Red Hat Satellite 6.16

Managing content

Import content from Red Hat and custom sources, manage applications lifecycle across lifecycle environments, filter content by using Content Views, synchronize content between Satellite Servers, and more

Last Updated: 2024-11-05

Red Hat Satellite 6.16 Managing content

Import content from Red Hat and custom sources, manage applications lifecycle across lifecycle environments, filter content by using Content Views, synchronize content between Satellite Servers, and more

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

Legal Notice

Abstract

Use this guide to understand and manage content in Satellite 6. Examples of such content include RPM files, and ISO 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

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our documentation. Let us know how we can improve it.

Use the **Create Issue** form in Red Hat Jira to provide your feedback. The Jira issue is created in the Red Hat Satellite Jira project, where you can track its progress.

Prerequisites

- Ensure you have registered a Red Hat account.

Procedure

1. Click the following link: **Create Issue**. If Jira displays a login error, log in and proceed after you are redirected to the form.
2. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
3. Click **Create**.

CHAPTER 1. INTRODUCTION TO CONTENT MANAGEMENT

In the context of Satellite, *content* is defined as the software installed on systems. This includes, but is not limited to, the base operating system, middleware services, and end-user applications. With Red Hat Satellite, you can manage the various types of content for Red Hat Enterprise Linux systems at every stage of the software lifecycle.

Red Hat Satellite manages the following content:

Subscription management

This provides organizations with a method to manage their Red Hat subscription information.

Content management

This provides organizations with a method to store Red Hat content and organize it in various ways.

1.1. CONTENT TYPES IN RED HAT SATELLITE

With Red Hat Satellite, you can import and manage many content types.

For example, Satellite supports the following content types:

RPM packages

Import RPM packages from repositories related to your Red Hat subscriptions. Satellite Server downloads the RPM packages from the Red Hat Content Delivery Network and stores them locally. You can use these repositories and their RPM packages in content views.

Kickstart trees

Import the Kickstart trees to provision a host. New systems access these Kickstart trees over a network to use as base content for their installation. Red Hat Satellite contains predefined Kickstart templates. You can also create your own Kickstart templates.

ISO and KVM images

Download and manage media for installation and provisioning. For example, Satellite downloads, stores, and manages ISO images and guest images for specific Red Hat Enterprise Linux and non-Red Hat operating systems.

Custom file type

Manage custom content for any type of file you require, such as SSL certificates, ISO images, and OVAL files.

CHAPTER 2. MANAGING RED HAT SUBSCRIPTIONS

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

Use this chapter to import a Red Hat subscription manifest and manage the manifest within the Satellite web UI.

Subscription allocations and organizations

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

Future-dated subscriptions

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

Prerequisites

- Ensure you have a Red Hat subscription manifest.
  - If your Satellite is connected, use the Red Hat Hybrid Cloud Console to create the manifest. For more information, see Creating and managing manifests for a connected Satellite Server in *Subscription Central*.
  - If your Satellite is disconnected, use the Red Hat Customer Portal to create the manifest. For more information, see Using manifests for a disconnected Satellite Server in *Subscription Central*.

Additional resources

- Configuring Satellite Server to Consume Content from a Custom CDN in *Installing Satellite Server in a disconnected network environment*

2.1. IMPORTING A RED HAT SUBSCRIPTION MANIFEST INTO SATELLITE SERVER

Use the following procedure to import a Red Hat subscription manifest into Satellite Server.

NOTE

Simple Content Access (SCA) is set on the organization, not the manifest. Importing a manifest does not change your organization's Simple Content Access status.

Prerequisites

- Ensure you have a Red Hat subscription manifest.

- If your Satellite is connected, use the Red Hat Hybrid Cloud Console to create and export the manifest. For more information, see Creating and managing manifests for a connected Satellite Server in *Subscription Central*.
- If your Satellite is disconnected, use the Red Hat Customer Portal to create and export the manifest. For more information, see Using manifests for a disconnected Satellite Server in *Subscription Central*.

Procedure

1. In the Satellite web UI, ensure the context is set to the organization you want to use.
2. In the Satellite web UI, navigate to **Content** > **Subscriptions** and click **Manage Manifest**.
3. In the **Manage Manifest** window, click **Choose File**.
4. Navigate to the location that contains the Red Hat subscription manifest file, then click **Open**.

CLI procedure

1. Copy the Red Hat subscription manifest file from your local machine to Satellite Server:

```
$ scp ~/manifest_file.zip root@satellite.example.com:~/.
```

2. Log in to Satellite Server as the **root** user and import the Red Hat subscription manifest file:

```
# hammer subscription upload \
--file ~/manifest_file.zip \
--organization "My_Organization"
```

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

2.2. LOCATING A RED HAT SUBSCRIPTION

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

Prerequisites

- You must have a Red Hat subscription manifest file imported to Satellite Server. For more information, see Section 2.1, “Importing a Red Hat 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. In the Satellite web UI, navigate to **Content** > **Subscriptions**.
3. In the Subscriptions window, click the **Search** field to view the list of search criteria for building your search query.
4. Select 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 = character. If you select > and press the space bar, another list of automatic options appears. You can also enter your own criteria.

2.3. ADDING RED HAT SUBSCRIPTIONS TO SUBSCRIPTION MANIFESTS

Use the following procedure to add Red Hat subscriptions to a subscription manifest in the Satellite web UI.

Prerequisites

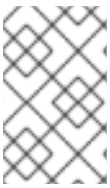
- You must have a Red Hat subscription manifest file imported to Satellite Server. For more information, see Section 2.1, “Importing a Red Hat subscription manifest into Satellite Server” .

Procedure

1. In the Satellite web UI, ensure the context is set to the organization you want to use.
2. In the Satellite web UI, navigate to **Content** > **Subscriptions**.
3. In the Subscriptions window, click **Add Subscriptions**.
4. On the row of each subscription you want to add, enter the quantity in the **Quantity to Allocate** column.
5. Click **Submit**

2.4. REMOVING RED HAT SUBSCRIPTIONS FROM SUBSCRIPTION MANIFESTS

Use the following procedure to remove Red Hat subscriptions from a subscription manifest 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.

Prerequisites

- You must have a Red Hat subscription manifest file imported to Satellite Server. For more information, see Section 2.1, “Importing a Red Hat 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. In the Satellite web UI, navigate to **Content** > **Subscriptions**.
3. On the row of each subscription you want to remove, select the corresponding checkbox.

4. Click **Delete**, and then confirm deletion.

2.5. UPDATING AND REFRESHING RED HAT 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 refresh the manifest directly in the Satellite web UI. Alternatively, you can import an updated manifest that contains the changes.

Procedure

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

2.6. CONTENT DELIVERY NETWORK STRUCTURE

Red Hat Content Delivery Network (CDN), located at **cdn.redhat.com**, is a geographically distributed series of static webservers which include content and errata designed to be used by systems. This content can be accessed directly through a system registered by using Subscription Manager or through the Satellite web UI. The accessible subset of the CDN is configured through content available to a system by using Red Hat Subscription Management or by using Satellite Server.

Red Hat Content Delivery network is protected by X.509 certificate authentication to ensure that only valid users can access it.

Directory structure of the CDN

```
$ tree -d -L 11
└── content (1)
    ├── beta (2)
    │   └── rhel (3)
    │       └── server (4)
    │           └── 7 (5)
    │               └── x86_64 (6)
    │                   └── sat-tools (7)
    └── dist
        └── rhel
            └── server
                └── 7
                    ├── 7.2
                    │   └── x86_64
```

```
│       └── kickstart
└── 7Server
    └── x86_64
        └── os
```

1. The **content** directory.
2. Directory responsible for the lifecycle of the content. Common directories include **beta** (for Beta code), **dist** (for Production) and **eus** (For Extended Update Support) directories.
3. Directory responsible for the product name. Usually **rhel** for Red Hat Enterprise Linux.
4. Directory responsible for the type of the product. For Red Hat Enterprise Linux this might include **server**, **workstation**, and **computenode** directories.
5. Directory responsible for the release version, such as **7**, **7.2** or **7Server**.
6. Directory responsible for the base architecture, such as **i386** or **x86_64**.
7. Directory responsible for the repository name, such as **sat-tools**, **kickstart**, **rhscl**. Some components have additional subdirectories which might vary.

This directory structure is also used in the Red Hat Subscription Manifest.

CHAPTER 3. MANAGING ALTERNATE CONTENT SOURCES

Alternate content sources define alternate paths to download content during synchronization. The content itself is downloaded from the alternate content source, while the metadata is downloaded from the Satellite Server or the upstream URL, depending on the configuration. You can use alternate content source to speed up synchronization if the content is located on the local filesystem or on a nearby network. You can set up alternate content sources for Satellite Server and Capsule. You must refresh the alternate content source after creation or after making any changes. A weekly cron job refreshes all alternate content sources. You can also refresh the alternate content sources manually by using the Satellite web UI or the Hammer CLI. Alternate content sources associated with your Satellite Server, or Capsule Servers attached to multiple organizations, affect all organizations.

