROVISIONING TEMPLATES

Provisioning templates provide the systematic means to run unattended installations. To view the provisioning templates provided by Satellite, issue the following command:

```
$ hammer template list
```

To add a new template, issue the following command:

```
$ hammer template create --name <template_name> --file
<path_to_template_file>
```

See the output of the **hammer template --help** command for other subcommands.

5.8. OPERATING SYSTEMS

Operating systems define combinations of installation methods and media and are grouped within families. As a default, Red Hat Satellite uses a Red Hat family. Families allow Satellite to change certain behaviors when provisioning hosts. To list operating systems, issue the following command:

```
$ hammer os list
```

To create a new operating system, issue the following command:

```
$ hammer os create --name <os_name> --major <version_number>
```

Then you can add architectures, partition tables, installation media, and configuration templates to the operating system. See the output of the **hammer os --help** command for details.

Example 5.1. Updating Multiple Operating Systems

The following Bash script assigns each operating system a partition table (**Kickstart default**), configuration template (**Kickstart default PXELinux**), and provisioning template (**Satellite Kickstart Default**).

```
PARTID=$(hammer --csv partition-table list | grep "Kickstart default" |
cut -d, -f1)
PXEID=$(hammer --csv template list --per-page=1000 | grep "Kickstart
default PXELinux" | cut -d, -f1)
SATID=$(hammer --csv template list --per-page=1000 | grep "provision" |
grep "Satellite Kickstart Default" | cut -d, -f1)

for i in $(hammer --csv os list | grep -vi '^ID' | awk -F, {'print $1'})
do
   hammer partition-table add-operatingsystem --id="${PARTID}" --
operatingsystem-id="${i}"
   hammer template add-operatingsystem --id="${PXEID}" --
operatingsystem-id="${i}"
   hammer os set-default-template --id="${i}" --config-template-
id="${PXEID}"
   hammer os add-config-template --id="${i}" --config-template-
id="${SATID}"
   hammer os set-default-template --id="${i}" --config-template-
id="${SATID}"
done
```

You can add **grep** commands to the **for** statement to further specify the affected operating systems. To verify if the assignment was performed correctly, use the **hammer os info** command.

5.9. PARAMETERS

Parameters define the behavior of Red Hat Satellite during provisioning. There are several types of parameters, see the Red Hat Satellite Host Configuration Guide for details. Use the following example to set a global parameter:

```
$ hammer global-parameter set --name <param_name> --value <param_value>
```

Example 5.2. Setting a Global Parameter to Disable the Firewall

Run the following command to set the **firewall** global option to disabled:

```
$ hammer global-parameter set --name firewall --value --disabled
```

To verify the setting, issue the following command:

```
$ hammer global-parameter list
---------|-------------
NAME     | VALUE
---------|-------------
firewall | --disabled
---------|-------------
```

Similarly, you can use **hammer** to set other parameter types:

- To set domain parameters, use:

```
$ hammer domain set-parameter \
--name <param_name> \
--value <param_value> \
--domain <domain_name>
```

- To set host group parameters, use:

```
$ hammer hostgroup set-parameter \
--name <param_name> \
--value <param_value> \
--hostgroup <hg_name>
```

- To set host parameters, use:

```
$ hammer host set-parameter \
--name <param_name> \
--value <param_value> \
--host <h_name>
```

- To update smart class parameters, use:

```
$ hammer sc-param \
--name <param_name> \
--default-value <param_value>
```

CHAPTER 6. MANAGING HOSTS

Host refers to any physical or virtual system Red Hat Satellite manages. This section shows how to create and configure hosts and host groups using **hammer**. For web UI equivalents of the following procedures see the Red Hat Satellite Host Configuration Guide.

6.1. CREATING A HOST GROUP

A host group is a collection of hosts or host groups. It is recommended to create host groups to hold shared host parameters. Members of the host group inherit these parameters, therefore you do not have to set them individually during host creation. Note that you can nest host groups in a hierarchical manner.

