Red Hat Satellite 6.4

Content Management Guide

An end-to-end guide on managing content from Red Hat and custom sources

Last Updated: 2018-10-19

Red Hat Satellite 6.4 Content Management Guide

An end-to-end guide on managing content from Red Hat and custom sources

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

Abstract

This guide provides an end-to-end scenario on managing content in Red Hat Satellite 6. Examples of such content include RPM files, ISO images, Puppet modules, and container images. Red Hat Satellite 6 manages this content using a set of Content Views promoted across the application lifecycle. This guide demonstrates how to create an application lifecycle that suits your organization and content views that fulfils host states within lifecycle environments. These content views eventually form the basis for provisioning and updating hosts in your Red Hat Satellite 6 environment.

Table of Contents

CHAPTER 1. INTRODUCTION

In the context of system management, we define **content** as the software installed on systems. This includes, but is not limited to, the base operating system, middleware services, and end user applications. Red Hat Satellite 6 provides tools to manage the various types of content for Red Hat Enterprise Linux systems. This provides system administrators with an easy way to collect a range of content, keep it up to date, and use it to provision new systems and update existing systems.

This guide aims to provide an end-to-end scenario to demonstrate how to manage your content. This guide is targeted at system administrators with a newly installed Satellite Server.

1.1. OVERVIEW OF RED HAT SATELLITE 6 CONTENT MANAGEMENT

Content management in the context of Red Hat Satellite 6 means a work flow that provides a sustainable repository for multiple content types. For Red Hat content, Red Hat Satellite 6 also uses subscription information so that it knows what content is available to a user. This means Red Hat Satellite 6 uses components to manage the following:

- **Subscription management**, which includes tools to manage Red Hat software subscriptions, and associated content, over a secure connection. This provides a means for organizations to manage their Red Hat subscription information.

- **Content management**, which includes applications to download and store content in custom repositories. This provides organizations with a method to store Red Hat content and organize it in various ways.

1.2. DEFINING THE APPLICATION LIFE CYCLE

The **application life cycle** is a concept central to Red Hat Satellite 6's content management functions. The application life cycle defines how a particular system and its software look at a particular stage. For example, an application life cycle might be simple; you might only have a development stage and production stage. In this case the application life cycle might look like this:

- Development

- Production

However, a more complex application life cycle might have further stages, such as a phase for testing or a beta release. This adds extra stages to the application life cycle:

- Development

- Testing

- Beta Release

- Production

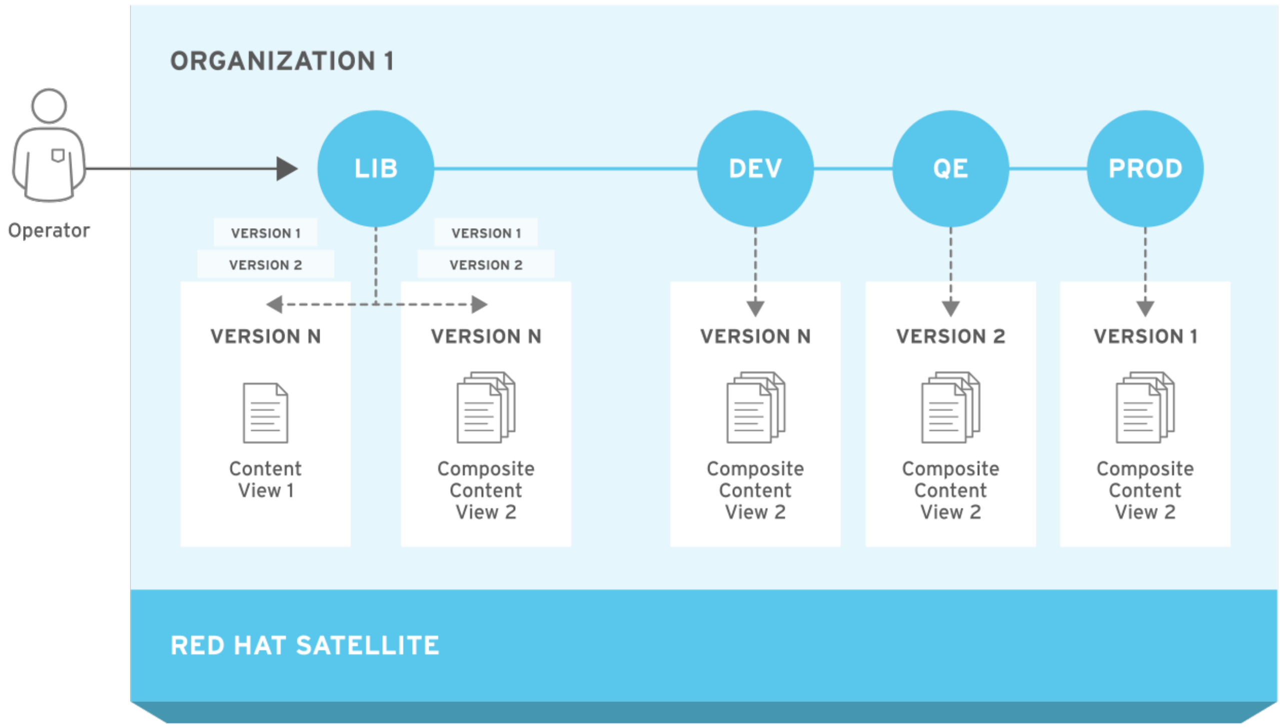
Ultimately, the stages in an application life cycle depend on your organization and its software development methods. However, Red Hat Satellite 6 provides methods to customize each application life cycle stage so that it suits your specifications.

Each stage in the application life cycle is called an **environment** in Red Hat Satellite 6. Each environment uses a specific collection of content. Red Hat Satellite 6 defines these content collections as a **Content View**. Each **Content View** acts as a filter where we can define what repositories, packages,

and Puppet modules to include in a particular environment. This provides a method for you to define specific sets of content to designate to each environment.

The concept of the application life cycle depends on a certain level of progression. For example, environments in the early stages of an application life cycle might use newer and pre-release packages for development and testing of new features. Likewise, environments in the later stages might use only stable packages to suit production-level software. If the packages in development pass testing, you can promote the development Content View across the application life cycle so that it becomes the Content View for the production environment. This ensures that your application life cycle progresses with the development of your application.

Figure 1.1. The Red Hat Satellite 6 Application Life Cycle



1.3. DEFINING CONTENT MANAGEMENT TYPES

Red Hat Satellite 6 manages different content types. This includes:

RPM Packages

Red Hat Satellite 6 provides a method to import RPM files from repositories related to your Red Hat subscriptions. This means Satellite Server downloads the RPM files from Red Hat's Content Delivery Network and stores them locally. You can use these repositories and their RPM files in Content Views.

Kickstart Trees

Red Hat Satellite 6 obtains the kickstart trees for creating a new system. New systems access these kickstart trees over a network to use as base content for their installation. Red Hat Satellite 6 also contains some predefined kickstart templates (and the ability to create your own), which are used to provision new systems and customize the installation.

ISO and KVM Images

Red Hat Satellite 6 downloads and manages media for installation and provisioning. For example, Satellite downloads, stores and manages ISO images and KVM guest images for specific Red Hat Enterprise Linux versions.

Puppet Modules

Red Hat Satellite 6 offers the ability to upload Puppet modules alongside RPM content so that it can configure the system's state after provisioning. Users can also manage Puppet classes and parameters as part of the provisioning process.

Container Images

Red Hat Satellite 6 can act as a registry for container images. This provides a method to create containers using Red Hat Enterprise Linux Atomic Host.

OSTree

Red Hat Satellite 6 can import OSTree branches and publish this content to a HTTP location.

1.4. DEFINING OUR SCENARIO

This guide uses an example scenario to demonstrate Red Hat Satellite 6's features. In this scenario, a software development company called **ACME** recently installed Red Hat Satellite 6 and its system administrators want to start importing and managing Red Hat content. The end goal for ACME is to:

- Have a set of content sources imported for their organization. This includes content from their Red Hat subscription and their own custom content.
- Have an application life cycle defined based on their software development process.
- Have their Satellite Server ready so they can provision new Red Hat Enterprise Linux hosts and register existing Red Hat Enterprise Linux hosts.

This guide only focuses on content management. Further features, such as host provisioning, environment architecture, and Satellite Server management, are covered in other guides in the Red Hat Satellite 6 series.

This guide provides both steps for using either the Red Hat Satellite 6 Web UI or its CLI tool — Hammer CLI. Use either depending on your preferred method of interacting with Red Hat Satellite 6. If using the CLI and do not want to include authentication details each time you enter a **`hammer`** command, edit the **`~/.hammer/cli.modules.d/foreman.yml`** file to include the credentials for a local user:

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

IMPORTANT

All uses of the **`hammer`** command in this guide use the configuration file and omit the authentication details.

Some CLI commands can use the **`--async`** option to perform certain tasks asynchronously. For example, you can synchronize content and publish Content Views as a asynchronous task rather than monitoring it. This guide omits the **`--async`** so that users can monitor progress of tasks to completion. If including the **`--async`** option for certain tasks, ensure the task completes before moving on to the next task.

1.5. CONTENT MANAGEMENT STORAGE

Red Hat Satellite 6 hosts repositories for RPM and Puppet content, including content synchronized with

Red Hat's Content Delivery Network and your own custom repositories. Such repositories grow in size over time. This means you must estimate adequate size requirements to suit your environment and scale future requirements accordingly.

Red Hat Satellite 6 temporarily adds content to **`/var/cache/pulp`**, then moves them into the **`/var/lib/pulp`** directory for storage and management. Ensure that both **`/var/cache/pulp`** and **`/var/lib/pulp`** have adequate storage. Mount the **`/var/lib/pulp`** directory onto a large local partition that you can scale. For example, use Logical Volume Manager (LVM) to create this partition.

IMPORTANT

Do not mount **`/var/lib/pulp`** on an NFS share. Parts of Red Hat Satellite 6 use transient SQLite databases, which have issues over NFS. If you want to use an NFS share, mount only the **`/var/lib/pulp/content`** directory, which contains the main source content units. Use of high-bandwidth, low-latency storage for the **`/var/lib/pulp`** file system. Red Hat Satellite has many operations that are I/O-intensive so usage of high-latency, low-bandwidth storage could result in performance degradation.

Red Hat Satellite 6 synchronize and stores packages from Red Hat's Content Delivery Network. This includes repositories for Red Hat Enterprise Linux and other Red Hat software. The recommended storage requirements for Red Hat content are as follows:

During Production Phase 1 of a major Red Hat Enterprise Linux version:

- At least 40GB for each binary package repository
- At least 80GB for each debug-info repository

After Production Phase 1 of a major Red Hat Enterprise Linux version:

- The estimated annual growth rates of these repositories are 10GB per binary package repository and 20GB per debug-info repository

For more information about Red Hat Production Phases, see "Red Hat Enterprise Linux Life Cycle".

NOTE

All repositories vary in size. These specifications are recommendations and must be adjusted according to the repositories you want to synchronize.

Red Hat Satellite 6 provides users with the ability to create Content Views. Content Views act as a snapshot of a user-defined content collection at a particular point in time. This provides a means to create customized content collections from preexisting repositories.

Each unit of content in a Content View uses a symbolic link to the Definitive Media Library stored in the **`/var/lib/pulp/content`** directory. In addition, each repository in a Content View contains metadata about the content belonging to the Content View. This means a Content View using a minimal number of packages uses a small amount of storage. However, the storage size adds up when you use multiple Content Views and a large number of packages per view.

For example, a Content View using the Red Hat Enterprise Linux 7 RPMs repository might contain over 7000 packages. In terms of disk space, this might only result in less than 100MB of symbolic links. However, take into account the following:

- The number of Content Views containing this repository

- The number of versions per Content View
- The number of life cycle environments using the promoted view
- Any additional content, such as kickstart trees or Live CD content

To help reduce the amount of storage that Content Views consume, use the following recommendations:

- Remove unused versions of Content Views. If you are not using a Content View in a life cycle environment and you have no intention of reusing it, delete it to reclaim the storage used.
- Use filters in Content Views. Filters limit the content that appears in a Content View. This helps you define only the necessary content required for your view and excludes redundant content. This reduces the size of each Content View significantly.
- Monitor the **`/var/lib/pulp/nodes`** directory. Red Hat Satellite 6 uses this directory to construct Content Views.
- Monitor the **`/var/lib/pulp/published`** directory. Red Hat Satellite 6 uses this directory to publish Content Views.

Red Hat Satellite 6 also uses the following databases for its content management components:

- **Main database** - PostgreSQL database stored in **`/var/lib/pgsql`**. The storage requirements depend on a number of factors including organizations, environments, registered systems, and Content Views.
- **Content database** - MongoDB database stored in **`/var/lib/mongodb`**. The storage requirements depend on the number of packages and Content Views for your Red Hat Satellite 6 environment. This usually takes a large amount of storage. Reserve at minimum 10GB for the Content database and plan for an estimated 5GB per repository. It is recommended to mount this directory onto a large local partition that you can scale. For example, use Logical Volume Manager (LVM) to create this partition.

IMPORTANT

Do not mount **`/var/lib/mongodb`** on an NFS share. Red Hat recommends the usage of high-bandwidth, low-latency storage for the **`/var/lib/mongodb`** file system. Red Hat Satellite has many operations that are I/O-intensive so usage of high-latency, low-bandwidth storage could result in performance degradation.

CHAPTER 2. MANAGING ORGANIZATIONS

Organizations divide Red Hat Satellite 6 resources into logical groups based on ownership, purpose, content, security level, or other divisions. You can create and manage multiple organizations through Red Hat Satellite 6, then divide and assign your Red Hat subscriptions to each individual organization. This provides a method of managing the content of several individual organizations under one management system. Here are some examples of organization management:

Single Organization

A small business with a simple system administration chain. In this case, we create a single organization for the business and assign content to it.

Multiple Organizations

A large company that owns several smaller business units. For example, a company with separate system administration and software development groups. In this case, we create organizations for the company and each of the business units it owns. This keeps the system infrastructure for each separate. We then assign content to each organization based on their needs.

External Organizations

A company that manages external systems for other organizations. For example, a company offering cloud computing and web hosting resources to customers. In this case, we might create an organization for the company's own system infrastructure and then an organization for each external business. We then assign content to each organization where necessary.

A default installation of Red Hat Satellite 6 provides a default organization called **Default_Organization**.

IMPORTANT

If a new user is not assigned a default organization their access is limited. To grant systems rights to users, assign them to a default organization and have them log out and log back in again.

2.1. CREATING AN ORGANIZATION

For Web UI Users

1. Navigate to **Administer** > **Organizations**.
2. Click **New Organization**.
3. Provide the base details for the organization:
    - **Name** - A plain text name for the organization.
    - **Label** - A unique identifier for the organization. This is used for creating and mapping certain assets, such as directories for content storage. Use letters, numbers, underscores, and dashes, but no spaces.
    - **Description** - An optional plain text description for the organization.
4. Click **Submit**.
    - If you have hosts with no organization assigned, you are redirected to the **Select Hosts** tab. Select between three variants of assigning hosts to the organization:
        - **Assign All** to assign all hosts with no organization assigned.

- **Manually Assign** to select which hosts to assign to the organization.
- **Proceed to Edit** to move to the **Edit Properties** page without assigning hosts to the organization

- If you do not have hosts with no organization assigned, you are redirected to the **Edit** page, where you can assign infrastructure resources to the organization. This includes networking resources, installation media, kickstart templates, and other parameters. You can return to this page at any time by navigating to **Administer** > **Organizations** and then selecting an organization to edit.

5. After completing your organization creation, click **Submit**.

For CLI Users

Enter the following command to create an organization:

```
# hammer organization create \
--name "your_organization_name" \
--label "your_organization_label \
--description "your_organization_description"
```

2.2. SETTING THE ORGANIZATION CONTEXT

An organization context defines the organization to use for a host and its associated resources.

For Web UI Users

The organization menu is the first menu item in the menu bar, on the upper left of the Satellite web UI. If you have not selected a current organization, the menu says **Any Organization**. Click the **Any Organization** button and select the organization to use.

For CLI Users

While using the CLI, include either **`--organization "your_organization_name"`** or **`--organization-label "your_organization_label"`** as an option. For example:

```
# hammer subscription list --organization "Default_Organization"
```

This command outputs subscriptions allocated for the Default_Organization.

2.3. CREATING AN ORGANIZATION DEBUG CERTIFICATE

To Create a New Organization Debug Certificate:

1. Navigate to **Administer** > **Organizations**.
2. Select an organization for which you want to generate a debug certificate.
3. Click **Generate and Download**. This generates a debug certificate.
4. Save the certificate file in a secure location.

> **NOTE**
>
> Debug Certificates are automatically generated for provisioning template downloads if they do not already exist in the organization for which they are being downloaded.

2.4. USING AN ORGANIZATION DEBUG CERTIFICATE

You can view an organization's repository content using a browser or using the API if you have a debug certificate for that organization. The previous section describes creating and downloading the certificate which is in the X.509 format. To use a browser you must first convert the X.509 certificate to a format your browser supports and then import the certificate. The **`curl`** utility only requires extracting the certificate and key into separate files.

To Use an Organization Debug Certificate in Firefox.

1. Create and download an organization certificate as described in Section 2.3, "Creating an Organization Debug Certificate".

2. Open the X.509 certificate, for example, for the default organization:

   ```
   $ vi 'Default Organization-key-cert.pem'
   ```

3. Copy the contents of the file from: **`-----BEGIN RSA PRIVATE KEY-----`** to **`-----END RSA PRIVATE KEY-----`** inclusive, into a file called **`key.pem`**.

4. Copy the contents of the file from **`-----BEGIN CERTIFICATE-----`** to **`-----END CERTIFICATE-----`** inclusive, into a file called **`cert.pem`**.

5. Enter a command as follows to create a PKCS12 format certificate and enter a password or phrase when prompted:

   ```
   $ openssl pkcs12 -keypbe PBE-SHA1-3DES -certpbe PBE-SHA1-3DES -
   export -in cert.pem -inkey key.pem -out organization_label.pfx -name
   organization_name
   Enter Export Password:
   Verifying - Enter Export Password:
   ```

6. Using the preferences tab, import the resulting **`pfx`** file into your browser: Navigate to **Edit** > **Preferences** > **Advanced Tab**. Select **View Certificates** in the **Certificates** view to open the **Certificate Manager**. On the **Your Certificates** tab, click **Import** and select the **`pfx`** file to load. You are prompted for the password or phrase used when making the certificate.

7. Enter a URL in the following format into your browser's address bar to begin browsing for repositories:

   ```
   http://satellite.example.com/pulp/repos/organization_label
   ```

   Pulp uses the organization label so the URL must use the organization label too.

To Use an Organization Debug Certificate with Curl:

1. Create and download an organization certificate as described in Section 2.3, "Creating an Organization Debug Certificate"

2. Open the X.509 certificate, for example, for the default organization:

```
$ vi 'Default Organization-key-cert.pem'
```

3. Copy the contents of the file from: **`-----BEGIN RSA PRIVATE KEY-----`** to **`-----END RSA PRIVATE KEY-----`** inclusive, into a file called **`key.pem`**.

4. Copy the contents of the file from **`-----BEGIN CERTIFICATE-----`** to **`-----END CERTIFICATE-----`** inclusive, into a file called **`cert.pem`**.

5. Find a valid URL for a repository. You can use the browsing method described in the previous procedure or use the web UI. For example, using the web UI, navigate to **Content** > **Products** and select a Product by name. On the **Repositories** tab, select a repository by name and look for the **Published At** entry.

6. To use **`curl`** to access a repository, enter the following command:

```
$ curl -k --cert cert.pem --key key.pem
http://satellite.example.com/pulp/repos/Default_Organization/Library
/content/dist/rhel/server/7/7Server/x86_64/sat-tools/6.4/os/
```

Ensure the paths to **`cert.pem`** and **`key.pem`** are the correct absolute paths otherwise the command fails silently.

2.5. DELETING AN ORGANIZATION

You can delete an organization if the organization is not associated with any life cycle environments or host groups. If there are any life cycle environments or host groups associated with the organization you are about to delete, remove them by navigating to **Administer** > **Organizations** and clicking the relevant organization. It is not recommended to delete the default organization created during installation because the default organization is a placeholder for any unassociated hosts in the Satellite environment. There must be at least one organization in the environment in any given time.

For Web UI Users

1. Navigate to **Administer** > **Organizations**.

2. Select **Delete** from the list to the right of the name of the organization you want to delete.

3. Click **OK** to delete the organization.

For CLI Users

Enter the following command to delete an organization:

```
# hammer organization delete --organization "your_organization_name"
```

CHAPTER 3. MANAGING LOCATIONS

Locations function similar to organizations, they provide a method to group resources and assign hosts. Differences between organizations and locations are:

- Locations are based on physical or geographical settings.
- Locations have hierarchical structure.

3.1. CREATING A LOCATION

For Web UI Users

1. Navigate to **Administer** > **Locations**.
2. Click **New Location**.
    - Provide the base details for the location:
    - **Parent** - An optional parent location for this location. This creates a location hierarchy.
    - **Name** - A plain text name for the location.
    - **Description** - An optional plain text description for the location.
3. Click **Submit**.
    - If you have hosts with no location assigned, you are redirected to the **Select Hosts** tab. Select between three variants of assigning hosts to the location:
        - **Assign All** to assign all hosts with no location assigned.
        - **Manually Assign** to select which hosts to assign to the location.
        - **Proceed to Edit** to move to the **Edit Properties** page without assigning hosts to the location.
    - If you do not have hosts with no location assigned, you are redirected to the **Edit** page, where you can assign infrastructure resources to the location. This includes networking resources, installation media, kickstart templates, and other parameters. You can return to this page at any time by navigating to **Administer** > **Locations** and then selecting a location to edit.
4. After completing your location creation, click **Submit**.

For CLI Users

Enter the following command to create a location:

```
# hammer location create \
--parent-id "parent_location_id" \
--name "your_location_name" \
--description "your_location_description"
```

3.2. SETTING THE LOCATION CONTEXT

A location context defines the location to use for a host and its associated resources.

For Web UI Users

The location menu is the second menu item in the menu bar, on the upper left of the Satellite web UI. If you have not selected a current location, the menu says **Any Location**. Click the **Any location** button and select the location to use.

For CLI Users

While using the CLI, include either **`--location "your_location_name"`** or **`--location-id "your_location_id"`** as an option. For example:

```
# hammer subscription list --location "Default_Location"
```

This command outputs subscriptions allocated for the Default_Location.