There are three types of alternate content sources:

Custom

Custom alternate content sources download the content from any upstream repository on the network or filesystem.

Simplified

Simplified alternate content sources copy the upstream repository information from your Satellite Server for the selected products. Simplified alternate content sources are ideal for situations where the connection from your Capsule to the upstream repo is faster than from your Satellite Server. Selecting the Red Hat products when creating a simplified alternate content source will download the content to the Capsules from the Red Hat CDN.

RHUI

RHUI alternate content sources download content from a Red Hat Update Infrastructure server. Satellite web UI provides examples to help you find the network paths and to import authentication credentials. The RHUI alternate content source must be RHUI version 4 or greater and use the default installation configuration. For example, AWS RHUI is unsupported because it uses an installation scenario with unique authentication requirements.

Permission requirements for alternate content sources

Non-administrator users must have the below permissions to manage alternate content sources:

1. **view_smart_proxies**
2. **view_content_credentials**
3. **view_organizations**
4. **view_products**

In addition to the above permissions, assign permissions specific to alternate content sources, depending on the actions the users can perform:

1. **view_alternate_content_sources**
2. **create_alternate_content_sources**
3. **edit_alternate_content_sources**
4. **destroy_alternate_content_sources**

3.1. CONFIGURING CUSTOM ALTERNATE CONTENT SOURCES

Prerequisites

- If the repository requires SSL authentication, import the SSL certificate and key to the Satellite. For more information, see Importing Custom SSL Certificates in *Managing content*.
- Note that the alternate content source paths consist of a base URL appended with the subpaths that you provide. For example, if your base URL is **https://server.example.com** and your subpaths are **rhel7/** and **rhel8/**, then both **https://server.example.com/rhel7/** and **https://server.example.com/rhel8/** will be searched.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Alternate Content Sources**.
2. Click **Add Source** and set the **Source type** as **Custom**.
3. Select the **Content type**.
4. In the **Name** field, enter a name for the alternate content source.
5. Optional: In the the **Description** field, provide a description for the ACS.
6. Select Capsules to which the alternate content source is to be synced.
7. Enter the Base URL of the alternate content source.
8. Enter a comma-separated list of Subpaths.
9. Provide the **Manual Authentication** or **Content Authentication** credentials, if these are needed.
10. If SSL verification is required, enable **Verify SSL** and select the SSL CA certificate.
11. Review details and click **Add**.
12. Navigate to **Content** > **Alternate Content Sources** > click the vertical ellipsis next to the newly created alternate content source > Select **Refresh**.

CLI procedure

1. On Satellite Server, enter the following command:

```
# hammer alternate-content-source create \
--alternate-content-source-type custom \
--base-url "https://local-repo.example.com:port" \
--name "My_ACS_Name" \
--smart-proxy-ids My_Capsule_ID
```

2. Check if the newly created alternate content source is listed:

```
# hammer alternate-content-source list
```

3. Refresh the alternate content source:

```
# hammer alternate-content-source refresh --id My_Alternate_Content_Source_ID
```

4. Add the Capsules to which the alternate content source is to be synced:

```
# hammer alternate-content-source update \
--id My_Alternate_Content_Source_ID \
--smart-proxy-ids My_Capsule_ID
```

5. Refresh the alternate content sources:

```
# hammer alternate-content-source refresh --id My_Alternate_Content_Source_ID
```

3.2. CONFIGURING SIMPLIFIED ALTERNATE CONTENT SOURCES

Procedure

1. In the Satellite web UI, navigate to **Content** > **Alternate Content Sources**.
2. Click **Add Source** and set the **Source type** as **Simplified**.
3. Select the **Content type**.
4. In the **Name** field, enter a name for the alternate content source.
5. Optional: In the **Description** field, provide a description for the ACS.
6. Select Capsules to which the alternate content source is to be synced.
7. Optional: Select **Use HTTP proxies** if you want the ACS to use the Capsule Server's HTTP proxy.
8. Select the products that should use the alternate content source.
9. Review details and click **Add**.
10. Navigate to **Content** > **Alternate Content Sources**, click the vertical ellipsis next to the newly created alternate content source, and select **Refresh**.

CLI procedure

1. Create a simplified ACS:

```
# hammer alternate-content-source create \
--alternate-content-source-type simplified \
--name My_ACS_Name \
--product-ids My_Product_ID \
--smart-proxy-ids My_Capsule_ID
```

2. Check if the newly created ACS is listed:

```
# hammer alternate-content-source list
```

3. Refresh the ACS:

```
# hammer alternate-content-source refresh --id My_ACS_ID
```

3.2.1. Synchronizing Capsule directly from Red Hat CDN

You can use simplified alternate content sources to configure your Capsule to sync content directly from Red Hat CDN.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Alternate Content Sources**.
2. Click **Add Source** and set the **Source type** as **Simplified**.
3. Set the **Content type** to **Yum**.
4. In the **Name** field, enter a name for the alternate content source.
5. Optional: In the **Description** field, provide a description for the alternate content source.
6. Select Capsules that you want to sync directly from Red Hat CDN.
7. Optional: Select **Use HTTP proxies** if you want the ACS to use the Capsule Server's HTTP proxy.
8. Select the Red Hat products that should be synced to the Capsule from Red Hat CDN.
9. Review details and click **Add**.
10. Navigate to **Content** > **Alternate Content Sources**, click the vertical ellipsis next to the newly created alternate content source, and select **Refresh**.

The Capsule will now download content from Red Hat CDN and not the Satellite.

3.3. CONFIGURING RHUI ALTERNATE CONTENT SOURCES

Prerequisites

- Generate the client entitlement certificates for the required repos on the RHUA node as described in Creating a client entitlement certificate with the Red Hat Update Infrastructure Management Tool in *Configuring and Managing Red Hat Update Infrastructure* .
- Import the client entitlement certificates into Satellite. For more information, see Importing Custom SSL Certificates in *Managing content*.
- Obtain a list of the subpaths for the required repositories. Execute the following command on your RHUA server:

```
# rhui-manager repo info --repo_id My_Repo_ID
```

- Note that the alternate content source paths consist of a base URL appended with the subpaths that you provide. For example, if your base URL is **https://server.example.com** and your subpaths are **rhel7/** and **rhel8/**, then both **https://server.example.com/rhel7/** and **https://server.example.com/rhel8/** will be searched.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Alternate Content Sources**.

2. Click **Add Source** and set the **Source type** as **RHUI**.
3. Generate RHUI certificates using the command provided in the Satellite web UI. Ensure that you pass the repo labels of the desired repositories.
4. In the **Name** field, enter a name for the alternate content source.
5. Optional: In the **Description** field, provide a description for the ACS.
6. Select Capsules to which the alternate content source is to be synced.
7. Optional: Select **Use HTTP proxies** if you want the ACS to use the Capsule's HTTP proxy.
8. Enter the Base URL of the Red Hat Update Infrastructure CDS node.
9. Enter a comma-separated list of Subpaths.
10. Provide the **Content Credentials**, if these are needed.
11. If SSL verification is required, enable **Verify SSL** and select the SSL CA certificate.
12. Review details and click **Add**.
13. Navigate to **Content** > **Alternate Content Sources**, click the vertical ellipsis next to the newly created alternate content source, and select **Refresh**.

CLI procedure

1. On Satellite Server, enter the following command:

```
# hammer alternate-content-source create \
--alternate-content-source-type rhui \
--base-url "https://rhui-cds-node/pulp/content" \
--name "My_ACS_Name" \
--smart-proxy-ids My_Capsule_ID \
--ssl-client-cert-id My_SSL_Client_Certificate_ID \
--ssl-client-key-id My_SSL_Client_Key_ID \
--subpaths path/to/repo/1/,path/to/repo/2/ \
--verify-ssl 1
```

2. Check if the newly created alternate content source is listed:

```
# hammer alternate-content-source list
```

3. Refresh the alternate content source:

```
# hammer alternate-content-source refresh --id My_Alternate_Content_Source_ID
```

4. Add the Capsules to which the alternate content source is to be synced:

```
# hammer alternate-content-source update \
--id My_Alternate_Content_Source_ID \
--smart-proxy-ids My_Capsule_ID
```

5. Refresh the alternate content sources:

```
# hammer alternate-content-source refresh --id My_Alternate_Content_Source_ID
```

CHAPTER 4. IMPORTING CONTENT

This chapter outlines how you can import different types of custom content to Satellite. For example, you can use the following chapters for information on specific types of custom content but the underlying procedures are the same:

- Chapter 12, *Managing ISO images*
- Chapter 14, *Managing custom file type content*

4.1. PRODUCTS AND REPOSITORIES IN SATELLITE

Both Red Hat content and custom content in Satellite have similarities:

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

Red Hat content is already organized into products. For example, Red Hat Enterprise Linux Server is a *product* in Satellite. The repositories for that product consist of different versions, architectures, and add-ons. For Red Hat repositories, products are created automatically after enabling the repository. For more information, see Section 4.6, “Enabling Red Hat repositories” .