The following command demonstrates a basic set of options for creating a host group:

```
$ hammer hostgroup create \
--name "<hostgroup_name>" \
--environment "<environment_name>" \
--architecture "<architecture_name>" \
--domain <domain_name> \
--subnet <subnet_name> \
--puppet-proxy <proxy_name> \
--puppet-ca-proxy <ca-proxy_name> \
--operatingsystem "<os_name>" \
--partition-table "<table_name>" \
--medium "<medium_name>" \
--organization-ids <org_ID1>,<org_ID2>... \
--location-ids <loc_ID1>,<loc_ID2>...
```

See **hammer hostgroup create --help** for the full list of configurable options. There are two settings that cannot be configured during host group creation:

- An activation key has to be added afterwards using:

```
$ hammer hostgroup set-parameter \
--hostgroup "<hostgroup_name>" \
--name "kt_activation_keys" \
--value <key_name>
```

Run **hammer activation-key list** to find the activation key name (see Chapter 4, *Managing Activation Keys* for details on activation keys).

- The **root** password has to be specified when adding a host to the host group.

Example 6.1. Creating Host Groups for Multiple Content Views

The following Bash script creates a host group for each life cycle environment.

```
MAJOR="7"
OS=$(hammer --output csv os list | awk -F "," "/RedHat ${MAJOR}/ {print
\$2;exit}")
ARCH="x86_64"
ORG="ACME"
LOCATIONS="london,munich"
```

```
PTABLE_NAME="ptable-acme-os-rhel-server"
DOMAIN="example.com"

hammer lifecycle-environment list --organization "${ORG}" | awk -F "|"
'/[[:digit:]]/ {print $2}' | sed s'/ //' | while read LC_ENV
do
  if [[ ${LC_ENV} == "Library" ]]; then
    continue
  fi

  LC_ENV_LOWER=$(echo ${LC_ENV} | tr '[[:upper:]' '[[:lower:]]')
  ParentID=$(hammer --output csv hostgroup list --per-page 999 | awk -
F"," "(\$3 ~ /^${LC_ENV_LOWER}$/) {print \$1}")

  hammer hostgroup create --name "rhel-${MAJOR}server-${ARCH}" \
    --medium
"${ORG}/Library/Red_Hat_Server/Red_Hat_Enterprise_Linux_${MAJOR}_Server_
Kickstart_${ARCH}_${MAJOR}
    Server" \
    --parent-id ${ParentID} \
    --architecture "${ARCH}" \
    --operatingsystem "${OS}" \
    --partition-table "${PTABLE_NAME}" \
    --subnet "${DOMAIN}" \
    --domain "${DOMAIN}" \
    --organizations "${ORG}" \
    --locations "${LOCATIONS}" \
    --content-view "cv-os-rhel-${MAJOR}Server" \
    --environment-id $(hammer --output csv environment list --per-page
999 | awk -F "," "/KT_$ {ORG}_${LC_ENV}_cv_os_rhel_${MAJOR}Server/
{print \$1}")

  HgID=$(hammer --output csv hostgroup list --per-page 999 | awk -F"," "
(\$3 ~ /^${LC_ENV_LOWER}\/rhel-${MAJOR}server-${ARCH}$/) {print \$1}")

  hammer hostgroup set-parameter \
    --hostgroup-id "${HgID}" \
    --name "kt_activation_keys" \
    --value "act-${LC_ENV_LOWER}-os-rhel-${MAJOR}server-${ARCH}"
done
```

6.2. CREATING A HOST

It is recommended to set general parameters in a host group to reduce the number of required options when creating a host. The following command creates a basic host associated to a host group:

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

```
--organization-id <org_ID> \
--location-id <loc_ID> \
--ask-root-password yes
```

After executing the above command you will be prompted 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, see Table 6.1, "Available Keys for the --interface Option".

If you decide to create a host without host group membership, specify additional options described in Section 6.1, "Creating a Host Group". There is a wide range of available host parameters, for details see the output of **hammer host create --help**. The value of certain parameters depends on the type of compute resource the host is provisioned on, see Table 6.2, "Host Options Specific to Provider" for reference.

Table 6.1. Available Keys for the --interface Option

Keys	Description
type	Defines the interface type, one of **Nic::Managed**, **Nic::BMC**, **Nic::Bond**.
name, identifier	Identification of the interface.
mac, ip, domain (or domain_id), subnet (or subnet_id)	Network settings, domain and subnet identification can be inherited from the host group.
primary, provision, managed, virtual	Accept **true** or **false**. Managed hosts needs to have one primary and provisioning interface.
Specific to virtual interfaces	
tag	VLAN tag, this attribute has precedence over the subnet VLAN ID.
attached_to	Identifier of the interface to which the virtual interface belongs, for example **eth1**.
Specific to bonded interfaces	
mode	Bonding mode, one of **balance-rr**, **active-backup**, **balance-xor**, **broadcast**, **802.3ad**, **balance-tlb**, **balance-alb**.
Specific to BMC interfaces	
provider	BMC provider, set to **IPMI**.
username, password	BMC access credentials.