3.3. DELETING A LOCATION

You can delete a location if the location is not associated with any life cycle environments or host groups. If there are any life cycle environments or host groups associated with the location you are about to delete, remove them by navigating to **Administer** > **Locations** and clicking the relevant location. It is not recommended to delete the default location created during installation because the default location is a placeholder for any unassociated hosts in the Satellite environment. There must be at least one location in the environment in any given time.

For Web UI Users

1. Navigate to **Administer** > **Locations**.
2. Select **Delete** from the list to the right of the name of the location you want to delete.
3. Click **OK** to delete the location.

For CLI Users

Enter the following command to delete a location:

```
# hammer location delete --location "your_location_name"
```

CHAPTER 4. MANAGING SUBSCRIPTIONS

Red Hat Satellite 6 imports content from the Red Hat Content Delivery Network (CDN). A subscription manifest is used to find, access, and download content from the corresponding repositories. You must have a subscription manifest containing a subscription allocation for each organization on your Satellite Server. All subscription information is available in your Red Hat Customer Portal account.

Use this chapter to create, modify, and refresh subscription allocations and manifests.

4.1. MANAGING SUBSCRIPTION ALLOCATIONS

4.1.1. Creating a Subscription Allocation in the Customer Portal

You can access your subscription information in the Red Hat Customer Portal and create a subscription allocation. You can then export the allocation as a subscription manifest for use in on-premise management applications, such as Red Hat Satellite.

Procedure

1. Open the Red Hat Customer Portal in your browser and log in to your Red Hat account.
2. Navigate to **Subscriptions**, which is in the upper-left corner of the Customer Portal page.
3. Click the **Subscription Allocations** tab.
4. Click **New Subscription Allocation**.
5. In the **Name** field, enter a descriptive name that helps identify the subscription allocation.
6. From the **Type** list, select the type and version that corresponds to your Satellite Server.
7. Click **Create**.
8. After a few minutes, the web UI reports that the allocation has been successfully created.

4.1.2. Adding Subscriptions to a Subscription Allocation

If you want to add additional Red Hat products or future-dated subscriptions for existing products, you must add the relevant subscriptions to your subscription allocation. The following procedure explains how to add subscriptions to a subscription allocation.

Procedure

1. Open the Red Hat Customer Portal in your browser and log in to your Red Hat account.
2. Navigate to **Subscriptions**, which is in the upper-left corner of the Customer Portal page.
3. Click the **Subscription Allocations** tab.
4. From the **Name** column, select the name of the subscription allocation you want to modify.
5. Click the **Subscriptions** tab, and then click **Add Subscriptions**.
6. On the row of each subscription you want to add, enter the quantity under the **Entitlements** column.

7. Click **Submit**.

8. If the modified subscription allocation has a corresponding subscription manifest on a Satellite Server, refresh the subscription manifest. For more information, see Section 4.2.5, “Updating and Refreshing Subscription Manifests”.

4.1.3. Removing Subscriptions from a Subscription Allocation

The following procedure explains how to remove subscriptions from a subscription allocation, for example, if you want to remove expired subscriptions.

Procedure

1. Open the Red Hat Customer Portal in your browser and log in to your Red Hat account.

2. Navigate to **Subscriptions**, which is in the upper-left corner of the Customer Portal page.

3. Click the **Subscription Allocations** tab.

4. From the **Name** column, select the name of the subscription allocation you want to modify.

5. Click the **Subscriptions** tab.

6. On the row of each subscription you want to remove, select the corresponding check box.

7. Click **Remove**, and then confirm removal.

8. If the modified subscription allocation has a corresponding subscription manifest on a Satellite Server, refresh the subscription manifest. For more information, see Section 4.2.5, “Updating and Refreshing Subscription Manifests”.

4.1.4. Moving Subscriptions between Subscription Allocations

The following procedure explains how to move subscriptions from one subscription allocation to another.

Procedure

1. Open the Red Hat Customer Portal in your browser and log in to your Red Hat account.

2. Navigate to **Subscriptions**, which is in the upper-left corner of the Customer Portal page.

3. Click the **Subscription Allocations** tab.

4. From the **Name** column, select the name of the subscription allocation you want to modify.

5. Click the **Subscriptions** tab.

6. On the row of each subscription you want to move, select the corresponding check box.

7. Click **Move to Another Allocation**.

8. In the "Move entitlements from this subscription allocation to another" window, use the appropriate method to select the subscriptions you want to move:

    - If you want to select all subscriptions, select **Move All Entitlements**.

- If you want to specify quantities, enter the appropriate quantity on the row of each subscription under the **Quantity to Move** column.
- If you want to remove a subscription from the selection, click the delete icon on the row of that subscription, and then click **OK** to confirm.

9. Click **Next: Select Destination Allocation**.
10. From the table, select the subscription allocation you want to move the subscriptions to, and then click **Submit**.
11. If the modified subscription allocation has a corresponding subscription manifest on a Satellite Server, refresh the subscription manifest. For more information, see Section 4.2.5, "Updating and Refreshing Subscription Manifests".

4.1.5. Using Future Dated subscriptions

You can use future-dated subscriptions in a subscription allocation. When you add future-dated subscriptions to content hosts before the expiry date of the existing subscriptions, you can have uninterrupted access to repositories.

Red Hat recommends that you manually attach the future-dated subscriptions to your content hosts before the current subscriptions expire and not to rely on the auto-attach method. For more information, see Section 4.2.2, "Attaching Subscriptions to Content Hosts".

4.1.6. Using Subscription Allocations to Manage Multiple Organizations

You can manage more than one organization if you have more than one subscription allocation. Satellite 6 requires a single allocation for each organization configured in your Satellite Server. The advantage of this is that each organization maintains completely separate subscriptions so that you can support multiple organizations, each with their own Red Hat accounts.

4.1.7. Exporting a Subscription Manifest from the Customer Portal

When viewing a subscription allocation that has at least one subscription, you can export the allocation as a subscription manifest.

A subscription manifest is a ZIP archive of the subscription allocation, containing encoded subscriptions, which can be imported into your Satellite Server.

Procedure

1. Open https://access.redhat.com/ in your browser and log in to your Red Hat account.
2. Navigate to **Subscriptions**, which is in the upper-left corner of the Customer Portal page.
3. Click the **Subscription Allocations** tab.
4. From the **Name** column, select the name of the subscription allocation you want to export.
5. On the **Details** pane, click **Export Manifest** to download the subscription manifest file.

4.1.8. Importing a Subscription Manifest into Satellite Server

The following procedure explains how to import a Subscription Manifest into your Satellite Server.

Prerequisite

You must have a subscription manifest file previously exported from the Customer Portal. For more information, see Section 4.1.7, “Exporting a Subscription Manifest from the Customer Portal”.

For Web UI Users

1. In the Satellite web UI, ensure the context is set to the organization you want to use.
2. Navigate to **Content** > **Subscriptions**.
3. On the Red Hat Subscriptions page, click **Manage Manifest**.
4. In the Manage Manifest window, click **Browse**.
5. Navigate to the location containing the subscription manifest file, and then click **Open**.
6. If the Manage Manifest window does not close automatically, click **Close** to return to the Red Hat Subscriptions page.

For CLI Users

The Red Hat Satellite 6 CLI requires the manifest to be on your Satellite Server.

1. Log in to your local client system, and then copy the subscription manifest file to your Satellite Server:

```
[user@client ~]$ scp ~/manifest_file.zip
root@satellite.example.com:~/.
```

2. In a terminal, connect to your Satellite Server as the root user, and then import the subscription manifest file:

```
[root@satellite ~]# hammer subscription upload \
--file ~/manifest_file.zip \
--organization "organization_name"
```

After a few minutes, the CLI reports a successful manifest import.

You can now enable repositories and import Red Hat content. For more information, see Chapter 5, *Importing Red Hat Content*.

4.2. MANAGING SUBSCRIPTION ALLOCATIONS IN THE SATELLITE WEB UI

With Red Hat Satellite 6.4, you can now manage your subscription allocations directly in the Satellite web UI.

4.2.1. Searching for a Subscription in the Satellite Web UI

When you import a subscription manifest into your Satellite Server, the subscriptions from your manifest are listed on the Red Hat Subscriptions page. If you have a high volume of subscriptions, you can filter the results to find a specific subscription.

Prerequisite

You must have a subscription manifest file previously imported to your Satellite Server. For more information, see Section 4.1.8, “Importing a Subscription Manifest into Satellite Server”.

Procedure

1. In the Satellite web UI, ensure the context is set to the organization you want to use.
2. Navigate to **Content** > **Subscriptions**.
3. On the Red Hat Subscriptions page, click the **Search** field to view the list search criteria for building your search query.
4. Select a search criteria to display further options.
5. When you have built your search query, click the search icon.

For example, if you place your cursor in the **Search** field, and select **expires**, then press the space bar, another list appears with the options of placing a **>**, **<**, or **=** characters. If you select, **>**, and press the space bar, another list of automatic options appear. You can also enter your own criteria.

4.2.2. Attaching Subscriptions to Content Hosts

Using activation keys is the recommended way to attach subscriptions to content hosts during provisioning. However, an activation key cannot update an existing host. If you need to attach new or additional subscriptions, such as future-dated subscriptions, to one host, follow the method below.

For more information about updating multiple hosts, see Section 4.2.6, “Bulk Updating Content Hosts' Subscriptions”.

For more information about activation keys, see Chapter 9, *Managing Activation Keys*.

Prerequisite

You must have a subscription manifest file previously imported to your Satellite Server. For more information, see Section 4.1.8, “Importing a Subscription Manifest into Satellite Server”.

Smart Management Subscriptions

In Satellite 6, the number of Red Hat Enterprise Linux Smart Management subscriptions correspond with how many systems Satellite can manage. You must maintain a Smart Management subscription for every Red Hat Enterprise Linux subscription that you want to manage with Satellite.

However, you are not required to attach Smart Management subscriptions to each content host. Smart Management subscriptions cannot attach automatically to content hosts in Satellite because they are not associated with any product certificates. Adding a Smart Management subscription to a content host does not provide any content or repository access. If you want, you can add a smart management subscription to a manifest for your own recording or tracking purposes.

For Web UI Users

1. In the Satellite web UI, ensure the context is set to the organization you want to use.
2. Navigate to **Hosts** > **Content Hosts**.
3. On the row of each content host whose subscription you want to change, select the corresponding check box.

4. From the **Select Action** list, select **Manage Subscriptions**.

5. Optionally, enter a key and value in the **Search** field to filter the subscriptions displayed.

6. Select the check box to the left of the subscriptions to be added or removed and click **Add Selected** or **Remove Selected** as required.

7. Click **Done** to save the changes.

For CLI Users

1. In a terminal, connect to your Satellite Server as the root user, and then list the available subscriptions:

```
# hammer subscription list \
--organization-id 1
```

2. Attach a subscription to the host:

```
# hammer host subscription attach \
--host host_name \
--subscription-id subscription_id
```

4.2.3. Adding Subscriptions to Subscription Allocations in the Satellite Web UI

The following procedure explains how to add subscriptions to a subscription allocation in the Satellite web UI.

Prerequisite

You must have a subscription manifest file imported to your Satellite Server. For more information, see Section 4.1.8, “Importing a Subscription Manifest into Satellite Server”.

Procedure

1. In the Satellite web UI, ensure the context is set to the organization you want to use.

2. Navigate to **Content** > **Subscriptions**.

3. On the Red Hat Subscriptions page, click **Add Subscriptions**.

4. On the row of each subscription you want to add, enter the quantity under the **Quantity to Allocate** column.

5. Click **Submit**.

After a few minutes, the web UI reports that the action has been successful.

4.2.4. Removing Subscriptions from Subscription Allocations in the Satellite Web UI

The following procedure explains how to remove subscriptions from a subscription allocation in the Satellite web UI.

NOTE

Manifests must not be deleted. If you delete the manifest from the Red Hat Customer Portal or in the Satellite web UI, all of the entitlements for all of your content hosts will be removed.

Prerequisite

You must have a subscription manifest file imported to your Satellite Server. For more information, see Section 4.1.8, "Importing a Subscription Manifest into Satellite Server".

Procedure

1. In the Satellite web UI, ensure the context is set to the organization you want to use.
2. Navigate to **Content** > **Subscriptions**.
3. On the row of each subscription you want to remove, select the corresponding check box.
4. Click **Delete**, and then confirm deletion.

After a few minutes, the web UI reports that the action has been successful.

4.2.5. Updating and Refreshing Subscription Manifests

Every time that you change a subscription allocation, you must refresh the manifest to reflect these changes. For example, you must refresh the manifest if you take any of the following actions:

- Renewing a subscription
- Adjusting subscription quantities
- Purchasing additional subscriptions

You can use refresh the manifest directly in the Satellite web UI. Alternatively, you can import an updated manifest that contains the changes. For more information about importing a manifest, see Section 4.1.8, "Importing a Subscription Manifest into Satellite Server".

Procedure

1. In the Satellite web UI, ensure the context is set to the organization you want to use.
2. Navigate to **Content** > **Subscriptions**.
3. On the Red Hat Subscriptions page, click **Manage Manifest**.
4. In the Manage Manifest window, click **Refresh**.

After a few minutes, the web UI reports that the action has been successful.

4.2.6. Bulk Updating Content Hosts' Subscriptions

The methods described here are intended for post installation changes to multiple content hosts at the same time. You can use the web UI and the filter function to select the content hosts to be changed, or use the Hammer command-line tool's CSV file export function, edit the configuration settings in the CSV file, and upload the changes.

For Web UI Users

1. In the Satellite web UI, ensure the context is set to the organization you want to use.

2. Navigate to **Hosts** > **Content Hosts**.

3. On the row of each content host whose subscription you want to change, select the corresponding check box.

4. From the **Select Action** list, select **Manage Subscriptions**.

5. Optionally, enter a key and value in the **Search** field to filter the subscriptions displayed.

6. Select the check box to the left of the subscriptions to be added or removed and click **Add Selected** or **Remove Selected** as required.

7. Click **Done** to save the changes.

For CLI Users

1. Export the current state of content hosts to a CSV file.

   ```
   # hammer --server https://satellite.example.com csv content-hosts --
   export --file content_hosts.csv
   ```

2. Change the required values in the CSV file. You can use an editor, with a CSV plug-in, or **sed** to change strings in the CSV file.

   a. Make a backup of the file:

      ```
      # cp content_hosts.csv content_hosts.csv.backup
      ```

   b. Edit the string you want to change. For example:

      ```
      # sed -i "s/1|RH1234|Red Hat Enterprise Linux Server/1|RH5678|Red
      Hat Enterprise Linux Server/g" content_hosts.csv
      ```

   c. Confirm only the required changes were made. For example:

      ```
      # diff content_hosts.csv content_hosts.csv.backup
      ```

3. Upload the changed file to Satellite Server:

   ```
   # hammer --server https://satellite.example.com csv content-hosts --
   file content_hosts.csv
   ```

CHAPTER 5. IMPORTING RED HAT CONTENT

Our Satellite Server now has the necessary subscription information imported. We are ready to add content to our system. For this chapter, we explore the concept of a Definitive Media Library (DML) and how to synchronize content to create our DML.

5.1. CREATING A DEFINITIVE MEDIA LIBRARY

A DML is a repository that stores and protects the definitive, authorized versions of software and configurations. In other words, the DML acts as a master version of any content imported into Satellite. This includes Red Hat content, such as RPM files, kickstart trees, and ISO images. As mentioned in Section 1.1, "Overview of Red Hat Satellite 6 Content Management", Red Hat Satellite 6 stores and manages content in a DML.

5.2. USING PRODUCTS AND REPOSITORIES IN SATELLITE

In Satellite, we use the concept of a **Product** as an organizational unit to group multiple repositories together. Such repository collections are analogous to the concept of real life products. For example, if we view Red Hat Enterprise Linux Server as a Product in Satellite, the repositories for that product might consist of different versions (6.0, 6.1, 7.0), different architectures (i386, x86_64, s390x, arm), and different add-ons (Optional repositories, Supplementary repositories, Virt V2V tools). This unifies all related repositories within the DML. Using Products ensures repositories that depend on each other are synchronized together. For Red Hat repositories, products are created automatically after enabling the repository.

In this chapter, our aim is to start creating our DML using Red Hat content. To do this, we synchronize our DML with Red Hat's Products and their repositories.

5.3. SYNCHRONIZING CONTENT

When you select which repositories form the DML, Satellite Server synchronizes its own repositories with the repositories on the Red Hat CDN. This ensures that Satellite Server retains an exact copy of Red Hat's repositories as a part of its DML. Satellite Server fetches this repository information and stores it on Satellite Server's file system. After an initial synchronization, you can create a synchronization plan that checks to ensure the repositories in the DML is up to date with the CDN's repositories.

It is possible to perform an initial synchronization using ISO images. See Appendix C, *Importing Content ISOs into a Connected Satellite* for more information on using Content ISOs. For locations with bandwidth limitations, using an **On Demand** or **Background** download policy as described below might be quicker than downloading and importing Content ISOs.

5.4. USING DOWNLOAD POLICIES

Red Hat Satellite provides multiple download policies for synchronizing RPM content. For example, you might aim to save time and only download the content metadata while deferring the actual content download for later.

Satellite Server offers the following policies:

- **Immediate** — Satellite Server downloads all metadata and packages during synchronization.

- **On Demand** — Satellite Server only downloads the metadata during synchronization. Satellite Server only fetches and stores packages on the file system when Capsules or directly connected clients request them. This setting has no effect if you set a corresponding repository on a

Capsule to **Immediate** because Satellite Server is forced to download all the packages.

- **Background** — Satellite Server creates a background task to download all packages after the initial synchronization.

The latter two policies act as a *Lazy Synchronization* feature because they save time synchronizing content. The lazy sync feature must only be used for **`yum`** repositories. The packages can be added to Content Views and promoted to life cycle environments as normal.

Capsule Server offers the following policies:

- **Immediate** — Capsule Server downloads all metadata and packages during synchronization. Do not use this setting if the corresponding repository on Satellite Server is set to **On Demand** as Satellite Server is forced to download all the packages.

- **On Demand** — Capsule Server only downloads the metadata during synchronization. The Capsule only fetches and stores packages on the file system when directly connected clients request them.

  NOTE

  When using an **On Demand** download policy, content is downloaded from Satellite Server if it is not available on the Capsule Server.

- **Background** — Capsule Server creates a background task to download all packages after the initial synchronization.

- **Inherit** — Capsule Server inherits the download policy for the repository from the corresponding repository on Satellite Server.

These policies are not available if a Capsule was installed or updated with **`--enable-foreman-proxy-plugin-pulp`** set to false.

5.5. SELECTING RED HAT REPOSITORIES TO SYNCHRONIZE

To select the repositories to synchronize you must first identify the product that contains the repository, and then enable that repository based on the relevant release version and base architecture.

IMPORTANT

If using a Disconnected Satellite, you need to import the Content ISOs for Red Hat Satellite and change the CDN URL on Satellite Server before synchronizing content. For more information, see Appendix B, *Importing Content ISOs into a Disconnected Satellite*.

For Web UI Users

1. Navigate to **Content** > **Red Hat Repositories**.
   The Red Hat Repositories page displays the following two panes: the Available Repositories and the Enabled Repositories panes. The Available Repositories pane contains a list of all subscribed products that provide RPM content. The Enabled Repositories pane contains a list of enabled repositories.

   The relationship between products and specific repositories is connected through a cascading hierarchy that is displayed in the Available Repositories pane.

2. To find the **Red Hat Enterprise Linux 7 Server (RPMs)** repositories, use one of the following methods:

    - In the **Search** field, enter the following repository name: **`rhel-7-server-rpms`**.
    - Toggle the **Recommended Repositories** button to the ON position to view a list of repositories that are used in a typical setup.

3. In the Available Repositories pane, click on the **Red Hat Enterprise Linux 7 Server (RPMs)** to expand the repository set.

4. For the **`x86_64 7Server`** entry, click the **Enable** icon to enable the repository. This enables the latest RPM files for Red Hat Enterprise Linux 7.

NOTE

The difference between associating Red Hat Enterprise Linux Operating System with either 7 Server repositories or 7.X repositories is that 7 Server repositories contain all the latest updates while Red Hat Enterprise Linux 7.X repositories stop getting updates after the next minor version release. Note that Kickstart repositories only have minor versions.

For CLI Users

The relationship between products and repositories is the same. To search for your product, enter the following command:

```
# hammer product list --organization "ACME"
```

List the repository set for the product:

```
# hammer repository-set list \
--product "Red Hat Enterprise Linux Server" \
--organization "ACME"
```

This displays the repositories in the product’s repository set, including their name and ID number. Enable the repository using either the name or ID number. Also include the release version (**`7Server`**) and base architecture (**`x86_64`**). For example:

```
# hammer repository-set enable \
--name "Red Hat Enterprise Linux 7 Server (RPMs)" \
--releasever "7Server" \
--basearch "x86_64" \
--product "Red Hat Enterprise Linux Server" \
--organization "ACME"
```

For this scenario, use either the Web UI or the CLI to enable the following repositories for ACME:

Repository	Type	Description

Repository	Type	Description
Red Hat Enterprise Linux 7 Server RPMs x86_64 7Server	RPM	Repositories for the latest version of Red Hat Enterprise Linux 7. Use the `7 Server` repository instead of the `7.2` repository so that you receive continuous package updates.
Red Hat Satellite Tools 6.4 for RHEL 7 Server RPMs x86_64	RPM	The Satellite Tools repository, which contains system management agents and tools for client systems. After provisioning a new system, Satellite installs tools such as `katello-agent` and Puppet to the client. Use the `7 Server` repository instead of the `7.2` repository so that you receive continuous package updates.
Red Hat Enterprise Linux 7.2 Kickstart x86_64 7Server	Kickstart	The kickstart tree for Red Hat Enterprise Linux 7.2. Use this as installation media for provisioning new systems over PXE.

These repositories provide some initial content for our scenario's DML. You can select more repositories based upon your needs.

NOTE

For this scenario, all repositories use `x86_64` as the base architecture.

5.6. SYNCHRONIZING RED HAT REPOSITORIES

We have enabled specific repositories to form our initial DML. Now we synchronize the repositories with the Red Hat CDN's repositories.