Other content can be organized into custom products however you want. For example, you might create an EPEL (Extra Packages for Enterprise Linux) Product and add an "EPEL 7 x86_64" repository to it.

For more information about creating and packaging RPMs, see the *Red Hat Enterprise Linux 7 RPM Packaging Guide*.

4.2. BEST PRACTICES FOR PRODUCTS AND REPOSITORIES

- Use one content type per product and content view, for example, yum content only.
- Make file repositories available over HTTP. If you set **Protected** to true, you can only download content using a global debugging certificate.
- Automate the creation of multiple products and repositories by using a Hammer script or an Ansible Playbook.
- For Red Hat content, import your Red Hat manifest into Satellite. For more information, see Chapter 2, *Managing Red Hat subscriptions*.
- Avoid uploading content to repositories with an **Upstream URL**. Instead, create a repository to synchronize content and upload content to without setting an **Upstream URL**.
  If you upload content to a repository that already synchronizes another repository, the content might be overwritten, depending on the mirroring policy and content type.

4.3. IMPORTING CUSTOM SSL CERTIFICATES

Before you synchronize custom content from an external source, you might need to import SSL certificates into your custom product. This might include client certs and keys or CA certificates for the upstream repositories you want to synchronize.

If you require SSL certificates and keys to download packages, you can add them to Satellite.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Content Credentials**. In the Content Credentials window, click **Create Content Credential**.
2. In the **Name** field, enter a name for your SSL certificate.
3. From the **Type** list, select **SSL Certificate**.
4. In the **Content Credentials Content** field, paste your SSL certificate, or click **Browse** to upload your SSL certificate.
5. Click **Save**.

CLI procedure

1. Copy the SSL certificate to your Satellite Server:

```
$ scp My_SSL_Certificate root@satellite.example.com:~/.
```

Or download the SSL certificate to your Satellite Server from an online source:

```
$ wget -P ~ http://upstream-satellite.example.com/pub/katello-server-ca.crt
```

2. Upload the SSL Certificate to Satellite:

```
# hammer content-credential create \
--content-type cert \
--name "My_SSL_Certificate" \
--organization "My_Organization" \
--path ~/My_SSL_Certificate
```

4.4. CREATING A CUSTOM PRODUCT

Create a custom product so that you can add repositories to the custom product. To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**, click **Create Product**.
2. In the **Name** field, enter a name for the product. Satellite automatically completes the **Label** field based on what you have entered for **Name**.
3. Optional: From the **GPG Key** list, select the GPG key for the product.
4. Optional: From the **SSL CA Cert** list, select the SSL CA certificate for the product.
5. Optional: From the **SSL Client Cert** list, select the SSL client certificate for the product.

6. Optional: From the **SSL Client Key** list, select the SSL client key for the product.
7. Optional: From the **Sync Plan** list, select an existing sync plan or click **Create Sync Plan** and create a sync plan for your product requirements.
8. In the **Description** field, enter a description of the product.
9. Click **Save**.

CLI procedure

To create the product, enter the following command:

```
# hammer product create \
--name "My_Product" \
--sync-plan "Example Plan" \
--description "Content from My Repositories" \
--organization "My_Organization"
```

4.5. ADDING CUSTOM RPM REPOSITORIES

Use this procedure to add custom RPM repositories in Satellite. To use the CLI instead of the Satellite web UI, see the CLI procedure.

The **Products** window in the Satellite web UI also provides a **Repo Discovery** function that finds all repositories from a URL and you can select which ones to add to your custom product. For example, you can use the **Repo Discovery** to search **https://download.postgresql.org/pub/repos/yum/16/redhat/** and list all repositories for different Red Hat Enterprise Linux versions and architectures. This helps users save time importing multiple repositories from a single source.

Support for custom RPMs

Red Hat does not support the upstream RPMs directly from third-party sites. These RPMs are used to demonstrate the synchronization process. For any issues with these RPMs, contact the third-party developers.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products** and select the product that you want to use, and then click **New Repository**.
2. In the **Name** field, enter a name for the repository. Satellite automatically completes the **Label** field based on what you have entered for **Name**.
3. Optional: In the **Description** field, enter a description for the repository.
4. From the **Type** list, select **yum** as type of repository.
5. Optional: From the **Restrict to Architecture** list, select an architecture. If you want to make the repository available to all hosts regardless of the architecture, ensure to select **No restriction**.
6. Optional: From the **Restrict to OS Version** list, select the OS version. If you want to make the repository available to all hosts regardless of the OS version, ensure to select **No restriction**.

7. Optional: In the **Upstream URL** field, enter the URL of the external repository to use as a source. Satellite supports three protocols: **http://**, **https://**, and **file://**. If you are using a **file://** repository, you have to place it under **/var/lib/pulp/sync_imports/** directory.
   If you do not enter an upstream URL, you can manually upload packages.

8. Optional: Check the **Ignore SRPMs** checkbox to exclude source RPM packages from being synchronized to Satellite.

9. Optional: Check the **Ignore treeinfo** checkbox if you receive the error **Treeinfo file should have INI format**. All files related to Kickstart will be missing from the repository if **treeinfo** files are skipped.

10. Select the **Verify SSL** checkbox if you want to verify that the upstream repository's SSL certificates are signed by a trusted CA.

11. Optional: In the **Upstream Username** field, enter the user name for the upstream repository if required for authentication. Clear this field if the repository does not require authentication.

12. Optional: In the **Upstream Password** field, enter the corresponding password for the upstream repository. Clear this field if the repository does not require authentication.

13. Optional: In the **Upstream Authentication Token** field, provide the token of the upstream repository user for authentication. Leave this field empty if the repository does not require authentication.

14. From the **Download Policy** list, select the type of synchronization Satellite Server performs. For more information, see Section 4.9, "Download policies overview".

15. From the **Mirroring Policy** list, select the type of content synchronization Satellite Server performs. For more information, see Section 4.12, "Mirroring policies overview".

16. Optional: In the **Retain package versions** field, enter the number of versions you want to retain per package.

17. Optional: In the **HTTP Proxy Policy** field, select an HTTP proxy.

18. From the **Checksum** list, select the checksum type for the repository.

19. Optional: You can clear the **Unprotected** checkbox to require a subscription entitlement certificate for accessing this repository. By default, the repository is published through HTTP.

20. Optional: From the **GPG Key** list, select the GPG key for the product.

21. Optional: In the **SSL CA Cert** field, select the SSL CA Certificate for the repository.

22. Optional: In the **SSL Client cert** field, select the SSL Client Certificate for the repository.

23. Optional: In the **SSL Client Key** field, select the SSL Client Key for the repository.

24. Click **Save** to create the repository.

CLI procedure

1. Enter the following command to create the repository:

```
# hammer repository create \
--arch "My_Architecture" \
```

```
--content-type "yum" \
--gpg-key-id My_GPG_Key_ID \
--name "My_Repository" \
--organization "My_Organization" \
--os-version "My_OS_Version" \
--product "My_Product" \
--publish-via-http true \
--url My_Upstream_URL
```

Continue to synchronize the repository.

4.6. ENABLING RED HAT REPOSITORIES

If outside network access requires usage of an HTTP proxy, configure a default HTTP proxy for your server. For more information, see Adding a Default HTTP Proxy to Satellite .

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.

For Red Hat Enterprise Linux 8 hosts

To provision Red Hat Enterprise Linux 8 hosts, you require the **Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)** and **Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)** repositories.

For Red Hat Enterprise Linux 7 hosts

To provision Red Hat Enterprise Linux 7 hosts, you require the **Red Hat Enterprise Linux 7 Server (RPMs)** repository.

The difference between associating Red Hat Enterprise Linux operating system release version with either **7Server** repositories or **7.*X*** repositories is that **7Server** 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.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Red Hat Repositories**.
2. To find repositories, either enter the repository name, or toggle the **Recommended Repositories** button to the on position to view a list of repositories that you require.
3. In the Available Repositories pane, click a repository to expand the repository set.
4. Click the **Enable** icon next to the base architecture and release version that you want.

CLI procedure

1. To search for your product, enter the following command:

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

2. List the repository set for the product:

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

3. Enable the repository using either the name or ID number. Include the release version, such as **7Server**, and base architecture, such as **x86_64**.

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

4.7. SYNCHRONIZING REPOSITORIES

You must synchronize repositories to download content into Satellite. You can use this procedure for an initial synchronization of repositories or to synchronize repositories manually as you need.

You can also sync all repositories in an organization. For more information, see Section 4.8, "Synchronizing all repositories in an organization".

Create a sync plan to ensure updates on a regular basis. For more information, see Section 4.24, "Creating a sync plan".

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

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products** and select the product that contains the repositories that you want to synchronize.
2. Select the repositories that you want to synchronize and click **Sync Now**.
3. Optional: To view the progress of the synchronization in the Satellite web UI, navigate to **Content** > **Sync Status** and expand the corresponding product or repository tree.