Keys	Description
Specific to hosts provisioned on Libvirt	
compute_type	Interface type, one of **`bridge`**, **`network`**.
compute_network or compute_bridge	Specifies interface name, pick one depending on the interface type.
compute_model	One of **`virtio`**, **`rtl8139`**, **`ne2k_pci`**, **`pcnet`**, **`e1000`**.
Specific to hosts provisioned on RHEV	
compute_name	Interface name, for example **eth0**.
compute_network	Select one of the available networks for a cluster, use UUID from RHEV.
Specific to hosts provisioned on VMware	
compute_type	Type of the network adapter, depends on your version of vSphere.
compute_network	Network ID form VMware.

Table 6.2. Host Options Specific to Provider

Provider	Keys
Keys for the --compute-attributes option	
EC2	flavor_id, image_id, availability_zone, security_group_ids, managed_ip
GCE	machine_type, image_id, network, external_ip
Libvirt	cpus, memory, start
OpenStack	flavor_ref, image_ref, tenant_id, security_groups, network
RHEV	cluster, template, cores, memory, start
VMware	cpus, corespersocket, memory_mb, cluster, path, guest_id, scsi_controller_type, hardware_version, start

Provider	Keys
Keys for the --volume option	
Libvirt	poll_name, capacity, format_type
RHEV	size_gb, storage_domain, bootable
VMware	datastore, name, size_gb, thin, eager_zero

Example 6.2. Creating a Host with a Bonded Interface Pair

The following example shows how to create a host with a bonded interface pair. For more information on interface bonding see the Red Hat Enterprise Linux Networking Guide.

```
$ hammer host create --name bondtest \
--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-id 1 \
--location-id 1 \
--ask-root-password yes
```

6.3. CREATING A HOST COLLECTION

A host collection in Red Hat Satellite is a group of hosts. The following command shows the minimal set of options required to create a host collection:

```
$ hammer host-collection create \
--organization-label <org_label> \
--name <hc_name>
```

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

```
$ hammer host-collection add-host \
--id <hc_ID> \
--host-ids <ch_ID1>,<ch_ID2>...
```

Run the following command to associate a host collection with an activation key (see Chapter 4, *Managing Activation Keys* for details on activation keys):

```
$ hammer activation-key add-host-collection \
--id <ak_ID> \
--host-collection <hc_name>
```

Hosts grouped in the host collection now inherit the configuration from the activation key.

6.4. RUNNING REMOTE JOBS ON HOSTS

The remote execution feature enables defining arbitrary commands on the Satellite Server and executing them on remote hosts. Commands are defined in job templates that are similar to provisioning templates. Several job templates are included by default, you can use them or define a custom template for example to manage software packages or start a Puppet process on remote hosts. For more information about remote execution on Satellite, see the Running Jobs on Satellite Hosts section of the *Host Configuration Guide*. To use this feature in Hammer, install the remote execution CLI module by executing the following command as **root**:

```
# yum install tfm-rubygem-hammer_cli_foreman_remote_execution
```

To list job templates available, issue:

```
$ hammer job-template list
```

To create a job template using a template-definition file, use a command as follows:

```
$ hammer job-template create \
--file "<template>" \
--name "<template_name>" \
--provider-type SSH \
--job-category "<category_name>"
```

Replace *<template>* with the path to the file containing the template definition. Specify a custom *<category_name>* or select one of the existing categories (**`Commands`**, **`Katello`**, **`Packages`**, **`Power`**, **`Puppet`**, or **`Services`**). See the output of **`hammer job-template create --help`** for information on other available parameters.

To invoke a job with custom parameters, issue:

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

```
--search-query "<query>"
```

Specify the template name you want to use for the remote job. Specify inputs as a comma separated list of key-value pairs. Run **hammer job-template info** to see what parameters are required by your template. Replace *<query>* with the filter expression defining which hosts will be affected (for example "name ~ rex01").

Example 6.3. Starting the httpd Service on Selected Hosts

This example shows how to execute a remote job based on the default **Service Action - SSH Default** template, that will start the **httpd** service on hosts that have a name that contains "target".

```
$ hammer job-invocation create \
--job-template "Service Action - SSH Default" \
--inputs service="httpd",action="start" \
--search-query "name ~ target"
```

To monitor the job output, issue:

```
$ hammer job-invocation output \
--id <job_ID> \
--host <host_name>
```

Find the *<job_ID>* in the output of **hammer job-invocation list**. For more information on executing remote commands with hammer, issue **hammer job-template --help** or **hammer job-invocation --help**.