For Web UI Users

Navigate to **Content** > **Products** and select **Red Hat Enterprise Linux Server**. This displays all enabled repositories in our product. Select all repositories and click **Sync Now**. You can also view the progress of the synchronization in the Web UI. Navigate to **Content** > **Sync Status** and expanding the Product/Repository tree (or click **Expand All**).

For CLI Users

Synchronize the enabled repositories in the Red Hat Enterprise Linux Server product:

```
# hammer product synchronize \
--name "Red Hat Enterprise Linux Server" \
--organization "ACME"
```

You can also synchronize each repository individually. List all repositories in the product, then synchronize using the ID number for the corresponding repositories. For example:

```
# hammer repository list \
--product "Red Hat Enterprise Linux Server" \
--organization "ACME"
# hammer repository synchronize \
--name "Red Hat Enterprise Linux 7 Server RPMs x86_64 7Server" \
--product "Red Hat Enterprise Linux Server" \
--organization "ACME"
```

The synchronization duration depends on the size of each repository and the speed of your network connection. The following table provides estimates of how long it would take to synchronize content, depending on the available Internet bandwidth:

	Single Package (10Mb)	Minor Release (750Mb)	Major Release (6Gb)
256 Kbps	5 Mins 27 Secs	6 Hrs 49 Mins 36 Secs	2 Days 7 Hrs 55 Mins
512 Kbps	2 Mins 43.84 Secs	3 Hrs 24 Mins 48 Secs	1 Day 3 Hrs 57 Mins
T1 (1.5 Mbps)	54.33 Secs	1 Hr 7 Mins 54.78 Secs	9 Hrs 16 Mins 20.57 Secs
10 Mbps	8.39 Secs	10 Mins 29.15 Secs	1 Hr 25 Mins 53.96 Secs
100 Mbps	0.84 Secs	1 Min 2.91 Secs	8 Mins 35.4 Secs
1000 Mbps	0.08 Secs	6.29 Secs	51.54 Secs

A manual synchronization is often required for the initial content import into the DML. However, it is recommended to create a synchronization plan to ensure that our DML updates on a regular basis.

NOTE

You can change the download policy for a Red Hat repository. Select a repository in the **`Red Hat Enterprise Linux Server`** product and navigate to the **Download Policy** field. If using the CLI, enter the **`hammer repository update`** command with the **`--download-policy`** option.

5.6.1. Recovering a Repository

In the case of repository corruption, you can recover it by using an advanced synchronization, which has three options:

- **Optimized Sync** - it synchronizes the repository bypassing RPMs that have no detected differences from the upstream RPMs.

- **Complete Sync** - it synchronizes all RPMs regardless of detected changes. Use this option if specific RPMs could not be downloaded to the local repository even though they exist in the upstream repository.
- **Validate Content Sync** - it synchronizes all RPMs and then verifies the checksum of all RPMs locally. If the checksum of an RPM differs from the upstream, it re-downloads the RPM. This option is relevant for **`yum`** repositories only. Use this option if you have one of the following errors:
  - Specific RPMs cause a **`404`** error while synchronizing with **`yum`**.
  - **`Package does not match intended download`** error, which means that specific RPMs are corrupted.

To synchronize a specific repository with an advanced option:

For Web UI users

1. Navigate to **Content** > **Products**.
2. Select the product containing the corrupted repository.
3. Click on the name of a repository you want to synchronize.
4. Extend the **Select Action** menu and select **Advanced Sync**.
5. Select the option and click **Sync**.

For CLI users

1. Obtain a list of repository IDs:

```
# hammer repository list --organization "Default Organization"
```

2. Synchronize a corrupted repository using the necessary option:
   - For the optimized synchronization:

   ```
   # hammer repository synchronize --incremental true --id 1
   ```

   - For the complete synchronization:

   ```
   # hammer repository synchronize --skip-metadata-check true --id 1
   ```

   - For the validate content synchronization:

   ```
   # hammer repository synchronize --validate-contents true --id 1
   ```

5.6.2. Limiting Synchronization Speed

You can control the speed of synchronization to avoid exhaustion of available bandwidth and prevent other performance issues. This is done by configuring **PULP_CONCURRENCY** and **max_speed** parameters. Note that these settings are overwritten on an upgrade. It is recommended to backup changed files prior to an upgrade to be able to restore the configuration.

1. To control the number of synchronization jobs that run in parallel, configure the **PULP_CONCURRENCY** parameter in the **/etc/default/pulp_workers** file. For example, to set the number of jobs that run in parallel to 1, change **PULP_CONCURRENCY** to 1:

```
PULP_CONCURRENCY=1
```

   By default, on a system with less than 8 CPUs, **PULP_CONCURRENCY** is set to the number of CPUs. On a system with more than 8 CPUs, it is set to 8.

2. To set the maximum network speed for synchronizing in bytes per second, configure the **max_speed** parameter. This parameter must be configured separately for each importer in the **/etc/pulp/server/plugins.conf.d/** directory.

   a. For example, to set the maximum speed for synchronizing RPM content to 10 bytes per second, set the **"max_speed"** parameter in the **/etc/pulp/server/plugins.conf.d/yum_importer.json** file to 10:

```
# cat /etc/pulp/server/plugins.conf.d/yum_importer.json
{
    "proxy_host": null,
    "proxy_port": null,
    "proxy_username": null,
    "proxy_password": null,
    "max_speed": 10
}
```

   b. Verify the syntax of the file after editing:

```
# json_verify < /etc/pulp/server/plugins.conf.d/yum_importer.json
JSON is valid
```

3. Restart Satellite services to apply the changes:

```
# foreman-maintain service restart
```

5.7. CREATING A SYNCHRONIZATION PLAN

A synchronization plan checks and updates the content in your DML at a regularly scheduled date and time. Red Hat Satellite 6 provides users with the ability to create a synchronization plan and assign products to it.

For Web UI Users

Navigate to **Content** > **Sync Plans** and click **New Sync Plan**. The UI provides a set of fields where you can input details about your synchronization plan:

- **Name** - A plain text name for the plan. Enter **Example Plan**.
- **Description** - A plain text description of the plan. Enter **Example Plan for ACME's repositories**.
- **Interval** - Defines when to run the synchronization. Select **daily**.

- **Start Date** and **Start Time** - Defines when to run the synchronization. We already completed a synchronization today, so set the synchronization for tomorrow at 1:00 (1AM).

Click **Save** to create a plan. The plan details page displays along with two tabs for **Details** and **Products**.

Now add your products. Click the **Products** tab, then click **Add**. Select the **Red Hat Enterprise Linux Server** product and click **Add Selected**.

For CLI Users

To create the synchronization plan, enter the following command:

```
# hammer sync-plan create \
--name "Red Hat Products 2" \
--description "Example Plan for ACME's Red Hat Products" \
--interval daily \
--sync-date "2016-02-01 01:00:00" \
--enabled true \
--organization "ACME"
```

Then assign the Red Hat Enterprise Linux Server product to it:

```
# hammer product set-sync-plan \
--name "Red Hat Enterprise Linux Server" \
--sync-plan "Red Hat Products" \
--organization "ACME"
```

Now Satellite Server checks its DML content against the Red Hat CDN on a daily basis and keeps its Red Hat repositories up to date.

CHAPTER 6. IMPORTING CUSTOM CONTENT

The previous chapter examined how to import Red Hat content into the Definitive Media Library (DML). This chapter focuses on custom content, which differs slightly from Red Hat content. We create our own product beforehand, customize it, and add our own repositories. In addition, you can add Puppet modules to a custom repository.

6.1. USING CUSTOM PRODUCTS IN SATELLITE

In Section 5.2, “Using Products and Repositories in Satellite”, we explored the concept of a Product in Red Hat Satellite 6 and how it is used to group repositories together. Red Hat Satellite 6 also provides the ability to create custom Products so you can add multiple related repositories. Both Red Hat content and custom content in Red Hat Satellite 6 share some similarities:

- The relationship between a product and its repositories is the same and the repositories still require synchronization.
- Custom Products require a subscription for clients to access, similar to subscriptions to Red Hat Products. Red Hat Satellite 6 creates a new subscription for each custom Product you create.

In this chapter, we aim to create a product that contains two related repositories: an RPM repository containing RPM content and a Puppet repository for modules to configure the RPM content.

For more information about creating and packaging RPMs, see the RPM Packaging Guide in the Red Hat Enterprise Linux documentation.

6.2. CREATING A CUSTOM PRODUCT

In our scenario, ACME is aiming to develop a product called *Exampleware*, which is a web-based application that requires a PostgreSQL database. ACME might aim to use production-level Exampleware in the field using PostgreSQL from the Red Hat Enterprise Linux repositories but test Exampleware with newer versions of PostgreSQL. For this situation, let’s create a custom product for PostgreSQL so that we can synchronize the newer versions.

For Web UI Users

Navigate to **Content** > **Products**, click **New Product** and enter the following details:

- **Name** - The plain text name for the product. Enter **`PostgreSQL`**.
- **Label** - An internal ID for the product. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **GPG Key** - The GPG Key for the entire product. Leave this blank as we import a GPG for a specific version of PostgreSQL and attach it to the repository instead of the product.
- **Sync Plan** - A synchronization plan for the product. We can attach this to the **`Example Plan`** we created in the previous chapter.
- **Description** - A plain text description of the product. Enter **`Content from PostgreSQL repositories`**.

When you have entered this information, click **Save**.

For CLI Users

To create the product, enter the following command:

```
# hammer product create \
--name "PostgreSQL" \
--sync-plan "Example Plan" \
--description "Content from PostgreSQL repositories" \
--organization "ACME"
```

This creates a new product where we can create our own custom repositories and synchronize their content.

6.3. IMPORTING A CUSTOM GPG KEY

Before we create a custom product, we might need to create a custom GPG key. This is used to provide a certain level of security for RPM transactions with the PostgreSQL repository.

First download a copy of the version specific repository package to your client system. In our case, we download **`pgdg-redhat95`**:

```
[user@client ~]$ wget http://yum.postgresql.org/9.5/redhat/rhel-7-
x86_64/pgdg-redhat95-9.5-2.noarch.rpm
```

Extract the RPM file without installing it:

```
[user@client ~]$ rpm2cpio pgdg-redhat95-9.5-2.noarch.rpm | cpio -idmv
```

The GPG key is located relative to the extraction at **`etc/pki/rpm-gpg/RPM-GPG-KEY-PGDG-95`**.

For Web UI Users

To import a GPG key, complete the following procedure:

1. In the Satellite web UI, navigate to **Content** > **Content Credentials** and in the upper-right of the window, click **Create Content Credential**.
2. Enter the name **`PostgreSQL 9.5`** and select **GPG Key** from the **Type** list.
3. Either paste the GPG key into the **Content Credential Contents** field, or click **Browse** and select the GPG key file that you want to import.
4. Click **Save**.

For CLI Users

Copy the GPG key to your Satellite Server:

```
[user@client ~]$ scp ~/etc/pki/rpm-gpg/RPM-GPG-KEY-PGDG-95
root@satellite.example.com:~/.
```

Upload the GPG key to Satellite:

```
[root@satellite ~]# hammer gpg create \
--key ~/RPM-GPG-KEY-PGDG-95 \
--name "PostgreSQL 9.5" \
--organization "ACME"
```

Now we have a GPG key we can associate with our repository.

6.4. CREATING A CUSTOM RPM REPOSITORY

Normally, a production-level server would use the version of PostgreSQL included with Red Hat Enterprise Linux for stability purposes. However, ACME's developers might aim to test Exampleware with a more recent version of PostgreSQL. In this instance, they can create a custom repository for a newer version of PostgreSQL.

For Web UI Users

We follow on from creating our custom PostgreSQL product. After creating our custom product, the repositories screen appears. Click **Create Repository**, which displays a form for a new repository. Enter the following details:

- **Name** - A plain text name for the repository. Enter **`PostgreSQL 9.5`**.
- **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **Type** - The type of repository. You can select either a repository for RPM files (**`yum`**), Puppet modules (**`puppet`**), or Docker images (**`docker`**). For our scenario, select **`yum`**. A new set of fields appear.
- **URL** - The URL of the external repository to use as a source. Enter **`http://yum.postgresql.org/9.5/redhat/rhel-7-x86_64/`**.
- **Download Policy** - Determines the type of synchronization Satellite Server performs. Select **`Immediate`**. See Section 5.4, "Using Download Policies" for more information.
- **Mirror on Sync** - Ensures the content that is no longer part of the upstream repository is removed during synchronization. Leave this checked, which is the default.
- **Checksum** - The checksum for the repository. For this example, leave this as the **`Default`**, which defaults to SHA256. This is the checksum required for Red Hat Enterprise Linux 7. For Red Hat Enterprise Linux 5 or previous versions, select SHA1 for the checksum.
- **Publish via HTTP** - Enables this repository for publication through HTTP. This option is automatically selected.
- **GPG Key** - The GPG Key for this repository. Select the PostgreSQL 9.5 GPG key we created previously.

Click **Save** to save this repository entry.

The repository uses the synchronization plan to periodically keep it up to date. However, let's perform an initial synchronization. Select the PostgreSQL 9.5 repository and click **Sync Now**. This starts a task to synchronize our repository with the external PostgreSQL repository.

NOTE

Monitor the progress of this synchronization on the **Content** > **Sync Status** page.

For CLI Users

Enter the following command to create the repository:

```
# hammer repository create \
--name "PostgreSQL 9.5" \
--content-type "yum" \
--publish-via-http true \
--url http://yum.postgresql.org/9.5/redhat/rhel-7-x86_64/ \
--gpg-key "PostgreSQL 9.5" \
--product "PostgreSQL" \
--organization "ACME"
```

Then synchronize the repository:

```
# hammer repository synchronize \
--name "PostgreSQL 9.5" \
--product "PostgreSQL" \
--organization "ACME"
```

Now we have a synchronized copy of PostgreSQL 9.5. Let’s take this a step further and include a Puppet module to configure a PostgreSQL server.

NOTE

The Products page in the Web UI also provides a **Repo Discovery** function that finds all repositories from a URL and allows you to select which ones to add to your custom Product. For example, you can use the **Repo Discovery** to search `http://yum.postgresql.org/9.5/redhat/` and list all PostgreSQL 9.5 repositories for different Red Hat Enterprise Linux versions and architectures. This helps users save time importing multiple repositories from a single source.

IMPORTANT

Red Hat does not support the upstream RPMs directly from the PostgreSQL site. These RPMs are used to demonstrate the synchronization process. For any issues with these RPMs, contact the developers for PostgreSQL.

6.5. CREATING A CUSTOM PUPPET REPOSITORY

Custom products can also include repositories for Puppet modules. This provides a method to incorporate state configuration of hosts.

First, let’s create a repository for our PostgreSQL product.

For Web UI Users

Ensure you are on the Product page (**Content** > **Products**). Click on the PostgreSQL product, which displays our repository listing. Click **Create Repository**, which displays a form for a new repository. Enter the following details:

- **Name** - A plain text name for the repository. Enter **`PostgreSQL Puppet Modules`**.
- **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **Type** - The type of repository. Select **`puppet`** and a URL field appears.

- **URL** - The URL of the external repository to use as a source. We manually import our Puppet module, but you can use a repository source to synchronize your own Puppet modules.

Click **Save** to save this repository entry.

For CLI Users

Enter the following command to create our Puppet module repository:

```
# hammer repository create \
--name "PostgreSQL Puppet Modules" \
--content-type "puppet" \
--product "PostgreSQL" \
--organization "ACME"
```

Now we have a custom repository for our Puppet modules. Let’s add one now.

6.6. MANAGING INDIVIDUAL PUPPET MODULES

For this scenario, we manually import a Puppet module for configuring PostgreSQL.

Download this module from the Puppet Forge site (https://forge.puppetlabs.com/puppetlabs/postgresql). Open the page in your browser and click **download latest tar.gz** to save to your local file system.

For Web UI Users

Ensure you are on the repository listing page for the PostgreSQL product. Click on the PostgreSQL Puppet Modules repository, which displays the details page for that repository.

Navigate to the **Upload Puppet Module** section, click **Browse**, select the PostgreSQL Puppet Module we downloaded previously, and click Upload. After a few seconds, Satellite Server reports **`Content successfully uploaded`**.

NOTE

Click on the **Manage Puppet Modules** page to manage and remove Puppet modules from a product.

For CLI Users

Copy the Puppet module to your Satellite Server’s file system:

```
[user@client ~]$ scp ~/puppet_module.tar.gz root@satellite.example.com:~/.
```

Import the Puppet module to the PostgreSQL Puppet Modules repository:

```
[root@satellite ~]# hammer repository upload-content \
--path ~/puppet_module.tar.gz \
--name "PostgreSQL Puppet Modules" \
--product "PostgreSQL" \
--organization "ACME"
```

Now we have a custom repository that contains both RPM content and a Puppet module to install and configure a server using the RPM content.

IMPORTANT

Red Hat does not support the modules from Puppet Forge. The PostgreSQL module is used to demonstrate the module management process. For any issues with these modules, contact the module developer.

6.7. SYNCHRONIZING PUPPET REPOSITORIES

In addition to creating a repository of uploaded Puppet modules, Satellite Server can synchronize a complete Puppet module repository. In this example, Satellite Server synchronizes the entire Puppet Forge repository.

For Web UI Users

Navigate to **Content** > **Products** and click **New Product**. A form for a new Product appears. Enter the following details:

- **Name** - The plain text name for the product. Enter **`Puppet Forge`**.
- **Label** - An internal ID for the product. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **GPG Key** - The GPG Key for the entire product. Leave this blank.
- **Sync Plan** - A synchronization plan for the product. We can attach this to the **`Example Plan`** we created in the previous chapter.
- **Description** - A plain text description of the product. Enter **`All modules from Puppet Forge`**.

When you have entered this information, click **Save**.

After creating our custom product, the repositories screen appears. Click **Create Repository**, which displays a form for a new repository. Enter the following details:

- **Name** - A plain text name for the repository. Enter **`Puppet Forge Modules`**.
- **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **Type** - The type of repository. Select **`puppet`** and a URL field appears.
- **URL** - The URL of the external repository to use as a source. Enter **`http://forge.puppetlabs.com/`**.

Click **Save** to save this repository entry.

Select the **`Puppet Forge Modules`** repository and click **Sync Now**. This imports all modules from Puppet Forge into Satellite Server.

For CLI Users

Create the product:

```
# hammer product create \
--name "Puppet Forge" \
--sync-plan "Example Plan" \
```

```
--description "All modules from Puppet Forge" \
--organization "ACME"
```

Create the Puppet Forge repository:

```
# hammer repository create \
--name "Puppet Forge Modules" \
--content-type "puppet" \
--product "Puppet Forge" \
--organization "ACME" \
--url http://forge.puppetlabs.com/
```

Synchronize the repository:

```
# hammer repository synchronize \
--name "Puppet Forge Modules" \
--product "Puppet Forge" \
--organization "ACME"
```

NOTE

The Puppet Forge repository contains several thousand modules and can take a long time to synchronize.

IMPORTANT

Red Hat does not support the modules from Puppet Forge. The modules are used to demonstrate the synchronization process. For any issues with these modules, contact the module developer.

6.8. SYNCHRONIZING PUPPET MODULES FROM A GIT REPOSITORY

Red Hat Satellite 6 includes a utility called **pulp-puppet-module-builder**, which you can install on other systems from the **pulp-puppet-tools** RPM. This tool checks out a Git repository, builds all the modules, and publishes them in a structure that Satellite 6 can synchronize. One common method is to run the utility on Satellite Server itself, publish to a local directory, and synchronize against that directory. For example:

```
# mkdir /modules
# chmod 755 /modules
# pulp-puppet-module-builder \
--output-dir=/modules \
--url=git@mygitserver.com:mymodules.git \
--branch=develop
```

This checks out the **develop** branch of the Git repository from **git@mygitserver.com:mymodules.git** and publishes it to **/modules**. Add this directory as the URL (**file:///modules**) for a new repository on Satellite Server. For example:

For Web UI Users

Open a custom product (in this case, we use **MyProduct** as an example) and click **Create Repository**. Enter the following details:

- **Name** - A plain text name for the repository. Enter **`Modules from Git`**.
- **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **Type** - The type of repository. Select **`puppet`** and a URL field appears.
- **URL** - The URL of the external repository to use as a source. Enter **`file:///modules`**.

For CLI Users

Create the Puppet Forge repository:

```
# hammer repository create \
--name "Modules from Git" \
--content-type "puppet" \
--product "MyProduct" \
--organization "ACME" \
--url file:///modules
```

NOTE

The same process also applies to publishing modules on a remote HTTP server. For example, if you use **`webserver.example.com`** as a standard web host to publish the Puppet modules, log into the host and enter the following commands:

```
# mkdir /var/www/html/modules/
# chmod 755 /var/www/html/modules/
# pulp-puppet-module-builder \
--output-dir=/var/www/html/modules/ \
--url=git@mygitserver.com:mymodules.git \
--branch=develop
```

On Satellite Server, set the repository's URL to **`http://webserver.example.com/modules/`**.

6.9. CREATING A CUSTOM FILE TYPE REPOSITORY IN RED HAT SATELLITE

Custom products in Red Hat Satellite can also include repositories for custom file types. This provides a generic method to incorporate arbitrary files in a Product. Applications range from distributing SSH keys and source code files to larger files such as virtual machine images and ISO files.

You can upload files to the repository or synchronize it from an upstream Satellite Server. When you add files to a custom file type repository, you can use the normal Satellite management functions such as adding a specific version to a Content View to provide version control and making the repository of files available on various Capsule Servers. Clients must download the files over HTTP or HTTPS using **`curl -O`**.

A file type repository in Satellite Server can be created only in a custom product, but you can create an independent file type repository in a directory on the system where Satellite is installed, or on a remote HTTP server, and then synchronize the contents of that directory into Satellite. This method is useful when you have multiple files to add to a Satellite repository.

This example creates a Product *My File Product*, and then creates a file type repository *My Files* within that product. For more information about file type repositories outside of Satellite, see Appendix E, *Creating a Remote File Type Repository*.