CLI procedure

- Synchronize an entire product:

```
# hammer product synchronize \
--name "My_Product" \
--organization "My_Organization"
```

- Synchronize an individual repository:

```
# hammer repository synchronize \
--name "My_Repository" \
--organization "My_Organization" \
--product "My Product"
```

4.8. SYNCHRONIZING ALL REPOSITORIES IN AN ORGANIZATION

Use this procedure to synchronize all repositories within an organization.

Procedure

1. Log in to your Satellite Server using SSH.
2. Run the following Bash script:

```
ORG="My_Organization"

for i in $(hammer --no-headers --csv repository list --organization $ORG --fields Id)
do
  hammer repository synchronize --id ${i} --organization $ORG --async
done
```

4.9. DOWNLOAD POLICIES OVERVIEW

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

Satellite Server has the following policies:

Immediate

Satellite Server downloads all metadata and packages during synchronization.

On Demand

Satellite Server downloads only 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.

The **On Demand** policy acts as a *Lazy Synchronization* feature because they save time synchronizing content. The lazy synchronization feature must be used only for Yum repositories. You can add the packages to content views and promote to lifecycle environments as normal.

Capsule Server has 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. Capsule Server fetches and stores packages only on the file system when directly connected clients request them. When you use an **On Demand** download policy, content is downloaded from Satellite Server if it is not available on Capsule Server.

Inherit

Capsule Server inherits the download policy for the repository from the corresponding repository on Satellite Server.

Streamed Download Policy

Streamed Download Policy for Capsules permits Capsules to avoid caching any content. When content is requested from the Capsule, it functions as a proxy and requests the content directly from the Satellite.

4.10. CHANGING THE DEFAULT DOWNLOAD POLICY

You can set the default download policy that Satellite applies to repositories that you create in all organizations.

Depending on whether it is a Red Hat or non-Red Hat custom repository, Satellite uses separate settings. Changing the default value does not change existing settings.

Procedure

1. In the Satellite web UI, navigate to **Administer** > **Settings**.
2. Click the **Content** tab.
3. Change the default download policy depending on your requirements:
   - To change the default download policy for a Red Hat repository, change the value of the **Default Red Hat Repository download policy**setting.
   - To change the default download policy for a custom repository, change the value of the **Default Custom Repository download policy**setting.

CLI procedure

- To change the default download policy for Red Hat repositories to one of **immediate** or **on_demand**, enter the following command:

```
# hammer settings set \
--name default_redhat_download_policy \
--value immediate
```

- To change the default download policy for a non-Red Hat custom repository to one of **immediate** or **on_demand**, enter the following command:

```
# hammer settings set \
--name default_download_policy \
--value immediate
```

4.11. CHANGING THE DOWNLOAD POLICY FOR A REPOSITORY

You can set the download policy for a repository.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Select the required product name.
3. On the **Repositories** tab, click the required repository name, locate the **Download Policy** field, and click the edit icon.
4. From the list, select the required download policy and then click **Save**.

CLI procedure

1. List the repositories for an organization:

```
# hammer repository list \
--organization-label My_Organization_Label
```

2. Change the download policy for a repository to **immediate** or **on_demand**:

```
# hammer repository update \
--download-policy immediate \
--name "My_Repository" \
--organization-label My_Organization_Label \
--product "My_Product"
```

4.12. MIRRORING POLICIES OVERVIEW

Mirroring keeps the local repository exactly in synchronization with the upstream repository. If any content is removed from the upstream repository since the last synchronization, with the next synchronization, it will be removed from the local repository as well.

You can use mirroring policies for finer control over mirroring of repodata and content when synchronizing a repository. For example, if it is not possible to mirror the repodata for a repository, you can set the mirroring policy to mirror only content for this repository.

Satellite Server has the following mirroring policies:

Additive

Neither the content nor the repodata is mirrored. Thus, only new content added since the last synchronization is added to the local repository and nothing is removed.

Content Only

Mirrors only content and not the repodata. Some repositories do not support metadata mirroring, in such cases you can set the mirroring policy to content only to only mirror the content.

Complete Mirroring

Mirrors content as well as repodata. This is the fastest method. This mirroring policy is only available for Yum content.

WARNING

Avoid republishing metadata for repositories with Complete Mirror mirroring policy. This also applies to content views containing repositories with the Complete Mirror mirroring policy.

4.13. CHANGING THE MIRRORING POLICY FOR A REPOSITORY

You can set the mirroring policy for a repository.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Select the product name.
3. On the **Repositories** tab, click the repository name, locate the **Mirroring Policy** field, and click the edit icon.
4. From the list, select a mirroring policy and click **Save**.

CLI procedure

1. List the repositories for an organization:

```
# hammer repository list \
--organization-label My_Organization_Label
```

2. Change the mirroring policy for a repository to **additive**, **mirror_complete**, or **mirror_content_only**:

```
# hammer repository update \
--id 1 \
--mirroring-policy mirror_complete
```

4.14. UPLOADING CONTENT TO CUSTOM RPM REPOSITORIES

You can upload individual RPMs and source RPMs to custom RPM repositories. You can upload RPMs using the Satellite web UI or the Hammer CLI. You must use the Hammer CLI to upload source RPMs.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Click the name of the custom product.
3. In the **Repositories** tab, click the name of the custom RPM repository.
4. Under **Upload Package**, click **Browse...** and select the RPM you want to upload.
5. Click **Upload**.

To view all RPMs in this repository, click the number next to **Packages** under **Content Counts**.

CLI procedure

- Enter the following command to upload an RPM:

```
# hammer repository upload-content \
--id My_Repository_ID \
--path /path/to/example-package.rpm
```

- Enter the following command to upload a source RPM:

```
# hammer repository upload-content \
--content-type srpm \
--id My_Repository_ID \
--path /path/to/example-package.src.rpm
```

When the upload is complete, you can view information about a source RPM by using the commands **hammer srpm list** and **hammer srpm info --id *srpm_ID***.

4.15. REFRESHING CONTENT COUNTS ON CAPSULE

If your Capsules have synchronized content enabled, you can refresh the number of content counts available to the environments associated with the Capsule. This displays the content views inside those environments available to the Capsule. You can then expand the content view to view the repositories associated with that content view version.

Procedure

1. In the Satellite web UI, navigate to **Infrastructure** > **Capsules**, and select the Capsule where you want to see the synchronized content.
2. Select the **Overview** tab.
3. Under **Content Sync**, toggle the **Synchronize** button to do an **Optimized Sync** or a **Complete Sync** to synchronize the Capsule which refreshes the content counts.
4. Select the **Content** tab.
5. Choose an **Environment** to view content views available to those Capsules by clicking >.
6. Expand the content view by clicking > to view repositories available to the content view and the specific version for the environment.
7. View the number of content counts under **Packages** specific to yum repositories.

8. View the number of errata, package groups, files, container tags, container manifests, and Ansible collections under **Additional content**.

9. Click the vertical ellipsis in the column to the right next to the environment and click **Refresh counts** to refresh the content counts synchronized on the Capsule under **Packages**.

4.16. CONFIGURING SELINUX TO PERMIT CONTENT SYNCHRONIZATION ON CUSTOM PORTS

SELinux permits access of Satellite for content synchronization only on specific ports. By default, connecting to web servers running on the following ports is permitted: 80, 81, 443, 488, 8008, 8009, 8443, and 9000.

Procedure

1. On Satellite, to verify the ports that are permitted by SELinux for content synchronization, enter a command as follows:

```
# semanage port -l | grep ^http_port_t
http_port_t     tcp      80, 81, 443, 488, 8008, 8009, 8443, 9000
```

2. To configure SELinux to permit a port for content synchronization, for example 10011, enter a command as follows:

```
# semanage port -a -t http_port_t -p tcp 10011
```

4.17. RECOVERING A CORRUPTED REPOSITORY

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

Optimized Sync

Synchronizes the repository bypassing packages that have no detected differences from the upstream packages.

Complete Sync

Synchronizes all packages regardless of detected changes. Use this option if specific packages could not be downloaded to the local repository even though they exist in the upstream repository.

Verify Content Checksum

Synchronizes all packages and then verifies the checksum of all packages locally. If the checksum of an RPM differs from the upstream, it re-downloads the RPM. This option is relevant only for Yum content. Use this option if you have one of the following errors:

- Specific packages cause a **404** error while synchronizing with **yum**.
- **Package does not match intended download** error, which means that specific packages are corrupted.

Procedure

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

2. Select the product containing the corrupted repository.
3. Select the name of a repository you want to synchronize.
4. To perform optimized sync or complete sync, select **Advanced Sync** from the **Select Action** menu.
5. Select the required option and click **Sync**.
6. Optional: To verify the checksum, click **Verify Content Checksum** from the **Select Action** menu.