CHAPTER 7. MANAGING USERS AND PERMISSIONS

For the administrator, Red Hat Satellite provides the ability to create, modify, and remove users. Also, it is possible to configure access permissions through assigning roles to users. This section shows how to perform these tasks using **hammer**. For web UI equivalents of the following procedures see Users and Roles in the *Red Hat Satellite Server Administration Guide*.

7.1. CREATING USERS

User in Red Hat Satellite defines a set of details for individuals using the system. To configure a user in Red Hat Satellite, **hammer** provides the **user create** and **user update** commands. Create a new user with the following command:

```
$ hammer user create \
--login <user_name> \
--password <user_password> \
--mail <user_mail> \
--auth-source-id 1 \
--organization-ids <org_ID1>,<org_ID2>...
```

The **--auth-source-id 1** setting means that the user is authenticated internally, you can specify an external authentication source as an alternative. Add the **--admin** option to grant administrator privileges to the user. Specifying organization IDs is not required, you can modify the user details later using the **update** subcommand.

For more information on user related subcommands see the output of **hammer user --help**.

7.2. CREATING USER GROUPS

You can manage permissions of several users at once by organizing them into user groups. User groups themselves can be further grouped to create a hierarchy of permissions. Use the following command to create a new user group:

```
$ hammer user-group create --name <usergroup_name>
```

To add a user to a user group, issue the following command:

```
$ hammer user-group add-user --user <user_name> --id <usergroup_id>
```

Find the user group ID by executing **hammer user-group list**. Similarly, you can add user groups using the **add-user-group** subcommand. For more information on operations related to user groups see the output of **hammer user-group --help**.

7.3. CREATING ROLES

Roles in Red Hat Satellite define a set of permissions and access levels. Satellite provides a number of predefined roles, to view them, enter the following command:

```
$ hammer role list
```

To view permissions associated with a role, issue the following command:

```
$ hammer role filters --id <role_id>
```

Here, *<role_id>* is the ID of the role from the output of **hammer role list**.

To create a custom role, issue the following command:

```
$ hammer role create --name <role_name>
```

Add a permission filter to the role with the following command:

```
$ hammer filter create \
--role <role_name> \
--permission-ids <perm_ID1>,<perm_ID2>...
```

Find the permissions to be added to the role by using **hammer filter available-permissions**. For details on roles and permissions see the output of **hammer role --help** and **hammer filter --help**.

Example 7.1. Granular Permission Filtering

Red Hat Satellite provides the ability to limit the configured user permissions to selected instances of a resource type. Use the **--search** option to limit permission filters, for example:

```
$ hammer filter create \
--permission-ids 91 \
--search "name ~ ccv*" \
--role qa-user
```

The above command adds to the **qa-user** role a permission to view, create, edit, and destroy Content Views that only applies to Content Views with name starting with **ccv**. See Granular Permission Filtering in the *Satellite Server Administration Guide* for more information.

7.4. ASSIGNING ROLES TO USERS

To assign a role to a user, issue the following command:

```
$ hammer user add-role --id <user_id> --role <role_name>
```

Similarly, you can assign a role to a user group:

```
$ hammer user-group add-role --id <usergroup_id> --role <role_name>
```

CHAPTER 8. MANAGING ERRATA

Software packages in Red Hat products are subject to updates, referred to as errata, that are released at regular intervals as well as asynchronously. This section shows how to inspect and apply errata using **hammer**. For web UI equivalents of the following procedures see the Red Hat Satellite Host Configuration Guide.

8.1. INSPECTING AVAILABLE ERRATA

To view errata that are available for all organizations, issue the following command:

```
$ hammer erratum list
```

Example 8.1. Filtering Errata

The **hammer erratum list** command provides numerous options for filtering and ordering the output list. For example, to find an erratum that contains a specific security fix, issue:

```
$ hammer erratum list --cve CVE-2014-0453
```

The following command displays applicable errata for the selected product that contain the specified bugs ordered so that the security errata are displayed on top:

```
$ hammer erratum list \
--product-id 7 \
--search "bug = 1213000 or bug = 1207972" \
--errata-restrict-applicable 1 \
--order "type desc"
```

For more information on syntax used in the **--search** option, refer to the Red Hat Satellite Host Configuration Guide. For more information on filtering options implemented in hammer, see the output of **hammer erratum list --help**.