For Web UI Users

1. Create a Custom Product
   Navigate to **Content** > **Products**, click **Create Product** and enter the following details:
   - **Name** - The plain text name for the product. Enter **`My File Product`**.
   - **Label** - An internal ID for the product. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
   - **GPG Key** - The GPG Key for the entire product. You can leave this blank.
   - **Sync Plan** - A synchronization plan for the product. For example, the **`Example Plan`** created in the previous chapter.
   - **Description** - A plain text description of the product. Enter **`My files`**.
2. Create a File Type Repository
   Navigate to **Content** > **Products**. Click the name of a product, in this example *My File Product*. On the **Repositories** tab, click **New Repository** and enter the following details:
   - **Name** - A plain text name for the repository. Enter **`My Files`**.
   - **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
   - **Type** - The content type of the repository. Select **`file`**.
   - **Upstream URL** - The URL of the upstream repository to use as a source. You can leave this blank if you plan to only upload files manually.
   - **Verify SSL** - Leave this option selected if you want to verify that the upstream repository's SSL certificates are signed by a trusted CA. Alternatively, clear this check box.
   - **Upstream Username** - If required for authentication, enter the user name for the upstream repository. Clear this field if the repository does not require authentication.
   - **Upstream Password** - Enter the password for the upstream repository. Clear this field if the repository does not require authentication.
3. Click **Save** to save this repository entry.

For CLI Users

1. Create a Custom Product

```
# hammer product create \
--name "My File Product" \
--sync-plan "Example Plan" \
--description "My files" \
--organization "ACME"
```

Optional Parameters

gpg-key *gpg_key_name*	Key name to search by
gpg-key-id *gpg_key_id*	GPG key numeric identifier
sync-plan *sync_plan_name*	Sync plan name to search by
sync-plan-id *sync_plan_id*	Sync plan numeric identifier

2. Create a File Type Repository

```
# hammer repository create \
--name "My Files" \
--content-type "file" \
--product "My File Product" \
--organization "ACME"
```

Optional Parameters

checksum-type *sha_version*	Repository checksum, currently 'sha1' & 'sha256' are supported
download-policy *policy_name*	Download policy for yum repos (either 'immediate', 'on_demand', or 'background').
gpg-key *gpg_key_name*	Key name to search by
gpg-key-id *gpg_key_id*	GPG key numeric identifier
mirror-on-sync *boolean*	Must this repo be mirrored from the source, and stale RPMs removed, when synced? Set to **true** or **false**, **yes** or **no**, **1** or **0**.
publish-via-http *boolean*	Must this also be published using HTTP? Set to **true** or **false**, **yes** or **no**, **1** or **0**.
upstream-username *repository_username*	Upstream repository user, if required for authentication
upstream-password *repository_password*	Password for the upstream repository user
url *source_repo_url*	URL of the Source repository
verify-ssl-on-sync *boolean*	Must Katello verify that the upstream URL’s SSL certificates are signed by a trusted CA? Set to **true** or **false**, **yes** or **no**, **1** or **0**.

6.10. UPLOADING FILES TO A CUSTOM FILE TYPE REPOSITORY IN RED HAT SATELLITE

For Web UI Users

1. Navigate to **Content** > **Products**.
2. Select a custom product by name. For example, **`My File Product`**.
3. Select a file type repository by name. For example, **`My Files`**.
4. Click **Browse** to search and select the file you want to upload.
5. Click **Upload** to upload the selected file to Satellite Server.
6. Visit the URL where the repository is published to see the file.

For CLI Users

```
# hammer repository upload-content \
--product "My File Product" \
--name "My Files" \
--organization "ACME" \
--path example_file
```

The **`--path`** option can indicate a file, a directory of files, or a glob expression of files. Globs must be escaped by single or double quotes.

6.11. DOWNLOADING FILES TO A HOST FROM A CUSTOM FILE TYPE REPOSITORY IN RED HAT SATELLITE

You can download files to a client over HTTPS using **`curl -O`**, and optionally over HTTP if the **Publish via HTTP** repository option is selected.

Prerequisites

- You have a custom file type repository. See Section 6.9, "Creating a Custom File Type Repository in Red Hat Satellite" for more information.
- You know the name of the file you want to download to clients from the file type repository.
- To use HTTPS you need the following certificates on the client:
    1. The **`katello-server-ca.crt`**. Ensure you obtain this in a secure way. You can export it from your browser or download it over HTTPS from **`https://satellite.example.com/pub/`**.
    2. An Organization Debug Certificate. See Section 2.3, "Creating an Organization Debug Certificate" for more information.

For Web UI Users

1. Navigate to **Content** > **Products**.

2. Select a custom product by name. For example, *My File Product*.

3. Select a file type repository by name. For example, *My Files*.

4. Check to see if **Publish via HTTP** is enabled. If it is not, you need the certificates to use HTTPS.

5. Copy the URL where the repository is published.

For CLI Users

1. List the file type repositories.

```
# hammer repository list --content-type file
---|----------|-----------------|--------------|----
ID | NAME     | PRODUCT         | CONTENT TYPE | URL
---|----------|-----------------|--------------|----
7  | My Files | My File Product | file         |
---|----------|-----------------|--------------|----
```

2. Display the repository information.

```
# hammer repository info --name "My Files" --product "My File
Product" --organization-id 1
```

- If HTTP is enabled, the output looks something like this:

```
Publish Via HTTP:   yes
Published At:       http://satellite.example.com/pulp/isos/uuid/
```

- If HTTP is not enabled, the output looks something like this:

```
Publish Via HTTP:   no
Published At:       https://satellite.example.com/pulp/isos/uuid/
```

On the client, enter a command in the appropriate format for HTTP or HTTPS:

- for HTTP:

```
# curl -O satellite.example.com/pulp/isos/uuid/my_file
```

- for HTTPS:

```
# curl -O --cert ./Default\ Organization-key-cert.pem  --cacert
katello-server-ca.crt satellite.example.com/pulp/isos/uuid/my_file
```

6.12. CREATING A CUSTOM FILE TYPE REPOSITORY IN A LOCAL DIRECTORY

You can create a custom file type repository, from a directory of files, external to Satellite Server using the **`pulp-manifest`** command. You can then synchronize the files into Satellite Server. When you add files to a file type repository, you can work with the files as with any other repository.

This procedure describes configuring a repository in a directory on the base system where Satellite is installed. To create a file type repository in a directory on a remote server, see Appendix E, *Creating a Remote File Type Repository*.

To Create a File Type Repository in a Local Directory:

1. Ensure the Server and Satellite Tools repositories are enabled:

```
# subscription-manager repos --enable=rhel-7-server-rpms \
--enable=rhel-7-server-satellite-tools-6.4-rpms
```

2. Install the Pulp Manifest package:

```
# yum install python-pulp-manifest
```

3. Create a directory that you want to use as the file type repository in the HTTP server's public folder:

```
# mkdir my_file_repo
```

4. Add files to the directory or create a test file:

```
# touch my_file_repo/test.txt
```

5. Enter the Pulp Manifest command to create the manifest:

```
# pulp-manifest my_file_repo
```

6. Verify the manifest was created:

```
# ls my_file_repo
PULP_MANIFEST  test.txt
```

To Import Files From a File Type Repo in a Local Directory

1. Ensure a custom product exists in Satellite Server. This example uses *My File Product*.

2. Create a File Type Repository:

   a. In the Satellite web UI, navigate to **Content** > **Products**.

   b. Select the name of a product, in this example *My File Product*.

   c. On the **Repositories** tab, select **New Repository** and enter the following details:

      - **Name** - A plain text name for the repository. Enter **`My Files`**.
      - **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you enter for **Name**.
      - **Type** - The content type of the repository. Select **`file`**.
      - **Upstream URL** - The path to the local directory with the repository to use as the source, in the form **`file:///my_file_repo`**.

- **Verify SSL** - Ensure this field is clear.
- **Upstream Username** - Ensure this field is clear.
- **Upstream Password** - Ensure this field is clear.

3. Select **Save** to save this repository entry.
4. Update the file type repository:
   a. Navigate to **Content** > **Products**.
   b. Select the name of a product, in this example *My File Product*.
   c. Select the name of the repository you want to update, in this example *My Files*.
   d. From the **Select Action** menu, select **Sync Now**.
   e. Visit the URL where the repository is published to see the files.

6.13. CREATING A CUSTOM CONTAINER IMAGES REPOSITORY

Custom products can include repositories for private Docker-formatted container images.

For Web UI Users

Navigate to **Content** > **Products**. Select a product by name. On the **Products** page, click **New Repository**. On the **New Repository** page, enter the following details:

- **Name** - A plain text name for the repository.
- **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **Type** - The type of repository. Select **`docker`**.
- **Upstream URL** - The URL of the registry that you want to use as a source.
- **Upstream Username** - If required for authentication, enter the user name for the upstream repository. Clear this field if the repository does not require authentication.
- **Upstream Password** - Enter the password for the private upstream repository. Clear this field if the repository does not require authentication.

Click **Save** to save this repository entry.

CHAPTER 7. CREATING AN APPLICATION LIFE CYCLE

In Section 1.2, “Defining the Application Life Cycle”, we defined what an application life cycle is and why it is important to Red Hat Satellite 6. In this chapter, we take a closer look at planning the application life cycle and implementing it in Red Hat Satellite 6.

7.1. REVISITING THE APPLICATION LIFE CYCLE

As we discussed, the application life cycle defines how systems appear at a certain stage. However, the actual application life cycle depends on your organization and how it structures a particular production chain.

For example, an email server might only require a simple application life cycle where you have a production-level server for real world use and a test server for trying out the latest mail server packages. When the test server passes the initial phase, we can set the production-level server to use the new packages.

Another example might be an development life cycle for a software product. You might aim to develop a new piece of software in a development environment, have it tested in a quality assurance environment, pre-release it as a beta, then release it as a production-level application.

Each application life cycle uses a set of stages called **environments**. Each environment act as a particular state for our systems. Each environment also follows on from a previous environment, creating a chain of environments that becomes our application life cycle. Each application life cycle starts with an initial **Library**, which acts as a central source in our Definitive Media Library (DML). The Library environment contains all of the content we previously synchronized in our previous chapters. From this point, we create additional environments that link starting from the Library environment.

7.2. CREATING A NEW APPLICATION LIFE CYCLE

For our scenario, we aim to create simple application life cycle for developing and releasing ACME’s Exampleware. We use the Library environment as our initial environment, then chain an additional three environments in this order: **Development**, **Testing**, and **Production**.

For Web UI Users

Navigate to **Content** > **Lifecycle Environments**. This screen displays the current environments in your application life cycle. At the moment, only the Library environment exists.

Click **New Environment Path** to start a new application life cycle. A form appears for adding a new environment to Library. Enter **`Development`** for the **Name** and the **Label** automatically completes. Enter **`Environment for ACME’s Development Team`** for the **Description**. Click **Save**.

We return to the environments listing. A new table appears with our Development environment. This new table represents our new application life cycle. Now we add the rest of the environments to the application life cycle.

Click **Add New Environment** just above the new table. This time, enter **`Testing`** for **Name** and **`Environment for ACME’s Quality Engineering Team`** for the **Description**. Click **Save**.

Finally, click **Add New Environment** again. Enter **`Production`** for **Name** and **`Environment for ACME’s Product Releases`** for the **Description**. Click **Save**.

The final application life cycle appears as a table with our three environments: **Development**, **Testing**, and **Production**.

For CLI Users

Run the **hammer lifecycle-environment create** for each environment. Use the **--prior** option to specify the previous environment. For example:

```
# hammer lifecycle-environment create \
--name "Development" \
--description "Environment for ACME's Development Team" \
--prior "Library" \
--organization "ACME"
# hammer lifecycle-environment create \
--name "Testing" \
--description "Environment for ACME's Quality Engineering Team" \
--prior "Development" \
--organization "ACME"
# hammer lifecycle-environment create \
--name "Production" \
--description "Environment for ACME's Product Releases" \
--prior "Testing" \
--organization "ACME"
```

View your chain with the **hammer lifecycle-environment paths** command:

```
# hammer lifecycle-environment paths --organization "ACME"
-----------------------------------------------
LIFECYCLE PATH
-----------------------------------------------
Library >> Development >> Testing >> Production
-----------------------------------------------
```

7.3. ADDING LIFE CYCLE ENVIRONMENTS TO CAPSULE SERVERS

The following is an optional step for environments that use Capsule Servers to provision content.

If your Capsule Server has content functionality enabled, you must add an environment. Adding an environment enables Capsule Server to synchronize content from Satellite Server and provide content to host systems.

Capsule Server is configurable using Hammer CLI on Satellite Server, or through the web UI.

For Web UI Users

Under **Infrastructure** > **Capsules**, select the capsule.

Click **Edit**.

Select **Env** in the Life Cycle Environments tab.

To synchronize capsule’s content, click the **Synchronize** button in the Overview tab.

Select one of two options:

- Optimized Sync
- Complete Sync

For CLI Users

Display a list of all Capsule Servers and note the ID:

```
# hammer capsule list
```

Using the ID, verify the details of your Capsule Server:

```
# hammer capsule info --id capsule_id_number
```

Verify the life cycle environments available and note the environment ID:

```
# hammer capsule content available-lifecycle-environments \
--id capsule_id_number
```

Available life cycle environments are available for Capsule Server, but not currently attached.

Add the life cycle environment to your Capsule Server:

```
# hammer capsule content add-lifecycle-environment \
--id capsule_id_number --environment-id environment_id_number
```

Repeat for each life cycle environment you want to add to Capsule Server.

To synchronize all content from your Satellite Server environment to Capsule Server, enter the following command:

```
# hammer capsule content synchronize --id capsule_id_number
```

To synchronize a specific life cycle environment from your Satellite Server to Capsule Server, enter the following command:

```
# hammer capsule content synchronize --id external_capsule_id_number \
--environment-id environment_id_number
```

For more information on working with Life Cycle Environments, see Life Cycle Environments in *Planning for Red Hat Satellite 6*.

7.4. PROMOTING CONTENT ACROSS THE APPLICATION LIFE CYCLE

Content moves from one environment to the next in an application life cycle chain. This is a process called **promotion**. Promotion is an important concept to understand because it is the basis for managing content across an application life cycle.

For our scenario, let’s say that ACME’s Exampleware is in full development and production. ACME packages their Exampleware content into a RPM file, which becomes part of a separate product. In this situation, ACME might have version 1.0 of Exampleware released, which means **`exampleware-1.0-0.noarch.rpm`** is in the Production environment.

ACME wants to release a patch for Exampleware, so they start development on version 1.1 of Exampleware. In this instance, the environments in the application life cycle would contain the following version of Exampleware:

Development	Testing	Production
exampleware-1.1-0.noarch.rpm	exampleware-1.0-0.noarch.rpm	exampleware-1.0-0.noarch.rpm

After completing development on the patch, ACME promotes the RPM to the Testing environment so ACME's Quality Engineering team can review the patch. The application life cycle then contains the following package versions in each environment:

Development	Testing	Production
exampleware-1.1-0.noarch.rpm	**exampleware-1.1-0.noarch.rpm**	exampleware-1.0-0.noarch.rpm

While the Quality Engineering team reviews the patch, the Development team starts work on Exampleware 2.0. This results in the following application life cycle:

Development	Testing	Production
exampleware-2.0-0.noarch.rpm	exampleware-1.1-0.noarch.rpm	exampleware-1.0-0.noarch.rpm

The Quality Engineering team completes their review of the patch. Now Exampleware 1.1 is ready for release. ACME promotes 1.1 to the Production environment:

Development	Testing	Production
exampleware-2.0-0.noarch.rpm	exampleware-1.1-0.noarch.rpm	**exampleware-1.1-0.noarch.rpm**

The Development team completes their work on Exampleware 2.0 and promotes it to the Testing environment:

Development	Testing	Production
exampleware-2.0-0.noarch.rpm	**exampleware-2.0-0.noarch.rpm**	exampleware-1.1-0.noarch.rpm

Finally, the Quality Engineering team reviews the package. After a successful review, we promote the package to the Production environment:

Development	Testing	Production
exampleware-2.0-0.noarch.rpm	exampleware-2.0-0.noarch.rpm	**exampleware-2.0-0.noarch.rpm**

This example demonstrates the mapping of ACME's development process into a Red Hat Satellite 6 application life cycle. Each environment contains a set of systems registered to Red Hat Satellite 6. These systems only have access to repositories relevant to their environment. When we promote packages from one environment to the next, the target environment's repositories receive new package versions. As a result, each system in the target environment can update to the new package versions.

Another concept we explore in the next chapter is **Content Views**. A Content View is essentially a collection of content gathered and filtered from the Library and published in its own repository. The Red Hat and custom repositories we synchronized previously act as source content, while the Content Views customize exactly how the end repositories appear. Content Views contain packages and Satellite Server promotes Content Views across environments in the application life cycle. An environment might use one Content View (Version 1.0) but then might receive a new version of the Content View (Version 1.1). The repositories from Version 1.0 are replaced with the repositories for Version 1.1, which might include a new version of the Exampleware RPMs.

7.5. ADDING LIFE CYCLE ENVIRONMENTS TO A RED HAT SATELLITE CAPSULE SERVER

If a newly created Red Hat Satellite Capsule Server has content functionality enabled, the Capsule Server needs an environment added to enable it to synchronize content from Satellite Server and provide content to host systems.

IMPORTANT

Satellite Capsule Server is configured through Satellite Server's command line interface (CLI). Enter all **hammer** commands on Satellite Server.

To Add Environments to Satellite Capsule Server:

1. Log in to Satellite Server CLI as root.
2. Select the Red Hat Satellite Capsule Server that you want from the list and take note of its **id**:

   ```
   # hammer capsule list
   ```

   To verify Satellite Capsule Server's details, enter the following command:

   ```
   # hammer capsule info \
   --id capsule_id_number
   ```

3. Verify the list of life cycle environments available for the Red Hat Capsule Server and note down the **environment id**:

   ```
   # hammer capsule content available-lifecycle-environments \
   --id capsule_id_number
   ```

   Where:

   available-lifecycle-environments are life cycle environments that are available to Satellite Capsule but are currently not attached to Satellite Capsule.

4. Add the life cycle environment to Satellite Capsule Server:

   ```
   # hammer capsule content add-lifecycle-environment \
   --id capsule_id_number \
   --environment-id environment_id_number
   ```

   Where:

   - **capsule_id_number** stands for Satellite Capsule Server's identification number.

- **`environment_id_number`** stands for the life cycle environment's identification number.

Repeat this step for every life cycle environment to be added to the Capsule Server.

5. Synchronize the content from Satellite Server's environment to Satellite Capsule Server:

```
# hammer capsule content synchronize \
--id capsule_id_number
```

When an external Satellite Capsule Server has various life cycle environments, and only one life cycle environment needs to be synchronized, it is possible to target a specific environment by specifying the environment identification number:

```
# hammer capsule content synchronize \
--id external_capsule_id_number \
--environment-id environment_id_number
```

7.6. PROMOTING CONTENT VIEWS

After you have created a Content View and an environment path consisting of two or more life cycle environments, you can promote the Content View from one environment to the next as required. This means that the most recent version of the Content View that exists in a specified environment is promoted, or copied, to the next environment in the life cycle environment path.

You can promote a Content View to any environment where that version does not exist. The system automatically suggests the next environment in the life cycle environment path, but you can override this and promote to a different environment if required.

To Promote a Content View:

1. On the main menu, click **Content** > **Content Views**.
2. In the **Name** column, click the name of the Content View that you want to promote.
3. On the **Versions** tab, identify the latest version, and click **Promote**.
4. Identify the promotion path where you want to promote the Content View, select the appropriate life cycle environment, and click **Promote Version**.
5. After the promotion has completed, the **Versions** tab updates to display the new status of your Content Views.

7.7. REMOVING LIFE CYCLE ENVIRONMENTS FROM SATELLITE SERVER

This procedure describes how to remove a life cycle environment from Red Hat Satellite.

To Remove a Life Cycle Environment:

1. On the main menu, click **Content** > **Life Cycle Environments**.
2. Click the name of the life cycle environment that you want to remove, and then click **Remove Environment**.

3. In the confirmation dialog box, click **Remove** to remove the environment.

> **NOTE**
>
> You can only delete the latest environment in an environment path. For example, if three environments exist in the order **Library**, **Dev**, and **Prod**, you need to delete **Prod** before you can delete **Dev**. You cannot delete the **Library** environment.

7.8. REMOVING LIFE CYCLE ENVIRONMENTS FROM CAPSULE SERVER

There are multiple reasons to remove life cycle environments from Capsule Server. For example:

- When life cycle environments are no longer relevant to the host systems
- When life cycle environments have been incorrectly added to Capsule Server

To remove a life cycle environment from Capsule Server:

1. Log in to Satellite Server CLI as the root user.
2. Select the Capsule Server that you want from the list and take note of its **id**:

   ```
   # hammer capsule list
   ```

   To verify the Capsule Server's details, enter the following command:

   ```
   # hammer capsule info \
   --id capsule_id_number
   ```

3. Verify the list of life cycle environments currently attached to the Capsule Server and take note of the **environment id**:

   ```
   # hammer capsule content lifecycle-environments \
   --id capsule_id_number
   ```

4. Remove the life cycle environment from Capsule Server:

   ```
   # hammer capsule content remove-lifecycle-environment \
   --id capsule_id_number \
   --environment-id environment_id_number
   ```

   Where:

   - **capsule_id_number** is the Capsule Server's identification number.
   - **environment_id_number** is the life cycle environment's identification number.
     Repeat this step for every life cycle environment to be removed from the Capsule Server.

5. Synchronize the content from Satellite Server's environment to Capsule Server:

   ```
   # hammer capsule content synchronize \
   --id capsule_id_number
   ```

CHAPTER 8. MANAGING CONTENT VIEWS

This chapter shows how to create different types of Content Views and apply various filters to them.

8.1. UNDERSTANDING CONTENT VIEWS

Red Hat Satellite 6 uses Content Views to create customized repositories from the core repositories in your Definitive Media Library (DML). It achieves this through defining which repositories to use and then applying certain filters to the content. These filters include both package filters, package group filters, and errata filters. We use Content Views as a method to define which software versions a particular environment uses. As mentioned in the previous chapter, a Production environment might use a Content View containing older package versions, while a Development environment might use a Content View containing newer package versions.

Each Content View creates a set of repositories across each environment, which Satellite Server stores and manages. When we promote a Content View from one environment to the next environment in the application life cycle, the respective repository on Satellite Server updates and publishes the packages. For example, let’s say that we use a Content View that contains our Exampleware package:

	Development	Testing	Production
Content View Version and Contents	Version 2 - exampleware-1.1-0.noarch.rpm	Version 1 - exampleware-1.0-0.noarch.rpm	Version 1 - exampleware-1.0-0.noarch.rpm

The repositories for Testing and Production contain the **`exampleware-1.0-0.noarch.rpm`** package. If we promote Version 2 of the Content View from Development to Testing, the repository for Testing regenerates and then contains the **`exampleware-1.1-0.noarch.rpm`** package:

	Development	Testing	Production
Content View Version and Contents	Version 2 - exampleware-1.1-0.noarch.rpm	**Version 2 - exampleware-1.1-0.noarch.rpm**	Version 1 - exampleware-1.0-0.noarch.rpm

This ensures systems are designated to a specific environment but receive updates when that environment uses a new version of the Content View.