CLI procedure

1. Obtain a list of repository IDs:

```
# hammer repository list \
--organization "My_Organization"
```

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

```
# hammer repository synchronize \
--id My_ID
```

   - For the complete synchronization:

```
# hammer repository synchronize \
--id My_ID \
--skip-metadata-check true
```

   - For the validate content synchronization:

```
# hammer repository synchronize \
--id My_ID \
--validate-contents true
```

4.18. RECOVERING CORRUPTED CONTENT ON CAPSULE

If the client is unable to consume content from a published repository to which it has a subscription, the content has been corrupted and needs to be repaired. In case of content corruption on a Capsule, you can recover it by using the **verify-checksum** command in Hammer CLI. The **verify-checksum** command can repair content in a content view, lifecycle environment, repository, or all content on Capsule. You can track the progress of a command by navigating to **Monitor** > **Satellite Tasks** > **Tasks** and searching for the action **Verify checksum for content on smart proxy**

CLI procedure

- To repair content in a content view, run Hammer on your Capsule:

```
$ hammer capsule content verify-checksum \
--id My_Capsule_ID \
--organization-id 1 --content-view-id 3
```

- To repair content in a lifecycle environment, run Hammer on your Capsule:

```
$ hammer capsule content verify-checksum \
--id My_Capsule_ID \
--organization-id 1 --lifecycle-environment-id 1
```

- To repair content in a repository, run Hammer on your Capsule:

```
$ hammer capsule content verify-checksum \
--id My_Capsule_ID \
--organization-id 1 --repository-id 1
```

- To repair all content on Capsule, run the following command:

```
$ hammer capsule content verify-checksum \
--id My_Capsule_ID
```

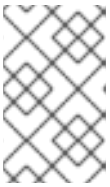
4.19. REPUBLISHING REPOSITORY METADATA

You can republish repository metadata when a repository distribution does not have the content that should be distributed based on the contents of the repository.

Use this procedure with caution. Red Hat recommends a complete repository sync or publishing a new content view version to repair broken metadata.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Select the product that includes the repository for which you want to republish metadata.
3. On the **Repositories** tab, select a repository.
4. To republish metadata for the repository, click **Republish Repository Metadata** from the **Select Action** menu.



NOTE

This action is not available for repositories that use the **Complete Mirroring** policy because the metadata is copied verbatim from the upstream source of the repository.

4.20. REPUBLISHING CONTENT VIEW METADATA

Use this procedure to republish content view metadata.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Select a content view.

3. On the **Versions** tab, select a content view version.
4. To republish metadata for the content view version, click **Republish repository metadata** from the vertical ellipsis icon.

Republishing repository metadata will regenerate metadata for all repositories in the content view version that do not adhere to the **Complete Mirroring** policy.

4.21. ADDING AN HTTP PROXY

Use this procedure to add HTTP proxies to Satellite. You can then specify which HTTP proxy to use for products, repositories, and supported compute resources.

Prerequisites

Your HTTP proxy must allow access to the following hosts:

Host name	Port	Protocol
subscription.rhsm.redhat.com	443	HTTPS
cdn.redhat.com	443	HTTPS
*.akamaiedge.net	443	HTTPS
cert.console.redhat.com (if using Red Hat Insights)	443	HTTPS
api.access.redhat.com (if using Red Hat Insights)	443	HTTPS
cert-api.access.redhat.com (if using Red Hat Insights)	443	HTTPS

If Satellite Server uses a proxy to communicate with subscription.rhsm.redhat.com and cdn.redhat.com then the proxy must not perform SSL inspection on these communications.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Infrastructure** > **HTTP Proxies**.
2. Select **New HTTP Proxy**.
3. In the **Name** field, enter a name for the HTTP proxy.
4. In the **URL** field, enter the URL for the HTTP proxy, including the port number.
5. If your HTTP proxy requires authentication, enter a **Username** and **Password**.
6. Optional: In the **Test URL** field, enter the HTTP proxy URL, then click **Test Connection** to ensure that you can connect to the HTTP proxy from Satellite.
7. Click the **Locations** tab and add a location.

8. Click the **Organization** tab and add an organization.
9. Click **Submit**.

CLI procedure

- On Satellite Server, enter the following command to add an HTTP proxy:

```
# hammer http-proxy create \
--name My_HTTP_Proxy \
--url http-proxy.example.com:8080
```

If your HTTP proxy requires authentication, add the **--username *My_User_Name*** and **--password *My_Password*** options.

For further information, see the Knowledgebase article How to access Red Hat Subscription Manager (RHSM) through a firewall or proxy on the Red Hat Customer Portal.

4.22. CHANGING THE HTTP PROXY POLICY FOR A PRODUCT

For granular control over network traffic, you can set an HTTP proxy policy for each product. A product's HTTP proxy policy applies to all repositories in the product, unless you set a different policy for individual repositories.

To set an HTTP proxy policy for individual repositories, see Section 4.23, "Changing the HTTP proxy policy for a repository".

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products** and select the products that you want to change.
2. From the **Select Action** list, select **Manage HTTP Proxy**.
3. Select an **HTTP Proxy Policy** from the list:
   - **Global Default**: Use the global default proxy setting.
   - **No HTTP Proxy**: Do not use an HTTP proxy, even if a global default proxy is configured.
   - **Use specific HTTP Proxy**: Select an **HTTP Proxy** from the list. You must add HTTP proxies to Satellite before you can select a proxy from this list. For more information, see Section 4.21, "Adding an HTTP proxy" .
4. Click **Update**.

4.23. CHANGING THE HTTP PROXY POLICY FOR A REPOSITORY

For granular control over network traffic, you can set an HTTP proxy policy for each repository. To use the CLI instead of the Satellite web UI, see the CLI procedure.

To set the same HTTP proxy policy for all repositories in a product, see Section 4.22, "Changing the HTTP proxy policy for a product".

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products** and click the name of the product that contains the repository.
2. In the **Repositories** tab, click the name of the repository.
3. Locate the **HTTP Proxy** field and click the edit icon.
4. Select an **HTTP Proxy Policy** from the list:
   - **Global Default**: Use the global default proxy setting.
   - **No HTTP Proxy**: Do not use an HTTP proxy, even if a global default proxy is configured.
   - **Use specific HTTP Proxy**: Select an **HTTP Proxy** from the list. You must add HTTP proxies to Satellite before you can select a proxy from this list. For more information, see Section 4.21, "Adding an HTTP proxy" .
5. Click **Save**.

CLI procedure

- On Satellite Server, enter the following command, specifying the HTTP proxy policy you want to use:

```
# hammer repository update \
--http-proxy-policy HTTP_Proxy_Policy \
--id Repository_ID
```

  Specify one of the following options for **--http-proxy-policy**:

  - **none**: Do not use an HTTP proxy, even if a global default proxy is configured.
  - **global_default_http_proxy**: Use the global default proxy setting.
  - **use_selected_http_proxy**: Specify an HTTP proxy using either **--http-proxy *My_HTTP_Proxy_Name*** or **--http-proxy-id *My_HTTP_Proxy_ID***. To add a new HTTP proxy to Satellite, see Section 4.21, "Adding an HTTP proxy" .

4.24. CREATING A SYNC PLAN

A sync plan checks and updates the content at a scheduled date and time. In Satellite, you can create a sync plan and assign products to the plan.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Sync Plans** and click **New Sync Plan**.
2. In the **Name** field, enter a name for the plan.
3. Optional: In the **Description** field, enter a description of the plan.
4. From the **Interval** list, select the interval at which you want the plan to run.
5. From the **Start Date** and **Start Time** lists, select when to start running the synchronization plan.

6. Click **Save**.

CLI procedure

1. To create the synchronization plan, enter the following command:

```
# hammer sync-plan create \
--description "My_Description" \
--enabled true \
--interval daily \
--name "My_Products" \
--organization "My_Organization" \
--sync-date "2023-01-01 01:00:00"
```

2. View the available sync plans for an organization to verify that the sync plan has been created:

```
# hammer sync-plan list --organization "My_Organization"
```

4.25. ASSIGNING A SYNC PLAN TO A PRODUCT

A sync plan checks and updates the content at a scheduled date and time. In Satellite, you can assign a sync plan to products to update content regularly.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content > Products**.
2. Select a product.
3. On the **Details** tab, select a **Sync Plan** from the drop down menu.

CLI procedure

1. Assign a sync plan to a product:

```
# hammer product set-sync-plan \
--name "My_Product_Name" \
--organization "My_Organization" \
--sync-plan "My_Sync_Plan_Name"
```

4.26. ASSIGNING A SYNC PLAN TO MULTIPLE PRODUCTS

Use this procedure to assign a sync plan to the products in an organization that have been synchronized at least once and contain at least one repository.