To view details of a specific erratum, issue the following command:

```
$ hammer erratum info --id <erratum_ID>
```

Replace *<erratum_ID>* with a unique identifier of the erratum found in the output of the **hammer erratum list** command. You can identify errata also by name and repository name, see the output of **hammer erratum info --help** for details.

8.2. APPLYING ERRATA TO A HOST

To list errata available for a host, issue the following command:

```
$ hammer host errata list --host <hostname>
```

To apply selected errata to the host, issue the following command:

```
$ hammer host errata apply \
--host <hostname> \
```

```
--errata-ids <erratum_ID1>,<erratum_ID2>...
```

Example 8.2. Applying All Available Errata to a Host

The following Bash script applies all errata available to a host (auth01.example.com):

```
HOST="auth01.example.com"
for i in $(hammer --csv host errata list --host $HOST | grep -vi '^ID' |
awk -F, {'print $1'})
do
  hammer host errata apply --host $HOST --errata-ids $i
done
```

Example 8.3. Applying a Security Advisory

This example shows how to apply a security fix to hosts using **hammer**:

1. Find the erratum that contains a fix for a selected issue (CVE-2015-3238):

```
$ hammer erratum list --cve CVE-2015-3238
-------|----------------|----------|--------------------------
----
ID     | ERRATA ID      | TYPE     | TITLE
-------|----------------|----------|--------------------------
----
f30e66 | RHSA-2015:1640 | security | Moderate: pam security update
-------|----------------|----------|--------------------------
----
```

2. Verify if the security erratum (RHSA-2015:1640) is applicable for your host (auth01.example.com):

```
$ hammer host errata list \
--host auth01.example.com \
--search "RHSA-2015:1640"
```

3. Apply the erratum to the host:

```
$ hammer host errata apply \
--host auth01.example.com \
--errata-ids "RHSA-2015:1640"
```

You can use the following Bash script to apply a security erratum (for example RHSA-2015:1640) to all hosts where it is applicable:

```
ORG="ACME"
RHSA="RHSA-2015:1640"

for i in $(hammer --csv host list --organization $ORG | grep -vi '^ID' |
awk -F, {'print $2'})
```

```
do
  hammer host errata apply --host $i --errata-ids $RHSA
done
```

To see if errata were applied successfully, find the corresponding task in the output of the following command:

```
$ hammer task list
```

To review the state of a selected task, issue the following command:

```
$ hammer task progress --id <task_ID>
```

8.3. APPLYING ERRATA TO A HOST COLLECTION

To apply selected errata to a Host Collection, enter a command as follows:

```
$ hammer host-collection erratum install \
--errata "<erratum_ID1>,<erratum_ID2>,..." \
--name "my-collection" --organization $ORG
```

This command is available in Red Hat Satellite 6.2.8 or later.

CHAPTER 9. MANAGING DOCKER CONTAINERS

A Docker container is a sandbox for isolating applications. The container image stores the configuration for the container. This section shows how to use **hammer** to provision Docker containers. For web UI equivalents of the following procedures see the Red Hat Satellite Host Configuration Guide.

In Red Hat Satellite, you can deploy containers only on a compute resource of the Docker provider type. See the Satellite Host Configuration Guide for instructions on how to prepare a container host. To register this host as a compute resource, issue the following command:

```
$ hammer compute-resource create
--name <cr_name> \
--organization-ids <org_ID1>,<org_ID2>... \
--location-ids <loc_ID1>,<loc_ID2>... \
--url <cr_url> \
--provider docker
```

Use the following syntax to provision a container on the compute resource:

```
$ hammer docker container create \
--name <container_name> \
--compute-resource-id <cr_ID> \
--repository-name <repo_name> \
--tag <tag> \
--command <command>
```

Find the compute resource ID in the output of **hammer compute-resource list**. Replace *<repo_name>* with the name of the synchronized repository that contains your docker images. This can be a custom repository pointing to Docker Hub or your internal registry (see Section 2.3.4, “Creating a Custom Repository”), or the official Red Hat image repository. If you provision from a Content View, replace *<repo_name>* with the name of the Content View. See Section 3.2.3, “Adding Docker Images to a Content View” for details on adding images to a Content View.

By starting a container you start the process specified with the **--command** option during the container creation. To start a container, issue the following command:

```
$ hammer docker container start --id <container_ID>
```

For the full list of container related options, see the output of the **hammer docker container --help** command.