The general workflow for creating Content Views for filtering and snapshotting is as follows:

1. Create a Content View.
2. Add the repository and the Puppet modules that you want to the Content View.
3. Optionally create one or more Filters to fine tune the content of the Content View.
4. Publish the Content View.
5. Attach the Content Host to the Content View.
6. Optionally promote the Content View to another environment.

NOTE

If a repository is not associated with the Content View, the file **`/etc/yum.repos.d/redhat.repo`** remains empty and systems registered to it cannot receive updates.

NOTE

Systems can only be associated with a single Content View. To associate a system with multiple Content Views, create a composite Content View. For more information, see Section 8.3.1, “Creating a Composite Content View”.

8.2. STANDARD CONTENT VIEWS

A standard Content View is a single Content View that can be associated with a single host.

8.2.1. Creating a Simple Content View

For this example, we’ll create a simple Content View using just two repositories and no filters. The repositories to include are the Red Hat Enterprise Linux repository and the Satellite Tools repository. Note that this Content View contains all RPMs for Red Hat Enterprise Linux and takes several minutes to publish.

For Web UI Users

Navigate to **Content** > **Content Views** and click **Create New View**. This displays a form for **View Details**. Enter the following information:

- **Name** - The plain text name for the view. Enter **`Base`**.
- **Label** - An internal ID for the view. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **Description** - A plain text description of the view. Enter **`Base operating system`**.
- **Composite View?** - Defines whether to use a Composite Content View. Leave this check box clear.

Click **Save** to complete.

This creates a new Content View entry and now you can add repositories to it. Select the repositories for Red Hat Enterprise Linux 7 Server RPMs (not the Kickstart RPMs) and Red Hat Satellite Tools. Click **Add Repository**. This adds all packages from these repositories to the Content View.

Our Content View is ready to publish. Navigate to **Versions** and click **Publish New Version**. Satellite Server provides some details about the new version (Version 1) and allows you to enter a **Description** for the version, which is useful to logging changes for new content versions. Enter **`Initial Content View for our operating system`** and click **Save**.

Satellite Server creates the new version of the view and publishes it to the Library environment.

For CLI Users

Obtain a list of repository IDs:

```
# hammer repository list --organization "ACME"
```

–

For our example, the two repositories we require are using 1 and 2 for their IDs:

ID	Name
1	Red Hat Enterprise Linux 7 Server RPMs x86_64 7Server
2	Red Hat Satellite Tools 6.4 for RHEL 7 Server RPMs x86_64

Create the Content View and add repositories:

```
# hammer content-view create \
--name "Base" \
--description "Base operating system" \
--repository-ids 1,2 \
--organization "ACME"
```

Now publish the view:

```
# hammer content-view publish \
--name "Base" \
--description "Initial Content View for our operating system" \
--organization "ACME"
```

Satellite Server creates the new version of the view and publishes it to the Library environment.

8.2.2. Creating a Content View with a Puppet Module

For this example, we'll create another Content View using one repository and no filters. The repository to include is the PostgreSQL repository.

For Web UI Users

Navigate to **Content** > **Content Views** and click **Create New View**. This displays a form for **View Details**. Enter the following information:

- **Name** - The plain text name for the view. Enter **`Database`**.
- **Label** - An internal ID for the view. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.
- **Description** - A plain text description of the view. Enter **`PostgreSQL Database`**.
- **Composite View?** - Defines whether to use a Composite Content View. Leave this check box clear.

Click **Save** to complete.

This creates a new Content View entry. Select the repositories for PostgreSQL and click **Add Repository**. This adds all packages from the PostgreSQL repository to the Content View.

Navigate to **Puppet Modules**. Click **Add New Module**. This shows all our imported Puppet modules. Navigate to the **`postgresql`** module and click **Select a Version**.

Navigate to the entry for **Use Latest** and click **Select Version** in the **Actions** column.

The Puppet module is now added to the Content View. Let's publish a new version.

Navigate to **Versions** and click **Publish New Version**. Enter **`Initial RPMs and Puppet module for database`** in the **Description** and click **Save**.

Satellite Server creates the new version of the view and publishes it to the Library environment.

For CLI Users

Obtain a list of repository IDs:

```
# hammer repository list --organization "ACME"
```

We only require the PostgreSQL RPM repository, not the Puppet module repository. In our example, we use the 4 for the ID of the PostgreSQL RPMs:

ID	Name
4	PostgreSQL 9.5

Create the Content View and add repositories:

```
# hammer content-view create \
--name "Database" --description "PostgreSQL Database" \
--repository-ids 4 \
--organization "ACME"
```

Now publish the view:

```
# hammer content-view publish \
--name "Database" \
--description "Initial RPMs and Puppet module for database" \
--organization "ACME"
```

Satellite Server creates the new version of the view and publishes it to the Library environment.

8.3. COMPOSITE CONTENT VIEWS

A Composite Content View combines the content from several Content Views. For example, you might have separate Content Views to manage a base OS and an application individually. A Composite Content View allows you to merge the contents of both Content Views into a new repository. The repositories for the original Content Views still exist but a new repository now exists for the combined content.

As a use case scenario, a company might aim to develop an application that supports different database servers. The general application stack appears as:

Exampleware Stack
Application
Database
Operating System

The company might develop four individual Content Views:

- Red Hat Enterprise Linux (Operating System)
- PostgreSQL (Database)
- MariaDB (Database)
- Exampleware (Application)

The company can then create two Composite Content Views. One with a PostgreSQL database:

Composite Content View 1 - Exampleware on PostgreSQL
Exampleware (Application)
PostgreSQL (Database)
Red Hat Enterprise Linux (Operating System)

And one with MariaDB:

Composite Content View 2 - Exampleware on MariaDB
Exampleware (Application)
MariaDB (Database)
Red Hat Enterprise Linux (Operating System)

Each individual Content View is then managed and published separately. When we create a new version of the application stack, we publish a new version of the Composite Content Views. You can also select the **Auto Publish** option when creating a Composite Content View, and then the Composite Content View is automatically republished when a Content View it includes is republished.

IMPORTANT

Composite Content Views do not allow more than one of each repository. For example, if you attempt to include two Content Views using the same repository in a Composite Content View, Satellite Server reports an error.

8.3.1. Creating a Composite Content View

Let's create a Content View for ACME that combines our two existing Content Views.

For Web UI Users

Navigate to **Content** > **Content Views** and click **Create New View**. Provide the following details for the view:

- **Name** - **`Stack`**
- **Description** - **`A stack that includes a base operating system and a database`**
- **Composite View?** - Select to create a Composite Content View.
- **Auto Publish** - Select if you want the Composite Content View to be automatically republished when a Content View it includes is republished.

The Content View listing for the Composite Content View appears. Select both the Base and Database Content Views. Also, the Base Content View contains two versions, which means we need to select a version. Select Version 2, which contains the errata filter. After selecting the Content Views and their versions, click **Add Content Views**.

Click **Publish New Version** to publish the Composite Content View. Enter **`Initial version of Stack`** for **Description** and click **Save**.

When the publication completes, notice that the **Content** column reports the total package, errata, and Puppet module count for all included Content Views.

Now **Promote** the Composite Content View across all environments: **Development**, **Testing**, **Production**.

As you can see, Composite Content Views are published and promoted across application life cycle environments similar to regular Content Views.

For CLI Users

Before we create the Composite Content Views, we need the version IDs for our existing Content Views:

```
# hammer content-view version list \
--full-results true \
--organization "ACME"
```

For our scenario, the Database v1.0 version ID is 5 and the Base v2.0 version ID is 6. Create a new Composite Content View called **`Stack`** and include the version IDs with the **`--component-ids`** option. When the **`--auto-publish`** option is set to **`yes`**, the Composite Content View is automatically republished when a Content View it includes is republished:

```
# hammer content-view create \
--composite \
--auto-publish yes \
--name "Stack" \
--description "A stack that includes a base operating system and a
database" \
--component-ids 5,6 \
--organization "ACME"
```

Publish the Composite Content View:

```
# hammer content-view publish \
--name "Stack" \
--description "Initial version of Stack" \
--organization "ACME"
```

Promote the Composite Content View across all environments:

```
# hammer content-view version promote \
--content-view "Stack" \
--version 1 \
--to-lifecycle-environment "Development" \
--organization "ACME"
# hammer content-view version promote \
--content-view "Stack" \
--version 1 \
--to-lifecycle-environment "Testing" \
--organization "ACME"
# hammer content-view version promote \
--content-view "Stack" \
--version 1 \
--to-lifecycle-environment "Production" \
--organization "ACME"
```

8.4. CONTENT FILTERS

Content Views also use filters to include or restrict certain RPM content. Without these filters, a Content View includes everything from selected repositories.

Content filters are either one of the following types:

- **Include** - Defines the content to include in the view. This filter’s behavior follows a pattern where you start with no content, then select which content to add from the selected repositories. Use this filter to combine multiple content items.

- **Exclude** - Defines the content to exclude in the view. This filter’s behavior follows a pattern where you start with all content from selected repositories, then select which content to remove. Use this filter when you want to use most of a particular content repository but exclude certain packages, such as blacklisted packages. The filter uses all content in the repository except for the content you select.

NOTE

If using a combination of Include and Exclude filters, publishing a Content View triggers the Include filters first, then the exclude filters. In this situation, select which content to include, then which content to exclude from the inclusive subset.

There are also four types of content to filter:

- **Package** - Filter packages based on their name and version number.

- **Package Group** - Select which package groups to add to the filter. The list of package groups is based on the repositories added to the Content View.

- **Erratum (by ID)** - Select which specific errata to add to the filter. The list of Errata is based on the repositories added to the Content View.
- **Erratum (by Date and Type)** - Select a issued or updated date range and errata type (Bugfix, Enhancement, or Security) to add to the filter.

IMPORTANT

Filters do not resolve any dependencies of the packages listed in the filters. Ensure to add package dependencies to the filter. This might require some level of testing to determine what dependencies are required.

Let’s look at some examples of using content filters.

Example 1

Create a repository with the base Red Hat Enterprise Linux packages. This filter requires a Red Hat Enterprise Linux repository added to the Content View.

Filter:

- **Inclusion Type:** Include
- **Content Type:** Package Group
- **Filter:** Select only the **Base** package group

Example 2

Create a repository that excludes all errata, except for security updates, after a certain date. This is useful if your aim to perform system updates on a regularly scheduled basis with the exception of critical security updates, which must be applied immediately. This filter requires a Red Hat Enterprise Linux repository added to the Content View.

Filter:

- **Inclusion Type:** Exclude
- **Content Type:** Erratum (by Date and Type)
- **Filter:** Select only the **Bugfix** and **Enhancement** errata types, and clear the **Security** errata type. Set the **Date Type** to **Updated On**. Set the **Start Date** to the date you want to restrict errata. Leave the **End Date** blank to ensure any new non-security errata is filtered.

Example 3

A combination of Example 1 and Example 2 where we only require the Base OS packages but want to filter out recent bug fix and enhancement errata. This requires two filters attached to the same Content View. The Content View processes the Include filter first, then the Exclude filter.

Filter 1:

- **Inclusion Type:** Include
- **Content Type:** Package Group
- **Filter:** Select only the **Base** package group

Filter 2:

- **Inclusion Type:** Exclude
- **Content Type:** Erratum (by Date and Type)
- **Filter:** Select only the **Bugfix** and **Enhancement** errata types, and clear the **Security** errata type. Set the **Date Type** to **Updated On**. Set the **Start Date** to the date you want to restrict errata. Leave the **End Date** blank to ensure any new non-security errata is filtered.

For another example of how content filters work, see the following article: "How do content filters work in Satellite 6"

8.4.1. Creating a Content Filter

For our scenario, we aim to create a content filter that restricts errata items after a certain date for ACME's base operating system.

For Web UI Users

Navigate to **Content** > **Content Views** and select the **`Base`** Content View. Navigate to **Yum Content** > **Filters** and click **New Filter**. Enter the following details for your filter:

- **Name** - **`Errata Filter`**
- **Content Type** - Erratum - Date and Type
- **Inclusion Type** - Exclude
- **Description** - **`Exclude errata items from the last year, with the exception of security updates`**

Click **Save**.

The **Erratum Date Range** screen appears, which allows you to select the errata type to filter and the date range. Select only **Enhancement** and **Bugfix**:

```
[ ] Security
[X] Enhancement
[X] Bugfix
```

For the **Date Type**, select either **Issued On** (the issued date of the erratum) or **Updated On** (the date of the erratum's last update). If the erratum has not been updated after the creation date, then the **Issued On** and **Updated On** date are the same for that erratum. Note that **Issued On** just means that the errata items are filtered based on the issued date. The filter does not exclude any updates that have been made to that erratum.

For the **Start Date**, select today's date from one year ago.

For the **End Date**, leave this field blank.

Click **Save**.

NOTE

You can also select which specific repositories use this filter. Select the **Affected repositories** tab to select these repositories. For this example, there is only one repository to use in our filter.

Our filter is complete. Click **Publish New Version** to publish the resulting repository. For the **Version Details** enter **`Adding errata filter`** in the **Description**. Click **Save**.

When the Content View completes publication, notice the **Content** column reports a reduced number of packages and errata (except Security errata) from the initial repository. This means the filter successfully excluded the all non-security errata from the last year.

Promote this Content View across all environments: **Development**, **Testing**, **Production**.

For CLI Users

Add a filter to the Content View. Use the **`--inclusion false`** option to set the filter to an Exclude filter:

```
# hammer content-view filter create \
--name "Errata Filter" \
--type erratum --content-view "Base" \
--description "Exclude errata items from the last year, with the exception
of security updates" \
--inclusion false  \
--organization "ACME"
```

Add a rule to the filter:

```
# hammer content-view filter rule create \
--content-view "Base" \
--content-view-filter "Errata Filter" \
--start-date "2015-01-01" \
--types enhancement,bugfix \
--date-type updated \
--organization "ACME"
```

Publish the Content View:

```
# hammer content-view publish \
--name "Base" \
--description "Adding errata filter" \
--organization "ACME"
```

Promote the view across all environments:

```
# hammer content-view version promote \
--content-view "Base" \
--version 1 \
--to-lifecycle-environment "Development" \
--organization "ACME"
# hammer content-view version promote \
--content-view "Base" \
--version 1 \
```

```
--to-lifecycle-environment "Testing" \
--organization "ACME"
# hammer content-view version promote \
--content-view "Base" \
--version 1 \
--to-lifecycle-environment "Production" \
--organization "ACME"
```

8.5. PROMOTING A CONTENT VIEW

Satellite Server has published two Content Views and the repositories are now available to the Library environment. Let’s promote one of these Content Views so the repository appears available to the other environments.

> **NOTE**
>
> Non-administrator users require two permissions to promote a Content View to an environment:
>
> 1. **`promote_or_remove_content_views`**
> 2. **`promote_or_remove_content_views_to_environment`**.
>
> The **`promote_or_remove_content_views`** permission restricts which Content Views a user is allowed to promote.
>
> The **`promote_or_remove_content_views_to_environment`** permission restricts the environments to which a user is allowed to promote Content Views.
>
> These permissions make it possible to designate that certain users are permitted to promote certain Content Views to certain environments, but not to other environments. For example, these permissions can be used to limit a user so that they are permitted to promote to test environments, but not to production environments. This provides a kind of multi-level authentication.
>
> Both permissions must be assigned to a user to allow them to promote Content Views.

For Web UI Users

Ensure you are on the **Versions** screen for the Database Content View. In the versions table, navigate to **Version 1.0** of our view and click **Promote** in the Actions column. A new screen appears where you can select the target environment. Select the **Development** environment and click **Promote Version**. After a few minutes, the promotion completes.

Click on the **Promote** button again. This time select the **Testing** environment and click **Promote Version**.

Finally click on the **Promote** button again. Select the **Production** environment and click **Promote Version**.

Now the repository for the Content View appears in all environments.

For CLI Users

Promote the Content View using the **`hammer content-view version promote`** each time:

```
# hammer content-view version promote \
--content-view "Database" \
--version 1 \
--to-lifecycle-environment "Development" \
--organization "ACME"
# hammer content-view version promote \
--content-view "Database" \
--version 1 \
--to-lifecycle-environment "Testing" \
--organization "ACME"
# hammer content-view version promote \
--content-view "Database" \
--version 1 \
--to-lifecycle-environment "Production" \
--organization "ACME"
```

Now the database content is available in all environments.

8.6. REGISTERING SYSTEMS TO ENVIRONMENTS AND THEIR CONTENT VIEWS

With Content Views in place, we can register systems to these environments and views.

8.6.1. Registering a RHEL System with Subscription Manager

First, log into a test Red Hat Enterprise Linux 7 client system as the **root** user and download the consumer RPM for your Satellite Server. This is located in the **pub** directory on the host. For example, for a Satellite Server with the host name **satellite.example.com**, enter the following command on the client to register:

```
[root@client ~]# rpm -Uvh http://satellite.example.com/pub/katello-ca-
consumer-latest.noarch.rpm
```

Enter the following command to view a list of environments and their Content Views on Satellite Server:

```
[root@client ~]# subscription-manager environments --org="acme"
```

Register the client system to an environment and Content View on Satellite Server:

```
[root@client ~]# subscription-manager register --org="acme" --
environment="Development/Stack"
```

NOTE

The client system asks for the user name and password for a Satellite Server user that belongs to the ACME organization. As an alternative, you can use an activation key to register a system, which is discussed in Chapter 9, *Managing Activation Keys*.

The client system now uses the published repositories from the Stack Content View in the Development environment.

8.6.2. Registering an Atomic Host with Subscription Manager

The following procedure explains how to register an Atomic Host with Subscription Manager.

Retrieve **katello-rhsm-consumer** from Satellite server:

```
[root@atomic_client ~]# wget http://satellite.example.com/pub/katello-rhsm-consumer
```

Change the mode of **katello-rhsm-consumer** in order to make it executable:

```
[root@atomic_client ~]# chmod +x katello-rhsm-consumer
```

Run **katello-rhsm-consumer**:

```
[root@atomic_client ~]# ./katello-rhsm-consumer
```

Register with **Red Hat Subscription Manager**:

```
[root@atomic_client ~]# subscription-manager register
```

NOTE

Because Atomic is functionally an appliance, we do not recommend that you try to install **katello-agent** on it.

CHAPTER 9. MANAGING ACTIVATION KEYS

At this point, we have published some Content Views, which resulted in Satellite Server publishing repositories. Systems now can register to Satellite Server and consume content from these repositories. Systems register in a similar way to how they register to the Red Hat Customer Portal. For example, users can use Red Hat Subscription Manager (**`subscription-manager`**) with the **`--baseurl`** pointing to Satellite Server instead of the Red Hat Content Delivery Network.

There are two methods to register a system. The first is to authenticate with a Satellite Server user name and password, which we explored in the previous chapter. An alternative and preferred method is to use an activation key, which acts as an authentication token. Activation keys provide a method for easy system registration and subscription attachment. Users can create multiple keys and associate them with different environments and Content Views. For example, you might create a basic activation key with a subscription for Red Hat Enterprise Linux workstations and associate it with Content Views from a particular environment.

Activation keys define selected properties of content hosts. You can use activation keys during content host registration to improve the speed, simplicity and consistency of the process. Activation can specify:

- Associated subscriptions and subscription attach behavior.
- Available products and repositories.
- A life cycle environment and a Content View.
- Host collection membership.

The same activation key can be applied to multiple content hosts, as long as it contains enough subscriptions. However, activation keys only set the initial configuration for a content host. When it is registered to an organization, other content which that organization possesses can be attached to the content host manually.

A content host can be associated with multiple activation keys that are combined to define the host settings. In case of conflicting settings, the last specified activation key takes precedence.

Note that 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.

For more information about activation keys and more use case examples outside of this guide, see "Activation Key Enhancements with Red Hat Satellite 6.1" on the Red Hat Customer Portal.

9.1. CREATING AN ACTIVATION KEY

With activation keys you can define how the content host is subscribed during registration. The subscription behavior defined by the activation key depends on two factors:

- Are there any subscriptions associated with the activation key?
- Is the auto-attach option enabled?

Based on the above factors, there are three possible scenarios for subscribing with activation keys:

- **Activation key with no subscriptions specified**. With no subscriptions specified and auto-attach enabled, hosts using the activation key search for the best fitting subscription from the ones provided by Satellite Server. This is similar to entering the **`subscription-manager --`**

auto-attach command.

- **Activation key providing a custom subscription pool for auto-attach**. If there are subscriptions specified and auto-attach is enabled, hosts using the activation key select the best fitting subscription from the list specified in the activation key.

- **Activation key with the exact set of subscriptions**. If there are subscriptions specified and auto-attach is disabled, hosts using the activation key are associated with all subscriptions specified in the activation key.

NOTE

If a custom product (typically containing content not provided by Red Hat) is assigned to an activation key, this product is always enabled for the registered content host regardless of the auto-attach setting.

For Web UI Users

1. Navigate to **Content** → **Activation keys** and click **Create Activation Key**. Provide the activation key with the following information:

   - **Name** - The name of the activation key. We use this name during the system registration process. In the **Name** field, type ***development-stack***.

   - **Limit** - Defines the maximum number of systems you can register with the activation key. Select the **Unlimited Hosts** check box.

   - **Description** - A plain text description for the activation key. In the **Description** field, type *Exampleware Stack in the Development Environment*.

   - **Environment** - The environment to use. In the **Environment** area, select ***Development***.

   - **Content View** - The Content View (and, by extension, the repository) in the environment to use. From the **Content View** list, select ***Stack***.

   Click **Save**. The activation key details screen displays.

2. To define which products to attach and repositories to enable upon registration, navigate to the **Subscriptions** tab. Click **Add**, select both the **Red Hat Enterprise Linux** subscription and the **PostgreSQL** product, and click **Add Selected**.

   NOTE

   The **Auto-Attach** option is enabled. This automatically attaches these products to the system upon registration and enables the required repositories. If **Auto-Attach** is disabled for the activation key, the system only registers to Satellite Server without attaching any subscriptions or content.

3. Navigate to the **Product Content** page. As default, Satellite Server only enables:

   - The repository that best matches the system requirements. In this case, it is only the Red Hat Enterprise Linux 7 Server RPMs.

   - Any custom content.

   Our scenario must have the following defaults set:

PostgreSQL:

- PostgreSQL 9.5 - Enabled: **`Yes (Default)`**
- PostgreSQL Puppet Modules - Enabled: **`Yes (Default)`**

Red Hat Enterprise Linux Server:

- Red Hat Enterprise Linux 7 Server (Kickstart) - Enabled: **`No (Default)`**
- Red Hat Satellite Tools 6.4 (for RHEL 7 Server) (RPMs) - Enabled: **`No (Default)`**
- Red Hat Enterprise Linux 7 Server (RPMs) - Enabled: **`Yes (Default)`**

However, we want to enable the Red Hat Satellite Tools repository because that contains our configuration tools (such as **`katello-agent`** and **`puppet`**). Change it to the following:

- Red Hat Satellite Tools 6.4 (for RHEL 7 Server) (RPMs) - Enabled: **`Override to Yes`**

4. Click **Save**

For CLI Users

1. Create the activation key:

```
# hammer activation-key create \
--name "development-stack" \
--unlimited-hosts true \
--description "Exampleware Stack in the Development Environment" \
--lifecycle-environment "Development" \
--content-view "Stack"  \
--organization "ACME"
```

2. Obtain a list of your subscription IDs:

```
# hammer subscription list --organization "ACME"
```

3. Attach the Red Hat Enterprise Linux subscription UUID to the activation key:

```
# hammer activation-key add-subscription \
--name "development-stack" \
--subscription-id ff808181533518d50152354246e901aa \
--organization "ACME"
```

4. Attach the PostgreSQL product to the activation key:

```
# hammer activation-key add-subscription \
--name "development-stack" \
--subscription-id ff8081815239acdc015238fefaa10002 \
--organization "ACME"
```

5. List the product content associated with the activation key:

```
# hammer activation-key product-content \
--name "development-stack" \
--organization "ACME"
```

6. Override the default auto-enable status for the Red Hat Satellite Tools 6.4 repository. The default status is set to disabled. To enable, enter the following command:

```
# hammer activation-key content-override \
--name "development-stack" \
--content-label rhel-7-server-satellite-tools-6.4-rpms \
--value 1 \
--organization "ACME"
```

9.2. USING ACTIVATION KEYS

The activation keys are used for registration. This includes:

- Registering new systems during provisioning through Red Hat Satellite 6. The kickstart provisioning templates in Red Hat Satellite 6 contain commands to register the system using an activation key defined when creating a new host.
- Registering existing Red Hat Enterprise Linux systems. Configure Red Hat Subscription Manager to use Satellite Server for registration and specify the activation key when running the **`subscription-manager register`** command.

You can test the activation key we created. Register an existing Red Hat Enterprise Linux 7 system to Satellite Server.

First, download the consumer RPM for your Satellite Server. This is usually located in the **pub** directory on the host's web server. For example, for a Satellite Server with the host name **`satellite.example.com`**, enter the following command on the client to register:

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

This RPM installs the necessary certificates for accessing repositories on Satellite Server and configures Red Hat Subscription Manager to use the server's URL.

Next, run Red Hat Subscription Manager on the client:

```
# subscription-manager register --activationkey="development-stack" \
--org="ACME"
The system has been registered with id: 744fb31c-c983-00f5-ca14-bddd0f711353
```

Check Satellite Server to confirm the registration.

For Web UI Users

Navigate to **Hosts** > **Content Hosts** and the system appears in the list.

For CLI Users

Enter the following command:

```
# hammer host list --organization "ACME"
```

IMPORTANT

After registering a client system to Satellite Server, install the **`katello-agent`** package on the system so that it can report back to Satellite Server:

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

The Red Hat Satellite Tools 6.4 repository provides this package.

You can use multiple activation keys when registering a content host. This allows you to create activation keys for specific subscription sets and then combine them according to content host requirements. For example, the following command registers a content host to the ACME organization with both VDC and OpenShift subscriptions:

```
# subscription-manager register --org="ACME" \
--activationkey="ak-VDC,ak-OpenShift"
```

If there are conflicting settings in activation keys, the rightmost key takes precedence.

- Settings that conflict: **Service Level**, **Release Version**, **Environment**, **Content View**, and **Product Content**.
- Settings that do not conflict and the host gets the union of them: **Subscriptions** and **Host Collections**.
- Settings that influence the behavior of the key itself and not the host configuration: **Content Host Limit** and **Auto-Attach**.

9.3. UPDATING SUBSCRIPTIONS ASSOCIATED WITH AN ACTIVATION KEY

You can change the subscriptions associated with an Activation Key using the web UI or using the Hammer command-line tool. The Hammer CLI method requires downloading a CSV file, making changes to the Activation Key settings, and then uploading the changed CSV file.

Note that changes to an Activation Key applies only to machines provisioned after the change. To update subscriptions on existing content hosts, see Section 4.2.6, “Bulk Updating Content Hosts' Subscriptions”.

For Web UI Users

1. Navigate to **Content** > **Activation keys** and click the name of the Activation Key.
2. Select the **Subscriptions** tab.
3. To remove subscriptions, select **List**/**Remove**, select the check boxes to the left of the subscriptions to be removed and then click **Remove Selected**.
4. To add subscriptions, select **Add**, select the check boxes to the left of the subscriptions to be added and then click **Add Selected**.
5. Select **Repository Sets** and review the repositories' status settings.

6. To enable or disable a repository, select the check box for a repository and then change the status using the **Select Action** list.

7. Select the **Details** tab, select a Content View for this Activation Key, and then click **Save**.

For CLI Users

1. Export the subscriptions from Satellite Server to a CSV file. For this example, **`a_keys.csv`**:

```
# hammer --server https://satellite.example.com csv activation-keys \
--export --file a_keys.csv --organization "Default Organization"
```

2. You can view the columns of the file as follows:

```
# column -s, -t < a_keys.csv | less -S
Name  Organization  Description  Limit  Environment  Content View
Host Collections  Auto-Attach  Service Level  Release Version
Subscriptions
```

The last column has the subscription information. Note that it is one field in the CSV file and can include a comma in quoted text. Examples of the **Subscriptions** field include:

- """Automatic|RH1234|Red Hat Enterprise Linux Server, Standard (Physical or Virtual Nodes)|11223344|55667788"""

- '1|MCT0369|Red Hat Satellite Capsule Server|11223344|55667788'
  The **Subscriptions** field has the following format:

  - Number of subscriptions allocated (can be set to **Automatic**).
  - The subscription's identification number.
  - The subscription's name.
  - The contract number.
  - The account number.

3. Change the required values in the CSV file. You can use an editor, with a CSV plug-in, or **sed** to change strings in the **Subscriptions** field.

   a. Make a backup of the file.

   ```
   # cp a_keys.csv a_keys.csv.backup
   ```

   b. Edit the string you want to change. For example:

   ```
   # sed -i "s/Automatic|RH1234|Red Hat Enterprise Linux
   Server/Automatic|RH4567|Red Hat Enterprise Linux Server/g"
   a_keys.csv
   ```

   c. Confirm only the required changes were made. For example:

   ```
   # diff a_keys.csv a_keys.csv.backup
   ```

```
2c2
< """Automatic|RH4567|Red Hat Enterprise Linux Server, Standard
(Physical or Virtual Nodes)|11223344|55667788"""
---
> """Automatic|RH1234|Red Hat Enterprise Linux Server, Standard
(Physical or Virtual Nodes)|11223344|55667788"""
```

4. Upload the changed file to Satellite Server.

```
# hammer --server https://satellite.example.com csv activation-keys
\
--file a_keys.csv
```

9.4. ENABLING AUTO-ATTACH

When auto-attach is enabled on an activation key and there are subscriptions associated with the key, the subscription management service selects and attaches the best-matched associated subscriptions based on a set of criteria like currently-installed products, architecture, and preferences like service level.

You can enable auto-attach and have no subscriptions associated with the key. This type of key is commonly used to register virtual machines when you do not want the virtual machine to consume a RHEL subscription but to inherit a RHEL Virtual Data Center (VDC) subscription from the hypervisor.

Auto-attach is enabled by default. Disable the option if you want to force attach all subscriptions associated with the activation key.

For Web UI Users

1. Click **Content** > **Activation Keys**.
2. Click the activation key name that you want to edit.
3. Click the **Subscriptions** tab.
4. Click the edit icon next to **Auto-Attach**.
5. Select or clear the check box to enable or disable auto-attach.
6. Click **Save**.

> **NOTE**
>
> To register virtual content hosts to Satellite Server using an auto-attach activation key, first use the **`virt-who`** utility to map those hosts to a hypervisor entitled with the Virtual Datacenter (VDC) subscription. Without this prerequisite, virtual hosts are registered only with a temporary virtual subscription for 24 hours. For more information, see Applying Virtual Guest Subscriptions in the *Virtual Instances Guide*.

For CLI Users

To enable auto-attach on the activation key **`development-stack`**:

```
# hammer activation-key update --name "development-stack" \
--organization "ACME" --auto-attach true
```

9.5. SETTING THE SERVICE LEVEL

An activation key can be configured to define a default service level for the new host created with the activation key. Setting a default service level selects only the matching subscriptions to be attached to the host. For example, if the default service level on an activation key is set to Premium, only subscriptions with premium service levels are attached to the host upon registration.

For Web UI Users

1. Click **Content** > **Activation Keys**.
2. Click the activation key name you want to edit.
3. Click the edit icon next to **Service Level**.
4. Select the required service level from the list. The list only contains service levels available to the activation key.
5. Click **Save**.

For CLI Users

To set a default service level to Premium on the activation key **`development-stack`**:

```
# hammer activation-key update --name "development-stack" \
--organization "ACME" --service-level premium
```

CHAPTER 10. MANAGING ERRATA

As a part of Red Hat's quality control and release process, we provide customers with updates for each release of official Red Hat RPMs. Red Hat compiles groups of related package into an **erratum** along with an advisory that provides a description of the update. There are three types of advisories (in order of importance):

Security Advisory

Describes fixed security issues found in the package. The security impact of the issue can be Low, Moderate, Important, or Critical.

Bug Fix Advisory

Describes bug fixes for the package.

Product Enhancement Advisory

Describes enhancements and new features added to the package.

Red Hat Satellite 6 imports this errata information when synchronizing repositories with Red Hat's Content Delivery Network (CDN). Red Hat Satellite 6 also provides tools to inspect and filter errata, allowing for precise update management. This way, you can select relevant updates and propagate them through Content Views to selected content hosts.

Errata are labeled according to the most important advisory type they contain. Therefore, errata labeled as **Product Enhancement Advisory** can contain only enhancement updates, while **Bug Fix Advisory** errata can contain both bug fixes and enhancements, and **Security Advisory** can contain all three types.

In Red Hat Satellite, there are two keywords that describe an erratum's relationship to the available content hosts:

Applicable

Erratum applies to one or more content hosts, which means it updates packages present on the content host. Applicable errata are not yet accessible by the content host.

Installable

Erratum applies to one or more content hosts and it has been made available to the content host. Installable errata are present in the content host's life cycle environment and Content View, but are not yet installed. This way, errata can be installed by users who have permissions to manage content hosts, but are not entitled for errata management at higher levels.

This chapter shows how to manage errata and apply them to either a single system or multiple systems.

10.1. INSPECTING AVAILABLE ERRATA

The following procedure describes how to view and filter the available errata and how to display metadata of the selected advisory.

1. Navigate to **Content** > **Errata** to view the list of available errata.

2. Use the filtering tools at the top of the page to limit the number of displayed errata:

   - Select the repository to be inspected from the list. **All Repositories** is selected by default.

   - The **Applicable** check box is selected by default to view only errata applicable to the selected repository. Select the **Installable** check box to view only errata marked as installable.

- To search the table of errata, type the query in the **Search** field in the form of:

  ```
  parameter operator value
  ```

  See Table 10.1, “Parameters Available for Errata Search” for the list of parameters available for search. Find the list of applicable operators in Supported Operators for Granular Search in *Administering Red Hat Satellite*. Automatic suggestion works as you type. You can also combine queries with the use of **and** and **or** operators. For example, to display only security advisories related to the **kernel** package, type:

  ```
  type = security and package_name = kernel
  ```

  Press **Enter** to start the search.

3. Click the **Errata ID** of the erratum you want to inspect:

   - The **Details** tab contains the description of the updated package as well as documentation of important fixes and enhancements provided by the update.
   - On the **Content Hosts** tab, you can apply the erratum to selected content hosts as described in Section 10.5, “Applying Errata to Multiple Systems”.
   - The **Repositories** tab lists repositories that already contain the erratum. You can filter repositories by the environment and Content View, and search for them by the repository name.

Table 10.1. Parameters Available for Errata Search

Parameter	Description	Example
bug	Search by the Bugzilla number.	*bug = 1172165*
cve	Search by the CVE number.	*cve = CVE-2015-0235*
id	Search by the errata ID. The auto-suggest system displays a list of available IDs as you type.	*id = RHBA-2014:2004*
issued	Search by the issue date. You can specify the exact date, like "Feb16,2015", or use keywords, for example "Yesterday", or "1 hour ago". The time range can be specified with the use of the "<" and ">" operators.	*issued < "Jan 12,2015"*
package	Search by the full package build name. The auto-suggest system displays a list of available packages as you type.	*package = glib2-2.22.5-6.el6.i686*

Parameter	Description	Example
package_name	Search by the package name. The auto-suggest system displays a list of available packages as you type.	*package_name = glib2*
severity	Search by the severity of the issue fixed by the security update. Specify *Critical*, *Important*, or *Moderate*.	*severity = Critical*
title	Search by the advisory title.	*title ~ openssl*
type	Search by the advisory type. Specify *security*, *bugfix*, or *enhancement*.	*type = bugfix*
updated	Search by the date of the last update. You can use the same formats as with the **`issued`** parameter.	*updated = "6 days ago"*

10.2. SUBSCRIBING TO ERRATA NOTIFICATIONS

You can configure email notifications for Satellite users. Users receive a summary of applicable and installable errata, notifications on Content View promotion or after synchronizing a repository. For more information, see the Configuring Email Notifications section in the *Administering Red Hat Satellite* guide.

10.3. MANAGING ERRATA WITH CONTENT VIEWS

Red Hat Satellite 6 provides various methods to manage and apply errata. You can use Content Views and content filters to limit errata. Such filters include:

- **ID** - Select specific erratum to allow into your resulting repositories.
- **Date Range** - Define a date range and include a set of errata released during that date range.
- **Type** - Select the type of errata to include such as bug fixes, enhancements, and security updates.

10.3.1. Creating a Content View Filter for Errata

Create a content filter to exclude errata after a certain date. This ensures your production systems in the application life cycle are kept up to date to a certain point. Then you can modify the filter's start date to introduce new errata into your testing environment to test the compatibility of new packages into your application life cycle.

Prerequisites

- A Content View with the repositories that contain required errata is created. For more information, see Section 8.2.1, “Creating a Simple Content View”.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Content Views** and select a Content View that you want to use for applying errata.
2. Navigate to **Yum Content** > **Filters** and click **New Filter**.
3. In the **Name** field, enter **`Errata Filter`**.
4. From the **Content Type** list, select **Erratum - Date and Type**.
5. From the **Inclusion Type** list, select **Exclude**.
6. In the **Description** field, enter **`Exclude errata items from YYYY-MM-DD`**.
7. Click **Save**.
8. For **Errata Type**, select the check boxes of errata types you want to exclude. For example, select the **Enhancement** and **Bugfix** check boxes and clear the **Security** check box to exclude enhancement and bugfix errata after certain date, but include all the security errata.
9. For **Date Type**, select one of two check boxes:
   - **Issued On** for the issued date of the erratum.
   - **Updated On** for the date of the erratum’s last update.
10. Select the **Start Date** to exclude all errata on or after the selected date.
11. Leave the **End Date** field blank.
12. Click **Save**.
13. Click **Publish New Version** to publish the resulting repository.
14. Enter **`Adding errata filter`** in the **Description** field.
15. Click **Save**.
    When the Content View completes publication, notice the **Content** column reports a reduced number of packages and errata from the initial repository. This means the filter successfully excluded the all non-security errata from the last year.
16. Click the **Versions** tab.
17. Click **Promote** to the right of the published version.
18. Select the environments you want to promote the Content View version to.
19. In the **Description** field, enter the description for promoting.
20. Click **Promote Version** to promote this Content View version across the required environments.

For CLI Users

1. Create a filter for the errata:

```
# hammer content-view filter create --name "Filter Name" \
--description "Exclude errata items from the YYYY-MM-DD" \
--content-view "CV Name" --organization "Default Organization" \
--type "erratum"
```

2. Create a filter rule to exclude all errata on or after the *Start Date* that you want to set:

```
# hammer content-view filter rule create --start-date "YYYY-MM-DD" \
--content-view "CV Name" --content-view-filter="Filter Name" \
--organization "Default Organization" --
types=security,enhancement,bugfix
```

3. Publish the Content View:

```
# hammer content-view publish --name "CV Name" \
--organization "Default Organization"
```

4. Promote the Content View to the lifecycle environment so that the included errata are available to that lifecycle environment:

```
# hammer content-view version promote \
--content-view "CV Name" \
--organization "Default Organization" \
--to-lifecycle-environment "Lifecycle Environment Name"
```

10.4. APPLYING ERRATA TO INDIVIDUAL SYSTEMS

Use these procedures to review and apply errata to individual systems.

Prerequisites

- Synchronize Red Hat Satellite repositories with the latest errata available from Red Hat. For more information, see Section 5.6, “Synchronizing Red Hat Repositories”.
- Register the client system to an environment and Content View on Satellite Server. For more information, see Section 8.6, “Registering Systems to Environments and their Content Views”.
- Install the **`katello-agent`** package on the client system. For more information, see the Installing the Katello Agent section in the *Managing Hosts* guide.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **Content Hosts** and select the host you want to apply errata to.
2. Navigate to the **Errata** tab to see the list of errata.
3. Select the errata to apply and click **Apply Selected**. In the confirmation window, click **Apply**.
4. After the task to update all packages associated with the selected errata completes, click the **Details** tab to view the updated packages.

For CLI Users

1. List all errata for the client system:

```
# hammer host errata list \
--host client.example.com
```

2. Apply the most recent erratum to the client system. Identify the erratum to apply using the erratum ID:

```
# hammer host errata apply --host "Host Name" \
--errata-ids ERRATUM_ID1,ERRATUM_ID2...
```

10.5. APPLYING ERRATA TO MULTIPLE SYSTEMS

Use these procedures to review and apply errata to multiple systems.

Prerequisites

- Synchronize Red Hat Satellite repositories with the latest errata available from Red Hat. For more information, see Section 5.6, “Synchronizing Red Hat Repositories”.
- Register the client systems to an environment and Content View on Satellite Server. For more information, see Section 8.6, “Registering Systems to Environments and their Content Views”.
- Install the **katello-agent** package on client systems. For more information, see the Installing the Katello Agent section in the *Managing Hosts* guide.

Procedure

1. Navigate to **Content** > **Errata**.
2. Click the name of an erratum you want to apply.
3. Click to **Content Hosts** tab.
4. Select the systems you want to apply errata to and click **Apply to Hosts**.
5. Click **Confirm**.

For CLI Users

Although the CLI does not have the same tools as the Web UI, you can replicate a similar procedure with CLI commands.

1. List all installable errata:

```
# hammer erratum list \
--errata-restrict-installable true \
--organization "Default Organization"
```

2. Select the erratum you want to use and list the systems that this erratum is applicable to:

```
# hammer host list \
--search "applicable_errata = ERRATUM_ID" \
--organization "Default Organization"
```

3. Apply the errata to a single system:

```
# hammer host errata apply \
--host client.example.com \
--organization "Default Organization" \
--errata-ids ERRATUM_ID1,ERRATUM_ID2...
```

4. Enter the following command for each client system and replace **$HOST** with the name of the system for each execution.

```
# for HOST in `hammer \
--csv --csv-separator "|" host list \
--search "applicable_errata = ERRATUM_ID" \
--organization "Default Organization" | tail -n+2 | awk \
-F "|" '{ print $2 }'` ; do echo \
"== Applying to $HOST ==" ; hammer host errata apply \
--host $HOST --errata-ids ERRATUM_ID1,ERRATUM_ID2 ; done
```

This command identifies all hosts with *erratum_IDs* as an applicable erratum and then applies the erratum to each host.

CHAPTER 11. MANAGING OSTREE CONTENT

OSTree is a tool to manage bootable, immutable, versioned file system trees. You can use a custom OSTree content on a build system, then export an OSTree repository to a static HTTP. Red Hat Enterprise Linux Atomic Server uses OSTree content composed from RPM files as a method to keep the operating system up to date.

You can use Red Hat Satellite 6 to synchronize and manage OSTree branches from an OSTree repository.

In Satellite Server 6.4, OSTree management tools are enabled by default. If you ever have a reason to enable the tool, enter the following command:

```
# satellite-installer --katello-enable-ostree=true
```

11.1. SELECTING RED HAT OSTREE CONTENT TO SYNCHRONIZE

Red Hat CDN provides OSTree Content for you to select and synchronize.

Procedure

To find and synchronize OSTree content, complete the following steps:

1. In the Satellite web UI, navigate to **Content** > **Red Hat Repositories**.
2. From the list, select the **OSTree** content type.
3. In the Available Repositories pane, locate the OSTree repisotry set you want to use, for example, the **Red Hat Enterprise Linux Atomic Host Trees** set from the **Red Hat Enterprise Linux Atomic Host** product group.
4. Click the **Enable** icon to enable the repository you want to use.
5. Navigate to **Content** > **Products** and click the product that you want to use, for example **Red Hat Enterprise Linux Atomic Host**.
6. Select this repository and click **Sync Now**.

To view the Synchronization Status

- In the Satellite web UI, navigate to **Content** > **Sync Status** and expand, for example, **Red Hat Enterprise Linux Atomic Host**.

For CLI Users

1. Search the Red Hat Enterprise Linux Server product for **`ostree`** repositories:

```
# hammer repository-set list \
--product "Red Hat Enterprise Linux Atomic Host" \
--organization "My_Organization" | grep "ostree"
```

2. Enable the **`ostree`** repository for Red Hat Enterprise Linux Atomic Host or any product that you want to use:

```
# hammer repository-set enable \
```

```
--product "Red Hat Enterprise Linux Atomic Host" \
--name "Red Hat Enterprise Linux Atomic Host (Trees)" \
--organization "My_Organization"
```

3. Locate and synchronize the repository for the product:

```
# hammer repository list \
--product "Red Hat Enterprise Linux Atomic Host" \
--organization "My_Organization"
# hammer repository synchronize \
--name "Red Hat Enterprise Linux Atomic Host Trees" \
--product "Red Hat Enterprise Linux Atomic Host" \
--organization "My_Organization"
```

11.2. IMPORTING CUSTOM OSTREE CONTENT

In addition to importing OSTree content from Red Hat CDN, you can also import content from other sources. This requires a published HTTP location for the OSTree to import.

Procedure

To import custom OSTree content, complete the following steps:

1. In the Satellite web UI, navigate to **Content** > **Products** and click **New Product**.
2. In the **Name** field, enter a name for your OSTree content. This automatically populates the **Label** field.
3. Optional: In the **GPG Key** field, enter a GPG Key for the entire product.
4. In the **Sync Plan** field, enter the name of a content synchronization plan to associate with the product.
5. In the **Description** field, enter a description of the product and click **Save**.
6. When the product creation completes, click **Create Repository**.
7. In the **Name** field, enter a name for the repository. This automatically populates the **Label** field.
8. From the **Type** list, select `ostree`.
9. In the **URL** field, enter the URL of the registry to use as a source. For example `http://www.example.com/rpm-ostree/`.
10. Click **Save**.
11. When the repository creation completes, select the new repository and click **Sync Now** to start the synchronization process.

To view the Synchronization Status:

- In the Satellite web UI, navigate to **Content** > **Sync Status** and expand the entry that you want to view.

For CLI Users

1. Create the custom **`OSTree Content`** product:

```
# hammer product create \
--name "Custom OSTree Content" \
--sync-plan "Example_Plan" \
--description "OSTree Content" \
--organization "My_Organization"
```

2. Create the repository for the OSTree:

```
# hammer repository create \
--name "Custom OSTree" \
--content-type "ostree" \
--url "http://www.example.com/rpm-ostree/" \
--product "OSTree Content" \
--organization "My_Organization"
```

3. Synchronize the repository:

```
# hammer repository synchronize \
--name "Custom OStree" \
--product "OSTree Content" \
--organization "My_Organization"
```

11.3. MANAGING OSTREE CONTENT WITH CONTENT VIEWS

Use Content Views to manage OSTree branches across the application life cycle. This process uses the same publication and promotion method that RPMs and Puppet modules use.

Procedure

To create a content view for your OSTree and add a repository, complete the following steps:

1. In the Satellite web UI, navigate to **Content** > **Content Views** and click **Create New View**.
2. In the **Name** field, enter a plain text name for the view. This automatically populates the **Label** field.
3. In the **Description** field, enter a description of the OSTree Content View.
4. If you want to use a Composite Content View, select the **Composite View** check box.
5. Click **Save** to complete.
6. Navigate to the **OSTree Content** tab, then click **Add**.
7. Select the OSTree repository for that you want to use. Click **Add Repository** to add the OSTree content from this repository to the Content View.
8. Navigate to **Versions** and click **Publish New Version**.
9. In the **Description** field, enter a description for the version, and click **Save**.

You can also click **Promote** to promote this Content View across environments in the application life cycle.

For CLI Users

1. Obtain a list of repository IDs:

```
# hammer repository list --organization "_My_Organization_"
```

2. Create the Content View and add the repository:

```
# hammer content-view create \
--name "OSTree" \
--description "OSTree for Red Hat Enterprise Linux Atomic Host" \
--repository-ids 5 \
--organization "My_Organization"
```

3. Publish the view:

```
# hammer content-view publish \
--name "OSTree" \
--description "Example Content View for the OSTree" \
--organization "My_Organization"
```

CHAPTER 12. MANAGING ISO IMAGES

You can use Red Hat Satellite 6 to store ISO images, either from Red Hat's Content Delivery Network or other sources. You can also upload other files, such as virtual machine images, and publish them in repositories.

12.1. IMPORTING ISO IMAGES FROM RED HAT

The Red Hat Content Delivery Network provides ISO images for certain products. The procedure for importing this content is similar to the procedure for enabling repositories for RPM content.

Procedure

To import Red Hat ISO images, complete the following steps:

1. In the Satellite web UI, navigate to **Content** > **Red Hat Repositories**.
2. In the **Search** field, enter an image name, for example, **`Red Hat Enterprise Linux 7 Server (ISOs)`**.
3. In the Available Repositories window, expand **Red Hat Enterprise Linux 7 Server (ISOs)**.
4. For the **x86_64 7.2** entry, click the **Enable** icon to enable the repositories for the image.
5. Navigate to **Content** > **Products** and click **Red Hat Enterprise Linux Server**.
6. Click the **Repositories** tab of the Red Hat Enterprise Linux Server window, and click **Red Hat Enterprise Linux 7 Server ISOs x86_64 7.2**.
7. In the upper right of the Red Hat Enterprise Linux 7 Server ISOs x86_64 7.2 window, click **Select Action** and select **Sync Now**.

To view the Synchronization Status

- In the web UI, navigate to **Content** > **Sync Status** and expand **Red Hat Enterprise Linux Server**.

For CLI Users

1. Locate the Red Hat Enterprise Linux Server product for **`file`** repositories:

```
# hammer repository-set list \
--product "Red Hat Enterprise Linux Server" \
--organization "My_Organization" | grep "file"
```

2. Enable the **`file`** repository for Red Hat Enterprise Linux 7.2 Server ISO:

```
# hammer repository-set enable \
--product "Red Hat Enterprise Linux Server" \
--name "Red Hat Enterprise Linux 7 Server (ISOs)" \
--releasever 7.2 \
--basearch x86_64 \
--organization "My_Organization"
```

3. Locate and synchronize the repository in the product:

```
# hammer repository list \
--product "Red Hat Enterprise Linux Server" \
--organization "My_Organization"
# hammer repository synchronize \
--name "Red Hat Enterprise Linux 7 Server ISOs x86_64 7.2" \
--product "Red Hat Enterprise Linux Server" \
--organization "My_Organization"
```

12.2. IMPORTING INDIVIDUAL ISO IMAGES AND FILES

Use this procedure to manually import ISO content and other files to Satellite Server. The procedure is similar to uploading custom Puppet modules procedure:

1. Create a custom product.
2. Add a repository for files to the product.
3. Select a file to upload to the repository.

Procedure

To import custom ISO images, complete the following steps:

1. In the Satellite web UI, navigate to **Content** > **Products**, and in the Products window, click **New Product**.
2. In the **Name** field, enter a name to identify the product. This name populates the **Label** field.
3. In the **GPG Key** field, enter a GPG Key for the product.
4. From the **Sync Plan** list, select a synchronization plan for the product.
5. In the **Description** field, enter a description of the product.
6. Click **Save**.
7. In the Products window, click the new product and then click **Create Repository**.
8. In the **Name** field, enter a name for the repository. This automatically populates the **Label** field.
9. From the **Type** list, select **file**.
10. In the **Upstream URL** field, enter the URL of the registry to use as a source. Add a corresponding user name and password in the **Upstream Username** and **Upstream Password** fields.
11. Click **Save**.
12. Click the new repository.
13. Navigate to the **Upload File** and click **Browse**.
14. Select the `.iso` file and click **Upload**.

For CLI Users

1. Create the custom product:

```
# hammer product create \
--name "My_ISOs" \
--sync-plan "Example Plan" \
--description "My_Product" \
--organization "My_Organization"
```

2. Create the repository:

```
# hammer repository create \
--name "My_ISOs" \
--content-type "file" \
--product "My_Product" \
--organization "My_Organization"
```

3. Upload the ISO file to the repository:

```
# hammer repository upload-content \
--path ~/bootdisk.iso \
--name "My_ISOs" \
--product "My_Product" \
--organization "My_Organization"
```

APPENDIX A. USING AN NFS SHARE FOR CONTENT STORAGE

Your environment requires adequate hard disk space to fulfill content storage. In some situations, it is useful to use an NFS share to store this content. This appendix shows how to mount the NFS share on your Satellite Server's content management component.

IMPORTANT

Do not mount the full **/var/lib/pulp** on an NFS share. Parts of Satellite Server use transient SQLite databases, which have issues over NFS. Red Hat recommends the use of high-bandwidth, low-latency storage for the **/var/lib/pulp** file system. Red Hat Satellite has many operations that are IO-intensive so usage of high-latency, low-bandwidth storage could potentially have issues with performance degradation. Only use the NFS share for the **/var/lib/pulp/content** directory.

1. Create the NFS share. This example uses a share at **nfs.example.com:/satellite/content**. Ensure this share provides the appropriate permissions to Satellite Server and its **apache** user.

2. Shutdown the Satellite services on the Satellite host:

   ```
   # foreman-maintain service stop
   ```

3. Ensure Satellite Server has the **nfs-utils** package installed:

   ```
   # yum install nfs-utils
   ```

4. You need to copy the existing contents of **/var/lib/pulp/content** to the NFS share. First, mount the NFS share to a temporary location:

   ```
   # mkdir /mnt/temp
   # mount -o rw nfs.example.com:/satellite/content /mnt/temp
   ```

   Copy the existing contents of **/var/lib/pulp/content** to the temporary location:

   ```
   # cp -r /var/lib/pulp/content/* /mnt/temp/.
   ```

5. Set the permissions for all files on the share to use the **apache** user. This ID of this user is usually 48.

6. Unmount the temporary storage location:

   ```
   # umount /mnt/temp
   ```

7. Remove the existing contents of **/var/lib/pulp/content**:

   ```
   # rm -rf /var/lib/pulp/content/*
   ```

8. Edit the **/etc/fstab** file and add the following line:

```
nfs.example.com:/satellite/content    /var/lib/pulp/content   nfs
rw,hard,intr,context="system_u:object_r:httpd_sys_rw_content_t:s0"
```

This makes the mount persistent across system reboots. Ensure to include the SELinux context.

9. Enable the mount:

```
# mount -a
```

10. Confirm the NFS share mounts to **var/lib/pulp/content**:

```
# df
Filesystem                         1K-blocks     Used Available
Use% Mounted on
...
nfs.example.com:/satellite/content 309506048 58632800 235128224  20%
/var/lib/pulp/content
...
```

Also confirm that the existing content exists at the mount on **var/lib/pulp/content**:

```
# ls /var/lib/pulp/content
```

11. Start the Satellite services on the Satellite host:

```
# foreman-maintain service start
```

Satellite Server now uses the NFS share to store content. Run a content synchronization (see Section 5.3, “Synchronizing Content”) to ensure the NFS share works as expected.

APPENDIX B. IMPORTING CONTENT ISOS INTO A DISCONNECTED SATELLITE

In high security environments where hosts are required to function in a closed network disconnected from the Internet, Satellite Server can provision systems with the latest security updates, errata, and packages. To accomplish this, download the Content ISOs for Red Hat Satellite from the Red Hat Customer Portal and import them into Satellite Server.

IMPORTANT

This section is not required if your Satellite Server is connected to the Internet.

Download the product ISO from the Red Hat Customer Portal, as follows:

1. Navigate to Downloads (at the very top of the window) and select Red Hat Satellite.
2. Open the Content ISOs tab. All products in your subscription are listed here.
3. Click the link for the product name, such as Red Hat Enterprise Linux 6 Server (x86_64) to download the ISO.
4. Copy all of Satellite Content ISOs to a directory Satellite can access. This example uses **`/root/isos`**.
5. Create a local directory that are shared through httpd on the Satellite. This example uses **`/var/www/html/pub/sat-import/`**.

   ```
   # mkdir -p /var/www/html/pub/sat-import/
   ```

6. Mount and recursively copy the contents of the first ISO to the local directory:

   ```
   # mkdir /mnt/iso
   # mount -o loop /root/isos/first_iso /mnt/iso
   # cp -ruv /mnt/iso/* /var/www/html/pub/sat-import/
   # umount /mnt/iso
   # rmdir /mnt/iso
   ```

7. Repeat the above step for each ISO until you have copied all the data from the Content ISOs into **`/var/www/html/pub/sat-import/`**.
8. Ensure the SELinux contexts for the directory are correct:

   ```
   # restorecon -rv /var/www/html/pub/sat-import/
   ```

9. Satellite Server now contains the content from the Content ISOs. However, Satellite Server needs to point to this location as the CDN URL. In the Satellite Web UI, navigate to **Content** > **Red Hat Subscriptions**. .
10. Click **Manage Manifest**.
11. On the Subscription Manifest information screen, select the **Actions** tab.
12. Navigate to Red Hat Provider Details. Click the edit icon on the **Red Hat CDN URL** and change the URL to the Satellite host name with the newly created directory, for example:

`http://server.example.com/pub/sat-import/`

13. Click **Save** and then upload your manifest using Section 4.1.8, “Importing a Subscription Manifest into Satellite Server”.

Satellite is now acting as its own CDN with the files located in `http://server.example.com/pub/sat-import/`. This is not a requirement. The CDN can be hosted on a different machine inside the same disconnected network as long as it is accessible to Satellite Server using HTTP.

If your environment changes from disconnected to connected, you can reconfigure a disconnected Satellite to pull content directly from Red Hat Customer Portal:

1. In the Satellite Web UI, navigate to **Content** > **Red Hat Subscriptions**.
2. Click **Manage Manifest**.
3. On the Subscription Manifest information screen, select the **Actions** tab.
4. Navigate to Red Hat Provider Details. Click the edit icon on the **Red Hat CDN URL** and change the URL to the Red Hat CDN URL:
   `https://cdn.redhat.com`
5. Click **Save**

Satellite Server pulls content directly from Red Hat Customer Portal on the next synchronization.

APPENDIX C. IMPORTING CONTENT ISOS INTO A CONNECTED SATELLITE

Even if Satellite Server can connect directly to the Red Hat Customer Portal, you can perform the initial synchronization from locally mounted content ISOs. When the initial synchronization is completed from the content ISOs, you can switch back to downloading content through the network connection. To accomplish this, download the Content ISOs for Red Hat Satellite from the Red Hat Customer Portal and import them into Satellite Server. For locations with bandwidth limitations, using an **On Demand** or **Background** download policy might be more efficient than downloading and importing Content ISOs.

IMPORTANT

This section is not required if your Satellite Server is connected to the Internet.

This example shows how to perform the first synchronization of the Red Hat Enterprise Linux 6 repository from content ISOs. At the time of writing there are 21 DVD size ISO files.

Downloading the content ISOs from the Red Hat Customer Portal

1. In your browser, navigate to Red Hat Customer Portal and log in.
2. Click **DOWNLOADS**.
3. Select **Red Hat Satellite**.
4. Select the **Content ISOs** tab. All products in your subscription are listed there.
5. Search for the section required, in this example **Red Hat Enterprise Linux 6**.
6. Click the link for the product name, such as **`RHEL 6 Server (x86_64)(2017-04-14T01:27:00)`** to reveal the ISO files.
7. Using your browser, download the required ISOs to a location accessible by your browser. For example, to your workstation's **`Downloads`** directory.

Importing the Content ISOs

1. In a terminal connected to Satellite Server, create a directory to act as a temporary store for all of the required Satellite Content ISOs. This example uses **`/tmp/isos/rhel6`**:

```
# mkdir -p /tmp/isos/rhel6
```

2. On your workstation, copy the ISO files to Satellite Server:

```
$ scp ~/Downloads/iso_file
root@satellite.example.com:/tmp/isos/rhel6
```

3. On Satellite Server, create a directory to serve as a mount point for the ISOs:

```
# mkdir /mnt/iso
```

4. Create a working directory to hold the contents of all the ISOs:

```
# mkdir /mnt/rhel6
```

5. Mount and recursively copy the contents of the first ISO to the working directory:

```
# mount -o loop /tmp/isos/iso_file /mnt/iso
# cp -ruv /mnt/iso/* /mnt/rhel6/
# umount /mnt/iso
```

6. Repeat the above step for each ISO until you have copied all the data from the Content ISOs into **`/mnt/rhel6`**.

7. If required, remove the empty directory used as the mount point:

```
# rmdir /mnt/iso
```

8. If required, remove the temporary working directory and its contents to regain the space:

```
# rm -rf /tmp/isos/
```

Performing the Initial Synchronization

1. Set the owner and the SELinux context for the directory and its contents to be the same as **`/var/lib/pulp`**:

```
# chcon -R --reference /var/lib/pulp  /mnt/rhel6/
# chown -R apache:apache /mnt/rhel6/
```

2. Create or edit the **`/etc/pulp/content/sources/conf.d/local.conf`** file. Insert the following text into the file:

```
[rhel-6-server]
enabled: 1
priority: 0
expires: 3d
name: Red Hat Enterprise Linux 6 Server
type: yum
base_url:
file:///mnt/rhel6/content/dist/rhel/server/6/6Server/x86_64/os/
```

The **`base_url`** path might differ in your content ISO. The directory specified in **`base_url`** must contain the **`repodata`** directory, otherwise the synchronization fails. To synchronize multiple repositories, create a separate entry for each of them in the configuration file **`/etc/pulp/content/sources/conf.d/local.conf`**.

3. In the Satellite web UI, navigate to **Content** > **Red Hat Repositories** and enable the following repository: **Red Hat Enterprise Linux 6 Server RPMs x86_64 6Server**.

4. Under **Content** > **Sync Status** select the repository to be synchronized and click **Synchronize Now**.

Note that there is no indication in the Satellite web UI of which source is being used. In case of problems with a local source, Satellite pulls content through the network. To monitor the process, enter the following command in a terminal (limited to Red Hat Enterprise Linux 7 base systems):

```
# journalctl -f -l SYSLOG_IDENTIFIER=pulp | grep -v worker[\-,\.]heartbeat
```

The above command displays interactive logs. First, Satellite Server connects to the Red Hat Customer Portal to download and process repository metadata. Then, the local repository is loaded. In case of any errors, cancel the synchronization in the Satellite web UI and verify your configuration.

After successful synchronization you can detach the local source by removing its entry from **/etc/pulp/content/sources/conf.d/local.conf**.

APPENDIX D. SYNCHRONIZING CONTENT BETWEEN SATELLITE SERVERS

Red Hat Satellite 6.4 uses Inter-Satellite Synchronization (ISS) to synchronize content between upstream and downstream servers. In the context of ISS, upstream refers to the server from which content is exported; downstream refers to the server into which content is imported.

ISS is designed to address two scenarios:

- If you have both connected and disconnected Satellite Servers, and want to propagate content from the connected servers to the disconnected servers.
- If you have a primary Satellite Server and want to propagate some, but not all, content to other Satellite Servers. For example, you might have Content Views (CVs) that are validated by the IT department, and you want to propagate the **`yum`** content from those CVs to a downstream Satellite.

NOTE

Be aware that you cannot use ISS to synchronize content from a Satellite Server to a Capsule Server. Capsule Server supports synchronization natively. For more information, see Capsule Server Overview in *Planning for Red Hat Satellite 6*.

Satellite 6.4 supports exports as a set of directories (default) or as ISO files. You can then import the resulting export to another Satellite Server. This replaces the **`katello-disconnected`** script in earlier Satellite versions, which exported repositories into a directory structure that could be later imported into another Satellite Server. In Satellite 6.4, all export and import functions are performed on the command line.

NOTE

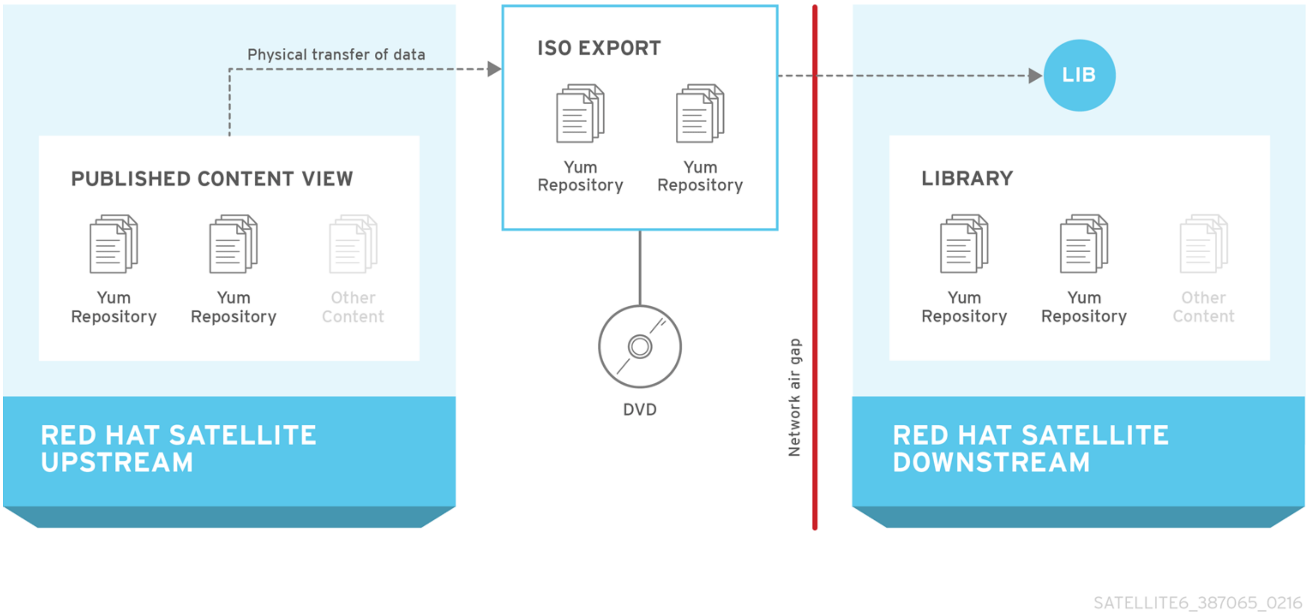
Only RPM, kickstart, and ISO files are exported. Content View definitions and metadata, such as package filters, is not transferred. Satellite 6.4 does not support the export of Puppet, Docker, or OSTree content. Imports occur as a normal repository synchronization, and consequently always arrive in the Library environment.

The disconnected use case is used extensively by customers who have air-gapped networks where a Satellite and its clients are on a network that is never connected to the Internet. The only way that these disconnected Satellites are populated with content is by exports from a connected Satellite.

IMPORTANT

Bidirectional synchronization is NOT used in a disconnected environment. Content never passes from the disconnected server to the connected server.

Figure D.1. Information Flow in ISS in an Air-gapped Network



D.1. SATELLITE SERVERS, CAPSULE SERVERS, AND ISS

ISS serves a distinct purpose in a Red Hat Satellite deployment. If you intend to include ISS as part of your deployment, it equates to keeping a separate Satellite Server that you need to maintain, back up, and so on. ISS does not provide a failover mechanism for clients, and neither is it designed as a backup and recovery system. It is a means of providing information sharing between Satellite Servers. The primary use case for implementing ISS is when you have a Satellite Server that is not connected to the Internet or to external content, and you need to synchronize content with a Satellite that **is** connected to the Internet. This can apply when you need complete isolation of management infrastructure for security or other purposes.

If you do not want to maintain another management web UI and platform, and you want to perform management and provisioning to local clients, consider setting up a Capsule Server.

D.2. PREREQUISITES

- In Red Hat Satellite 6, ISS is only available in Satellite 6.2 and later, which requires Red Hat Enterprise Linux 6.7, 7.2, or later.
- The export directory needs to be large enough to accommodate at least one Red Hat Enterprise Linux export. By default, the export directory is ***/var/lib/pulp/katello-export/***.
- The **`/var/cache/pulp/`** and **`/var/lib/pulp/`** directories must have free storage space equivalent to the size of the repository being exported for temporary files created during the export process. This is in addition to the space required by the default export directory.
- The downstream Satellite Server must have the required manifests and entitlements for any content that you intend to enable. You cannot enable repositories on a downstream Satellite for which no entitlements exist.
- The repository download policy needs to be set to **immediate**. This policy specifies whether or not Satellite first downloads metadata and other repository information, and only downloads actual repositories when requested. ISS does not function correctly if this policy is not set to **immediate**.

Section D.5, “Configuring ISS” describes how to configure required options.

D.3. SUPPORTED SYNCHRONIZATION OPTIONS

Satellite 6.4 supports the following synchronization options:

- Exporting repositories to a directory or ISO file.
- Exporting all repositories in an Environment or CV version to a directory or ISO file. You can also recreate any custom products during the import process, but Red Hat products are not recreated because they must be created using a manifest.
- Date-based incremental exports of RPM files and errata.

These synchronization options include a range of history details about the export and import, depending on the type of content. For example:

- The repository synchronization history includes upstream source information, as well as the time the export occurred.
- The CV synchronization history includes the export time and version, as well as the import time, version, and upstream source.

D.4. USING CHUNKED ISO FILES

Satellite 6.4 supports exports to chunked ISO files. A chunked ISO is similar to a split ISO but with one significant difference. Satellite tracks the size of the ISO file, and if the total size of the files being added to the ISO exceeds that value, Satellite stops writing to the ISO and creates a new one in the series. The advantage of this is that you can specify the ISO file size (for example, 4.7 GB), and still export larger repositories. The result is multiple 4.7 GB ISO files that you can burn to DVDs.

The difference between this and splitting large files is that the **`split`** utility is not aware of the ISO file format, which means that it does not create a new burnable ISO file for the next file in the series. This method requires that you copy all of the files to one place, concatenate those files, and then mount that single large ISO through loopback.

You can add the **`--iso-mb-size`** parameter to specify the size of ISO export files. The default value is 4380 MB, the size of a single-sided, single-layer DVD.

D.5. CONFIGURING ISS

This section describes how to configure the required settings for ISS. It is important that these settings be configured correctly or your synchronization may fail.

D.5.1. Configuring an Export Destination

Inter-Satellite Synchronization by default uses the ***/var/lib/pulp/katello-export/*** directory, as indicated by the '**pulp_export_destination**' setting. To change this directory you must create the new directory and configure the Pulp export destination setting. Only Satellite Administrators can specify this directory, and SELinux and other permissions are also in place to prevent Satellite from writing to arbitrary file systems.

It can be helpful to create symbolic links to commonly used directories after they are exported. Exported repositories and CVs have the organization name and environment name prepended to the repository directory structure, which might create paths of overlong length.

IMPORTANT

The directory used in this example is for demonstration purposes only. Confirm that the export directory has adequate space for the required export RPM and ISO files. Section 1.5, "Content Management Storage" provides information on estimating storage requirements. A temporary file is created during the export process in the **/var/cache/pulp/** and directory, then transferred to the **/var/lib/pulp/** directory. This means that storage space equal to twice the size of the repository being exported is required during the export process. The temporary file is deleted when the export is completed.

Create the Export Directory

1. Create the export directory:

```
# mkdir /var/www/html/pub/export
```

2. Ensure the **foreman** user has read and write permissions on the export directory:

```
# chown foreman:foreman /var/www/html/pub/export
```

3. Configure the SELinux context:

```
# semanage fcontext -a -t httpd_sys_rw_content_t \
"/var/www/html/pub/export(/.*)?"
# restorecon -RvF /var/www/html/pub/export
# ls -Zd /var/www/html/pub/export
drwxr-xr-x. foreman foreman
system_u:object_r:httpd_sys_rw_content_t:s0 /var/www/html/pub/export
```

Configure the Export Destination

For CLI Users

To change the export destination, enter the following **hammer** command:

```
# hammer settings set \
--name pulp_export_destination \
--value your-export-directory
```

For example, to specify **/var/www/html/pub/*export*/** as the export destination, enter:

```
# hammer settings set \
--name pulp_export_destination \
--value /var/www/html/pub/export
```

For Web UI Users

1. In the web UI, navigate to **Administer** > **Settings**, and click the **Content** tab.

2. Locate the '**pulp_export_destination**' variable in the **Name** column, and click the **Value** field.

3. Enter the export destination, for example **/var/www/html/pub/*export***, in the **Value** field and click **Save**.

D.5.2. Configuring the Download Policy

ISS requires that the '**Download Policy**' be set to **immediate**. You can set this globally so that it applies to new repositories created in all Organizations, or you can set it individually for every repository. Changing the default value does not change existing settings.

For CLI Users

To change the global default Download Policy, enter the following command:

```
# hammer settings set \
--name default_download_policy \
--value immediate
```

If required to change the policy for a specific repository, you can list the repositories for an organization as follows:

```
# hammer repository list \
--organization-label organization-label
```

To change the download policy for an existing repository, enter the following command:

```
# hammer repository update \
--organization-label organization-label  \
--product "Red Hat Enterprise Linux Server" \
--name "Red Hat Enterprise Linux 7 Server Kickstart x86_64 7.5"  \
--download-policy immediate
```

For Web UI Users

To change the global default Download Policy using the web UI:

1. Navigate to **Administer** > **Settings**.
2. On the **Content** tab, locate **`default_download_policy`**.
3. In the Value field, click the edit icon.
4. Set the value to **immediate**, and then click **Save**.

To change the Download Policy for an existing repository using the web UI:

1. In the web UI, navigate to **Content** > **Products**, and click the required product name.
2. On the **Repositories** tab, click the required repository name, locate the **Download Policy**' field, and click the edit icon.
3. From the list, select **Immediate**, and then click **Save**.

D.6. EXPORTING CONTENT

This section describes how to export different types of content from your upstream server, ready to be imported into to one or more downstream servers. For details on importing content, see Section D.7, “Importing Content”.

Synchronization between upstream and downstream servers is currently supported in disconnected deployments, for example in an air-gapped environment, where complete isolation is required.

D.6.1. Exporting Repositories

1. Enter the **`hammer repository list`** command to list the repositories which can be exported and identify their IDs, which are used in the export.

2. Enter the **`hammer repository export`** command to export content from your upstream server. This command exports content to the directory specified in '**pulp_export_destination**' setting. ISS exports to a directory by default. You can add the '**--export-to-iso 1**' parameter to export to an ISO file instead. For example:

   ```
   # hammer repository export --id 1 [--export-to-iso 1]
   ```

NOTE

If you use the '**--export-to-iso**' parameter, you need to specify either 1 (ISO) or 0 (directory). This parameter does not have a default value.

D.6.2. Exporting Content View Version to a Directory

You can export a specific version of a Content View to a directory. That means that you can label a particular version of a CV to suit your requirements. This way you can curate and track your exports and facilitate updates.

Prerequisites

- Ensure all repositories within the CV have their download policy set to **Immediate**. You cannot export repositories with policies other than **Immediate**.

- Ensure Products are synchronized to the required date.

For Web UI Users

1. Navigate to **Content** > **Content Views**. Click **Create New View**. Enter following details to create a CV:

   a. **Name** — A plain text name for the CV. Enter ***Export_CV***.

   b. **Label** — An internal ID for the CV. Red Hat Satellite 6 automatically completes this field based on what you have entered for **Name**.

   c. **Description** — An optional plain text description of the CV.

   d. **Composite View** — Defines whether or not to use a Composite Content View. Leave this option unselected.

2. Click **Save** to submit your changes.

3. On the **Repository Selection** screen, select the repositories to be added to the new CV from the **Repository Selection** table. Click **Add Repositories** to add selected packages to the CV.

4. Navigate to **Yum Content** > **Filters** and click **New Filter**. Enter following details to create a filter for including non-errata packages:

a. **Name** — A plain text name for the filter. Enter **Non-errata Products**.

b. **Content type** — A list of the types of content to be included into the filter. Select **Package**.

c. **Inclusion type** — A list defining whether the content are included or excluded from the CV. Select **Include**.

d. **Description** — An optional plain text description of the filter. Enter **Include all non-errata Products**.

e. Click **Save**.

f. On the **Include RPM** screen, select the **Include all RPMs with no errata** check box.

5. Navigate to **Yum Content** > **Filters** and click **New Filter**. Enter following details to create a filter for including errata packages in accordance with the required date range:

a. **Name** — A plain text name for the filter. Enter **Erratas untill *YYYY-MM-DD***.

b. **Content type** — A list of the types of content to be included into the filter. Select **Erratum - Date and Type**.

c. **Inclusion type** — A list defining whether the content is included or excluded from the CV. Select **Include**.

d. **Description** — An optional plain text description of the filter. Enter **Include errata products untill *YYYY-MM-DD***.

e. Click **Save**.

f. On the **Erratum Date Range** screen, select all **Security**, **Enhancement** and **Bugfix** errata types.

g. Select the **Updated On** check box in **Data type**.

h. Fill in the **Start Date** and **End Date** menus to configure the date range of Products for the filter.

i. Click **Save**.

6. Click **Publish New Version**, it is recommended to fill in the date ranges of repositories into the **Description** field. Ensure the **Force Yum Metadata Regeneration** check box is not selected.

7. Click **Save** to publish the CV version ready for exporting.

For CLI Users

1. Enter the **`hammer content-view create`** command to create a new CV:

```
# hammer content-view create \
--name "Export_CV" \
--organization "Default Organization"
```

2. Enter the **`hammer content-view add-repository`** command to add repositories to the CV:

```
# hammer content-view add-repository \
--name "Export_CV" \
```

```
--product "Red Hat Satellite" \
--repository "Red Hat Satellite Tools 6 for RHEL 7 Server RPMs
x86_64" \
--organization "Default Organization"
# hammer content-view add-repository \
--name "Export_CV" \
--product "Red Hat Satellite Capsule" \
--repository "Red Hat Satellite Capsule Tools 6.2 for RHEL 7 Server
RPMs x86_64" \
--organization "Default Organization"
```

3. Create filters for the new CV:

    a. Enter the **hammer content-view filter create** command to create a filter for including non-errata packages:

    ```
    # hammer content-view filter create \
    --content-view "Export_CV" \
    --inclusion true \
    --name "Non-errata_Products" \
    --type rpm \
    --original-packages true \
    --organization "Default Organization"
    ```

    b. Enter the **hammer content-view filter create** command to create a filter for including errata packages:

    ```
    # hammer content-view filter create \
    --content-view "Export_CV" \
    --inclusion true \
    --name "Erratas until YYYY-MM-DD" \
    --type erratum \
    --organization "Default Organization"
    ```

    c. Enter the **hammer content-view filter rule create** command to create a rule defining the date range:

    ```
    # hammer content-view filter rule create \
    --content-view "Export_CV" \
    --content-view-filter "Erratas until YYYY-MM-DD" \
    --end-date YYYY-MM-DD \
    --types security,enhancement,bugfix \
    --organization "Default Organization"
    ```

4. Enter the **hammer content-view publish** command to publish the CV version ready for exporting. It is recommended to fill in the date ranges of repositories under the **--description** option.

```
# hammer content-view publish \
--name "Export_CV" \
--description "Repositories until YYYY-MM-DD" \
--force-yum-metadata-regeneration true \
--async \
--organization "Default Organization"
```

To Determine Which Content View Version to Export:

1. Enter the **hammer content-view version list** command to determine which version of a Content View to export. For example:

```
$ hammer content-view version list \
--organization "Default Organization"
---|-------------------------------|---------|-------------------
----
ID | NAME                          | VERSION | LIFECYCLE
ENVIRONMENTS
---|-------------------------------|---------|-------------------
----
3  | Export_CV 2.0                 | 2.0     | Library
2  | Export_CV 1.0                 | 1.0     | Library
1  | Default Organization View 1.0 | 1.0     | Library
---|-------------------------------|---------|-------------------
----
```

To Export a Content View Version:

1. Enter the **hammer content-view version export** command to export a version of a Content View:

```
# hammer content-view version export --id 3
```

D.6.3. Incremental Export

To avoid exporting large repositories when updating, you can use incremental updates to only export the recent updates from a Satellite Server. Incremental updates export the recent changes made in the local repository by using synchronization events set to a particular date and time.

To make an incremental-update repository, enter the **hammer repository export** command with the **--since** option. For example:

```
# hammer repository export \
--id 1 [--export-to-iso 1] \
--since ISO_Date
```

Where *ISO_Date* is in ISO 8601 format. For example, **2010-01-01T12:00:00Z**.

The time stamp used for the calculations is the time that the RPMs were synchronized on Satellite Server. For example, if Red Hat adds RPMs to a repository on a Monday and then again on a Wednesday, you cannot synchronize your local repository on Thursday and then use a date of Tuesday to get only the Wednesday update.

In addition to exporting changes to a repository, this feature is useful with the Default Organization View Content View, but is not as useful for published CVs.

D.7. IMPORTING CONTENT

Red Hat Satellite 6.4 currently supports importing content that has been exported from an upstream Satellite Server in a disconnected environment. This method is used for disconnected Satellite Servers that have no Internet access, and requires the physical transfer of content between the servers, for

example using a DVD.

D.7.1. Importing a Repository

Prerequisites

- Export the repository from the upstream Satellite Server. For more details, see Section D.6.1, “Exporting Repositories”.

To Import a Repository:

1. Make the data available for a repository over HTTP, not HTTPS. For example, copy the exported directory to the **/var/www/html/pub/export/** directory on the downstream server, which is available over HTTP by default.
2. In the web UI, navigate to **Content** > **Subscriptions**.
3. Select **Manage Manifests**.
4. On the **Import/Remove Manifest** tab, set the **Red Hat CDN URL** address field to match the location of a **content** directory and a **listing** file within the exported repository.
   For example, if the exported repository is located in **/var/www/html/pub/export/Default_Organization-Red_Hat_Enterprise_Linux_7_Server_RPMs_x68_64**, set the URL to be **http://satellite.example.com/pub/export/Default_Organization-Red_Hat_Enterprise_Linux_7_Server_RPMs_x68_64/Default_Organization/Library/**.
5. Click **Save**.
6. Navigate to **Content** > **Red Hat Repositories** and enable the exported repository.

D.7.2. Importing a Content View as a Red Hat Repository

Prerequisites

- You have a Content View with Red Hat repositories on the upstream Satellite Server.
- Export the Content View from the upstream Satellite Server. For more details, see Section D.6.2, “Exporting Content View Version to a Directory”.

NOTE

A custom repository can also be imported to a custom product, for example a disconnected Satellite. For more information about Red Hat Content Delivery Network (CDN), see Content Delivery Network Structure in *Planning for Red Hat Satellite 6*.

To Import a Content View:

1. Make the data available for a repository over HTTP, not HTTPS. For example, copy it to the **/var/www/html/pub/export/** directory on the downstream server.
2. In the web UI, navigate to **Content** > **Subscriptions**.

3. Select **Manage Manifests**.

4. On the **Import/Remove Manifest** tab, set the **Red Hat CDN URL** address field to match the location of a **`content`** directory and a **`listing`** file within the exported Content View.
   For example, if the exported CV is located in **`/var/www/html/pub/export/Default_Organization-Export_CV-v1.0`**, set the URL to be **`http://satellite.example.com/pub/export/Default_Organization-Export_CV-v1.0/Default_Organization/content_views/Export_CV/1.0/`**

5. Click **Save**.

6. Navigate to **Content** > **Red Hat Repositories** and enable the exported repository.

7. On a downstream server, enter the **`hammer organization update`** command to add new repositories to an organization. Set the address to the directory corresponding to the version you want within the exported Content View as shown:

```
$ hammer organization update \
--name "Default Organization" \
--redhat-repository-url \
http://satellite.example.com/pub/export/Default_Organization-Export_
CV-v1.0/Default_Organization/content_views/Export_CV/1.0/

Organization updated
```

D.7.3. Incremental Import

When synchronizing from an incremental update, you must use the incremental synchronization settings. In the Satellite web UI, ensure the "Mirror on Sync" setting is disabled for the repositories before synchronizing. If you are using the Hammer CLI, you must add the "--incremental" option. If you do not use the incremental synchronization settings, the repository content will be deleted and only the content from the incremental update will remain. Recovering from an accidental overwrite requires a full export and then synchronizing again.

APPENDIX E. CREATING A REMOTE FILE TYPE REPOSITORY

You can create a custom file type repository, from a directory of files, external to Satellite Server using the **`pulp-manifest`** command. You can then synchronize the files into Satellite Server over HTTP or HTTPS. When you add files to a file type repository, you can work with the files as with any other repository.

This procedure describes configuring a repository in a directory on a remote server. To create a file type repository in a directory on the base system where Satellite Server is installed, see Section 6.12, “Creating a Custom File Type Repository in a Local Directory”.

Prerequisites

To configure a remote file type repository, ensure the following conditions have been met:

- You have a Red Hat Enterprise Linux 7 server registered to your Satellite or the Red Hat CDN.
- Your server has an entitlement to the Red Hat Enterprise Linux Server and Satellite Tools repositories.
- You have installed an HTTP server. For more information about configuring a web server, see The Apache HTTP Server in the Red Hat Enterprise Linux 7 *System Administrator’s Guide*.

To Create a File Type Repository in a Remote Directory:

1. Ensure the Server and Satellite Tools repositories are enabled:

```
# subscription-manager repos --enable=rhel-7-server-rpms \
--enable=rhel-7-server-satellite-tools-6.4-rpms
```

2. Install the Pulp Manifest package:

```
# yum install python-pulp-manifest
```

3. Create a directory that you want to use as the file type repository in the HTTP server’s public folder:

```
# mkdir /var/www/html/pub/my_file_repo
```

4. Add files to the directory or create a test file:

```
# touch /var/www/html/pub/my_file_repo/test.txt
```

5. Enter the Pulp Manifest command to create the manifest:

```
# pulp-manifest /var/www/html/pub/my_file_repo
```

6. Verify the manifest was created:

```
# ls /var/www/html/pub/my_file_repo
PULP_MANIFEST  test.txt
```

To Import Files From a Remote File Type Repo

1. Create a File Type Repository
   In the Satellite web UI, navigate to **Content** > **Products**. Select the name of a product, in this example *My File Product*. On the **Repositories** tab, select **New Repository** and enter the following details:

   - **Name** - A plain text name for the repository. Enter `My Files`.
   - **Label** - An internal ID for the repository. Red Hat Satellite 6 automatically completes this field based on what you enter for **Name**.
   - **Type** - The content type of the repository. Select `file`.
   - **Upstream URL** - The URL of the remote repository to use as the source.
   - **Verify SSL** - Leave this option selected if you want to verify that the upstream repository's SSL certificates are signed by a trusted CA. Alternatively, clear this check box.
   - **Upstream Username** - If required for authentication, enter the user name for the upstream repository. Clear this field if the repository does not require authentication.
   - **Upstream Password** - Enter the password for the upstream repository. Clear this field if the repository does not require authentication.

2. Select **Save** to save this repository entry.

3. To update the file type repository, navigate to **Content** > **Products**. Select the name of a product, in this example *My File Product*. Select the name of the repository you want to update, in this example *My Files*.

4. From the **Select Action** menu, select **Sync Now**.

5. Visit the URL where the repository is published to see the files.

APPENDIX F. SYNCHRONIZING TEMPLATES WITH GIT

Red Hat Satellite 6 enables synchronization of Job Templates, Provisioning Templates, and Partition Table Templates between Satellite Server and a Git repository or a local directory.

NOTE

Synchronizing templates between Satellite Server and a Git repository or a local directory 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. However, these features provide early access to upcoming product innovations, enabling customers to test functionality and provide feedback during the development process.

This section details the workflow for:

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

F.1. ENABLING THE TEMPLATESYNC PLUG-IN

1. To enable the plug-in on your Satellite Server:

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

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

F.2. CONFIGURING THE TEMPLATESYNC PLUG-IN

Navigate to **Administer** > **Settings** > **TemplateSync** to configure the plug-in. The following table explains the attributes behavior. Note that some attributes are only used on importing or exporting tasks.

Table F.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.

Parameter	API parameter name	Meaning on importing	Meaning on exporting
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
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.

Parameter	API parameter name	Meaning on importing	Meaning on exporting
Verbosity	**verbose** Accepted values: **true**, **false**	Enables writing verbose messages to the logs for this action.	N/A

F.3. IMPORTING AND EXPORTING TEMPLATES

Importing and exporting tasks are available through a series of API calls. API calls use the role-based access control system, which enables the tasks to be executed as any user. The TemplateSync plug-in allows synchronizing with a Git repository 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
%>
```

You can also import and export templates using Hammer. For more information, see Provisioning Templates in the *Hammer Cli Guide*.

F.3.1. Synchronizing Templates with a Git repository

1. Configure a Git server 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 any passphrase.

```
# sudo -u foreman ssh-keygen
```

5. Configure your Git 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 Git 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.

F.3.2. Synchronizing templates with a local directory

Synchronizing templates with a local directory is useful if you have configured any revision control system 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\"}"
> ```

F.4. 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.

F.5. UNINSTALLING THE PLUG-IN

To avoid errors after uninstallation:

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
```