Procedure

1. Run the following Bash script:

```
ORG="My_Organization"
SYNC_PLAN="daily_sync_at_3_a.m"
```

```
hammer sync-plan create --name $SYNC_PLAN --interval daily --sync-date "2023-04-5
03:00:00" --enabled true --organization $ORG
for i in $(hammer --no-headers --csv --csv-separator="|" product list --organization $ORG --
per-page 999 | grep -vi not_synced | awk -F'|' '$5 != "0" { print $1}')
do
  hammer product set-sync-plan --sync-plan $SYNC_PLAN --organization $ORG --id $i
done
```

2. After executing the script, view the products assigned to the sync plan:

```
# hammer product list --organization $ORG --sync-plan $SYNC_PLAN
```

4.27. BEST PRACTICES FOR SYNC PLANS

- Add sync plans to products and regularly synchronize content to keep the load on Satellite low during synchronization. Synchronize content rather more often than less often. For example, setup a sync plan to synchronize content every day rather than only once a month.
- Automate the creation and update of sync plans by using a Hammer script or an Ansible Playbook.
- Distribute synchronization tasks over several hours to reduce the task load by creating multiple sync plans with the **Custom Cron** tool.

Table 4.1. Cron expression examples

Cron expression	Explanation
0 22 * * 1-5	every day at 22:00 from Monday to Friday
30 3 * * 6,0	at 03:30 every Saturday and Sunday
30 2 8-14 * *	at 02:30 every day between the 8th and the 14th days of the month

4.28. LIMITING SYNCHRONIZATION CONCURRENCY

By default, each Repository Synchronization job can fetch up to ten files at a time. This can be adjusted on a per repository basis.

Increasing the limit may improve performance, but can cause the upstream server to be overloaded or start rejecting requests. If you are seeing Repository syncs fail due to the upstream servers rejecting requests, you may want to try lowering the limit.

CLI procedure

```
# hammer repository update \
--download-concurrency 5 \
--id Repository_ID \
--organization "My_Organization"
```

4.29. IMPORTING A CUSTOM GPG KEY

When clients are consuming signed custom content, ensure that the clients are configured to validate the installation of packages with the appropriate GPG Key. This helps to ensure that only packages from authorized sources can be installed.

Red Hat content is already configured with the appropriate GPG key and thus GPG Key management of Red Hat Repositories is not supported.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Prerequisites

Ensure that you have a copy of the GPG key used to sign the RPM content that you want to use and manage in Satellite. Most RPM distribution providers provide their GPG Key on their website. You can also extract this manually from an RPM:

1. Download a copy of the version specific repository package to your local machine:

```
$ wget http://www.example.com/9.5/example-9.5-2.noarch.rpm
```

2. Extract the RPM file without installing it:

```
$ rpm2cpio example-9.5-2.noarch.rpm | cpio -idmv
```

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

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 of your repository 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.
   If your custom repository contains content signed by multiple GPG keys, you must enter all required GPG keys in the **Content Credential Contents** field with new lines between each key, for example:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFy/HE4BEADttv2TCPzVrre+aJ9f5QsR6oWZMm7N5Lwxjm5x5zA9BLiPPGFN
4aTUR/g+K1S0aqCU+ZS3Rnxb+6fnBxD+COH9kMqXHi3M5UNzbp5WhCdUpISXjjpU
XIFFWBPuBfyr/FKRknFH15P+9kLZLxCpVZZLsweLWCuw+JKCMmnA
=F6VG
-----END PGP PUBLIC KEY BLOCK-----

-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFw467UBEACmREzDeK/kuScCmfJfHJa0Wgh/2fbJLLt3KSvsgDhORIptf+PP
OTFDlKuLkJx99ZYG5xMnBG47C7ByoMec1j94YeXczuBbynOyyPlvduma/zf8oB9e
Wl5GnzcLGAnUSRamfqGUWcyMMinHHIKIc1X1P4I=
=WPpI
-----END PGP PUBLIC KEY BLOCK-----
```

4. Click **Save**.

CLI procedure

1. Copy the GPG key to your Satellite Server:

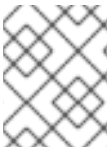
```
$ scp ~/etc/pki/rpm-gpg/RPM-GPG-KEY-EXAMPLE-95 root@satellite.example.com:~/.
```

2. Upload the GPG key to Satellite:

```
# hammer content-credentials create \
--content-type gpg_key \
--name "My_GPG_Key" \
--organization "My_Organization" \
--path ~/RPM-GPG-KEY-EXAMPLE-95
```

4.30. RESTRICTING A CUSTOM REPOSITORY TO A SPECIFIC OPERATING SYSTEM OR ARCHITECTURE IN SATELLITE

You can configure Satellite to make a custom repository available only on hosts with a specific operating system version or architecture. For example, you can restrict a custom repository only to Red Hat Enterprise Linux 9 hosts.



NOTE

Only restrict architecture and operating system version for custom products. Satellite applies these restrictions automatically for Red Hat repositories.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Click the product that contains the repository sets you want to restrict.
3. In the **Repositories** tab, click the repository you want to restrict.
4. In the **Publishing Settings** section, set the following options:
   - Set **Restrict to OS version** to restrict the operating system version.
   - Set **Restrict to architecture** to restrict the architecture.

CHAPTER 5. RESTRICTING HOSTS' ACCESS TO CONTENT

Satellite offers multiple options for restricting host access to content. To give hosts access to a specific subset of the content managed by Satellite, you can use the following strategies. Red Hat recommends to consider implementing the strategies in the order listed here:

Content views and lifecycle environments

Use content views and lifecycle environments, incorporating content view filters as needed. For more information about content views, see Chapter 7, *Managing content views*.

For more information about lifecycle environments, see Chapter 6, *Managing application lifecycles*.

Content overrides

By default, content hosted by Satellite can be either enabled or disabled. In custom products, repositories are always disabled by default, while Red Hat products can be either enabled or disabled by default depending on the specific repository. Enabling a repository gives the host access to the repository packages or other content, allowing hosts to download and install the available content. If a repository is disabled, the host is not able to access the repository content. A content override provides you with the option to override the default enablement value of either **Enabled** or **Disabled** for any repository. You can add content overrides to hosts or activation keys.

For more information about adding content overrides to hosts, see Enabling and Disabling Repositories on Hosts in *Managing hosts*.

For more information about adding content overrides to activation keys, see Section 9.5, "Enabling and disabling repositories on activation key".

Composite content views

You can use composite content views to combine and give hosts access to the content from multiple content views. For more information about composite content views, see Section 7.9, "Creating a composite content view".

Architecture and operating system version restrictions

In custom products, you can set restrictions on the architecture and operating system versions for **yum** repositories on which the product will be available. For example, if you restrict a custom repository to **Red Hat Enterprise Linux 8**, it is only available on hosts running Red Hat Enterprise Linux 8. Architecture and operating system version restrictions hold the highest priority among all other strategies. They cannot be overridden or invalidated by content overrides, changes to content views, or changes to lifecycle environments. For this reason, Red Hat recommends considering the other strategies mentioned before that use architecture or operating system version restrictions. Red Hat repositories set architecture and OS version restrictions automatically.

Release version

Certain Red Hat repositories, such as the Red Hat Enterprise Linux dot release repositories, include a **Release version** in their repository metadata. The release version is then compared with the release version specified in the **System purpose** properties of the host. Access to content may be limited or restricted based on this comparison. For more information about setting system purpose attributes, see Creating a Host in Red Hat Satellite in *Managing hosts*.

Incorporating all strategies

A particular package or repository is available to a host only if all of the following are true:

- The repository is included in the host's content view and lifecycle environment.

- The host's content view has been published after the repository was added to it.
- The repository has not been filtered out by a content view filter.
- The repository is enabled by default or overridden to **Enabled** by using a content override.
- The repository has no architecture or operating system version restrictions or it has architecture or operating system version restrictions that match the host.
- For certain Red Hat repositories either no release version is set or the release version matches that of the host.

Using activation keys

Using activation keys can simplify the workflow for some of these strategies. You can use activation keys to perform the following actions:

- Assign hosts to content views and lifecycle environments.
- Add content overrides to hosts.
- Set system purpose attributes on hosts, including release version.

Activation keys only affect hosts during registration. If a host is already registered, the above attributes can be changed individually for each host or through content host bulk actions. For more information, see Managing Activation Keys in *Managing content*.

CHAPTER 6. MANAGING APPLICATION LIFECYCLES

This chapter outlines the application lifecycle in Satellite and how to create and remove application lifecycles for Satellite and Capsule.

6.1. INTRODUCTION TO APPLICATION LIFECYCLE

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

- Development
- Production

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

- Development
- Testing
- Beta Release
- Production

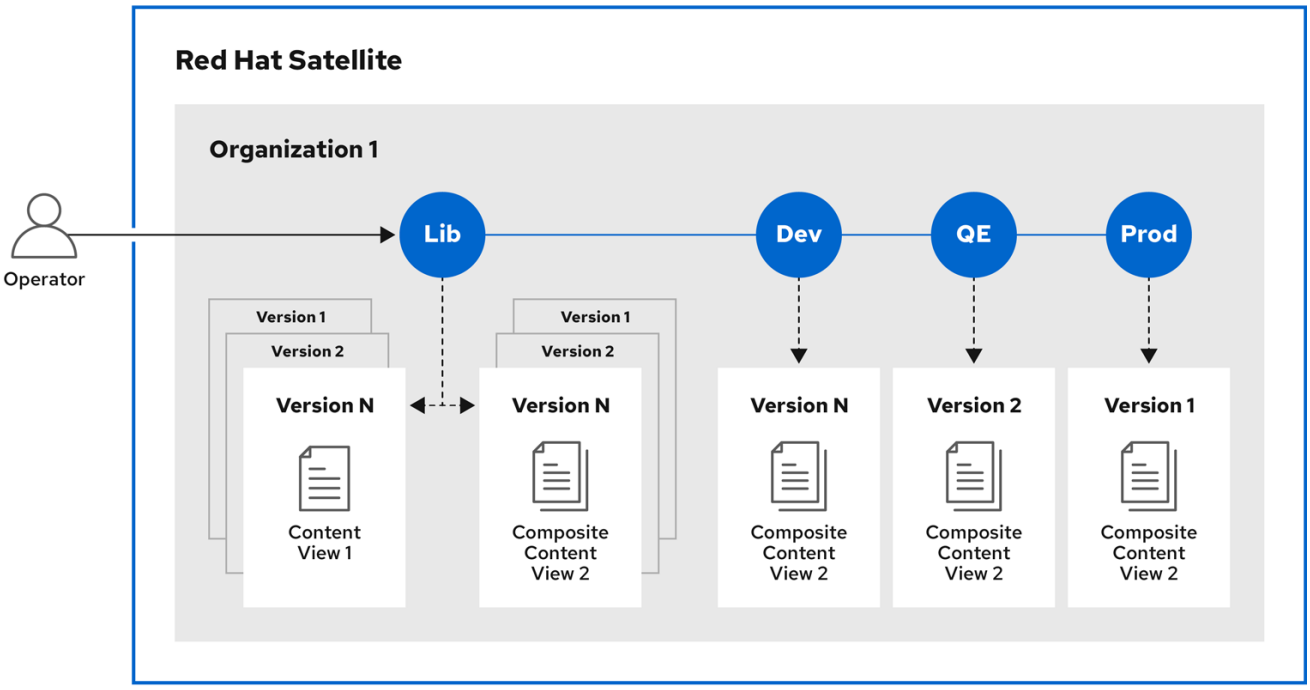
Satellite provides methods to customize each application lifecycle stage so that it suits your specifications.

Each stage in the application lifecycle is called an *environment* in Satellite. Each environment uses a specific collection of content. Satellite defines these content collections as a content view. Each content view acts as a filter where you can define what repositories, and packages to include in a particular environment. This provides a method for you to define specific sets of content to designate to each environment.

For example, an email server might only require a simple application lifecycle 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, you can set the production-level server to use the new packages.

Another example is a development lifecycle for a software product. To develop a new piece of software in a development environment, test it in a quality assurance environment, pre-release as a beta, then release the software as a production-level application.

Figure 6.1. Satellite application lifecycle



6.2. CONTENT PROMOTION ACROSS THE APPLICATION LIFECYCLE

In the application lifecycle chain, when content moves from one environment to the next, this is called *promotion*.

Example: Content promotion across Satellite lifecycle environments

Each environment contains a set of systems registered to Red Hat Satellite. These systems only have access to repositories relevant to their environment. When you 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.

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

After completing development on the patch, you promote the package to the Testing environment so the Quality Engineering team can review the patch. The application lifecycle then contains the following package versions in each environment:

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

While the Quality Engineering team reviews the patch, the Development team starts work on *example_software* 2.0. This results in the following application lifecycle:

Development	Testing	Production
***example_software*-2.0-0.noarch.rpm**	*example_software*-1.1-0.noarch.rpm	*example_software*-1.0-0.noarch.rpm

The Quality Engineering team completes their review of the patch. Now *example_software* 1.1 is ready to release. You promote 1.1 to the *Production* environment:

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

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

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

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

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

For more information, see Section 7.7, “Promoting a content view” .

6.3. BEST PRACTICES FOR LIFECYCLE ENVIRONMENTS

- Use multiple lifecycle environment paths to implement multiple sequential stages of content consumption. Each stage contains a defined set of content, for example in the *Production* lifecycle environment.
- Automate the creation of lifecycle environments by using a Hammer script or an Ansible Playbook.
- Default use case: Fixed stages in each lifecycle environment paths, for example *Development*, *Test*, and *Production*.
    - Promote content views to lifecycle environments, for example, from *Test* to *Production*. All content hosts consuming this content view or composite content view are able to install packages from the *Production* lifecycle environment. Note that these packages are not installed or updated automatically.

    - If you encounter errors during patching content hosts, attach the host to a previous version of the content view. This only affects the availability of packages but does not downgrade installed packages.

- Alternative use case: Using stages in lifecycle environments for fixed content, for example, quarterly updates, and only publishing new minor versions with incremental updates from errata.
    - When patching content hosts, change the lifecycle environment from **2023-Q4** to **2024-Q1** using the Satellite web UI, Satellite API, Hammer CLI, or an activation key.
    - Advantage: You can directly see which software packages a hosts receives by looking at its lifecycle environment.
    - Disadvantage: Promoting content is less dynamic without clearly defined stages such as *Development*, *Test*, and *Production*.
- Use multiple lifecycle environment paths to define multiple stages for different environments, for example to decouple web server and database hosts.
- Capsule Servers use lifecycle environments to synchronize content. They synchronize content more efficiently if you split content into multiple lifecycle environment paths. If a specific Capsule Server only serves content for one operating system in a single lifecycle environment path, it only synchronizes required content.

6.4. CREATING A LIFECYCLE ENVIRONMENT PATH

To create an application lifecycle for developing and releasing software, use the *Library* environment as the initial environment to create environment paths. Then optionally add additional environments to the environment paths.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Lifecycle Environments**.
2. Click **New Environment Path** to start a new application lifecycle.
3. In the **Name** field, enter a name for your environment.
4. In the **Description** field, enter a description for your environment.
5. Click **Save**.
6. Optional: To add an environment to the environment path, click **Add New Environment**, complete the **Name** and **Description** fields, and select the prior environment from the **Prior Environment** list.

CLI procedure

1. To create an environment path, enter the **hammer lifecycle-environment create** command and specify the Library environment with the **--prior** option:

```
# hammer lifecycle-environment create \
--name "Environment Path Name" \
--description "Environment Path Description" \
--prior "Library" \
--organization "My_Organization"
```

2. Optional: To add an environment to the environment path, enter the **hammer lifecycle-environment create** command and specify the parent environment with the **--prior** option:

```
# hammer lifecycle-environment create \
--name "Environment Name" \
--description "Environment Description" \
--prior "Prior Environment Name" \
--organization "My_Organization"
```

3. To view the chain of the lifecycle environment, enter the following command:

```
# hammer lifecycle-environment paths --organization "My_Organization"
```

6.5. ADDING LIFECYCLE ENVIRONMENTS TO CAPSULE SERVERS

If your Capsule Server has the content functionality enabled, you must add an environment so that Capsule can synchronize content from Satellite Server and provide content to host systems.

Do not assign the *Library* lifecycle environment to your Capsule Server because it triggers an automated Capsule sync every time the CDN updates a repository. This might consume multiple system resources on Capsules, network bandwidth between Satellite and Capsules, and available disk space on Capsules.

You can use Hammer CLI on Satellite Server or the Satellite web UI.

Procedure

1. In the Satellite web UI, navigate to **Infrastructure** > **Capsules**, and select the Capsule that you want to add a lifecycle to.
2. Click **Edit** and click the **Lifecycle Environments** tab.
3. From the left menu, select the lifecycle environments that you want to add to Capsule and click **Submit**.
4. To synchronize the content on the Capsule, click the **Overview** tab and click **Synchronize**.
5. Select either **Optimized Sync** or **Complete Sync**.
   For definitions of each synchronization type, see Recovering a Repository.

CLI procedure

1. To display a list of all Capsule Servers, on Satellite Server, enter the following command:

```
# hammer capsule list
```

   Note the Capsule ID of the Capsule to which you want to add a lifecycle.

2. Using the ID, verify the details of your Capsule:

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

3. To view the lifecycle environments available for your Capsule Server, enter the following command and note the ID and the organization name:

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

4. Add the lifecycle environment to your Capsule Server:

```
# hammer capsule content add-lifecycle-environment \
--id My_capsule_ID \
--lifecycle-environment-id My_Lifecycle_Environment_ID
--organization "My_Organization"
```

Repeat for each lifecycle environment you want to add to Capsule Server.

5. Synchronize the content from Satellite to Capsule.

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

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

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

```
# hammer capsule content synchronize \
--id My_capsule_ID
--lifecycle-environment-id My_Lifecycle_Environment_ID
```

- To synchronize all content from your Satellite Server to your Capsule Server without checking metadata:

```
# hammer capsule content synchronize \
--id My_capsule_ID \
--skip-metadata-check true
```

This equals selecting **Complete Sync** in the Satellite web UI.

6.6. REMOVING LIFECYCLE ENVIRONMENTS FROM SATELLITE SERVER

Use this procedure to remove a lifecycle environment.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle Environments**.
2. Click the name of the lifecycle environment that you want to remove, and then click **Remove Environment**.
3. Click **Remove** to remove the environment.

CLI procedure

1. List the lifecycle environments for your organization and note the name of the lifecycle environment you want to remove:

```
# hammer lifecycle-environment list \
--organization "My_Organization"
```

2. Use the **hammer lifecycle-environment delete** command to remove an environment:

```
# hammer lifecycle-environment delete \
--name "My_Environment" \
--organization "My_Organization"
```

6.7. REMOVING LIFECYCLE ENVIRONMENTS FROM CAPSULE SERVER

When lifecycle environments are no longer relevant to the host system or environments are added incorrectly to Capsule Server, you can remove the lifecycle environments from Capsule Server.

You can use both the Satellite web UI and the Hammer CLI to remove lifecycle environments from Capsule.

Procedure

1. In the Satellite web UI, navigate to **Infrastructure** > **Capsules**, and select the Capsule that you want to remove a lifecycle from.
2. Click **Edit** and click the **Lifecycle Environments** tab.
3. From the right menu, select the lifecycle environments that you want to remove from Capsule, and then click **Submit**.
4. To synchronize Capsule's content, click the **Overview** tab, and then click **Synchronize**.
5. Select either **Optimized Sync** or **Complete Sync**.

CLI procedure

1. Select Capsule Server that you want from the list and take note of its **id**:

```
# hammer capsule list
```

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

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

3. Verify the list of lifecycle environments currently attached to Capsule Server and take note of the **Environment ID**:

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

4. Remove the lifecycle environment from Capsule Server:

```
# hammer capsule content remove-lifecycle-environment \
--id My_Capsule_ID \
--lifecycle-environment-id My_Lifecycle_Environment_ID
```

Repeat this step for every lifecycle environment that you want to remove from Capsule Server.

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

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

CHAPTER 7. MANAGING CONTENT VIEWS

Red Hat Satellite uses content views to allow your hosts access to a deliberately curated subset of content. To do this, you must define which repositories to use and then apply certain filters to the content.

The general workflow for creating content views for filtering and creating snapshots is as follows:

1. Create a content view.
2. Add one or more repositories that you want to the content view.
3. Optional: Create one or more filters to refine the content of the content view. For more information, see Section 7.13, “Content filter examples” .
4. Optional: Resolve any package dependencies for a content view. For more information, see Section 7.11, “Resolving package dependencies” .
5. Publish the content view.
6. Optional: Promote the content view to another environment. For more information, see Section 7.7, “Promoting a content view” .
7. Attach the content host to the content view.

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.

Hosts can only be associated with a single content view. To associate a host with multiple content views, create a composite content view. For more information, see Section 7.9, “Creating a composite content view”.

7.1. CONTENT VIEWS IN RED HAT SATELLITE

A content view is a deliberately curated subset of content that your hosts can access. By creating a content view, you can define the software versions used by a particular environment or Capsule Server.

Each content view creates a set of repositories across each environment. Your Satellite Server stores and manages these repositories. For example, you can create content views in the following ways:

- A content view with older package versions for a production environment and another content view with newer package versions for a *Development* environment.
- A content view with a package repository required by an operating system and another content view with a package repository required by an application.
- A composite content view for a modular approach to managing content views. For example, you can use one content view for content for managing an operating system and another content view for content for managing an application. By creating a composite content view that combines both content views, you create a new repository that merges the repositories from each of the content views. However, the repositories for the content views still exist and you can keep managing them separately as well.

Default Organization View

A *Default Organization View* is an application-controlled content view for all content that is synchronized to Satellite. You can register a host to the *Library* environment on Satellite to consume the *Default Organization View* without configuring content views and lifecycle environments.

Promoting a content view across environments

When you promote a content view from one environment to the next environment in the application lifecycle, Satellite updates the repository and publishes the packages.

Example 7.1. Promoting a package from *Development* to *Testing*

The repositories for *Testing* and *Production* contain the ***my-software*-1.0-0.noarch.rpm** package:

	Development	Testing	Production
Version of the content view	Version 2	Version 1	Version 1
Contents of the content view	*my-software*-1.1-0.noarch.rpm	*my-software*-1.0-0.noarch.rpm	*my-software*-1.0-0.noarch.rpm

If you promote Version 2 of the content view from *Development* to *Testing*, the repository for *Testing* updates to contain the ***my-software*-1.1-0.noarch.rpm** package:

	Development	Testing	Production
Version of the content view	Version 2	**Version 2**	Version 1
Contents of the content view	*my-software*-1.1-0.noarch.rpm	***my-software*-1.1-0.noarch.rpm**	*my-software*-1.0-0.noarch.rpm

This ensures hosts are designated to a specific environment but receive updates when that environment uses a new version of the content view.

7.2. BEST PRACTICES FOR CONTENT VIEWS

- Content views that bundle content, such as Red Hat Enterprise Linux and additional software like **Apache-2.4** or **PostgreSQL-16.2**, are easier to maintain. Content views that are too small require more maintenance."
- If you require daily updated content, use the content view **Default Organization View**, which contains the latest synchronized content from all repositories and is available in the Library lifecycle environment.
- Restrict composite content views to situations that require greater flexibility, for example, if you update one content view on a weekly basis and another content view on a monthly basis.

- If you use composite content views, first publish the content views and then publish the composite content views. The more content views you bundle into composite content views, the more effort is needed to change or update content.
- Setting a lifecycle environment for content views is unnecessary if they are solely bundled to a composite content view.
- Automate creating and publishing composite content views and lifecycle environments by using a Hammer script or an Ansible Playbook. Use cron jobs, systemd timers, or recurring logics for more visibility.
- Add the changes and date to the description of each published content view or composite content view version. The most recent activity, such as moving content to a new lifecycle environment, is displayed by date in the Satellite web UI, regardless of the latest changes to the content itself.
- Publishing a new content view or composite content view creates a new major version. Incremental errata updates increment the minor version. Note that you cannot change or reset this counter.

7.3. BEST PRACTICES FOR PATCHING CONTENT HOSTS

- Registering hosts to Satellite requires Red Hat Satellite Client 6, which contains the **subscription-manager** package, **katello-host-tools** package, and their dependencies. For more information, see Registering hosts in *Managing hosts*.
- Use the Satellite web UI to install, upgrade, and remove packages from hosts. You can update content hosts with job templates using SSH and Ansible.
- Apply errata on content hosts using the Satellite web UI. When patching packages on hosts using the default package manager, Satellite receives a list of packages and repositories to recalculate applicable errata and available updates.
- Modify or replace job templates to add custom steps. This allows you to run commands or execute scripts on hosts.
- When running bulk actions on hosts, bundle them by major operating system version, especially when upgrading packages.
- Select **via remote execution – customize first** to define the time when patches are applied to hosts when performing bulk actions.
- You cannot apply errata to packages that are not part of the repositories on Satellite and the attached content view.
- Modifications to installed packages using **rpm** or **dpkg** are sent to Satellite with the next run of **apt**, **yum**, or **zypper**.

7.4. CREATING A CONTENT VIEW

Use this procedure to create a simple content view. To use the CLI instead of the Satellite web UI, see the CLI procedure.

Prerequisites

While you can stipulate whether you want to resolve any package dependencies on a content view by

content view basis, you might want to change the default Satellite settings to enable or disable package resolution for all content views. For more information, see Section 7.11, "Resolving package dependencies".

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Click **Create content view**.
3. In the **Name** field, enter a name for the view. Satellite automatically completes the **Label** field from the name you enter.
4. In the **Description** field, enter a description of the view.
5. In the **Type** field, select a **Content view** or a **Composite content view**.
6. Optional: If you want to solve dependencies automatically every time you publish this content view, select the **Solve dependencies** checkbox. Dependency solving slows the publishing time and might ignore any content view filters you use. This can also cause errors when resolving dependencies for errata.
7. Click **Create content view**.

Content view steps

1. Click **Create content view** to create the content view.
2. In the **Repositories** tab, select the repository from the **Type** list that you want to add to your content view, select the checkbox next to the available repositories you want to add, then click **Add repositories**.
3. Click **Publish new version** and in the **Description** field, enter information about the version to log changes.
4. Optional: You can enable a promotion path by clicking **Promote** to **Select a lifecycle environment from the available promotion paths to promote new version**.
5. Click **Next**.
6. On the **Review** page, you can review the environments you are trying to publish.
7. Click **Finish**.

You can view the content view on the **Content Views** page. To view more information about the content view, click the content view name. To register a host to your content view, see Registering Hosts in *Managing hosts*.

CLI procedure

1. Obtain a list of repository IDs:

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

2. Create the content view and add repositories:

```
# hammer content-view create \
--description "My_Content_View" \
--name "My_Content_View" \
--organization "My_Organization" \
--repository-ids 1,2
```

For the **--repository-ids** option, you can find the IDs in the output of the **hammer repository list** command.

3. Publish the view:

```
# hammer content-view publish \
--description "My_Content_View" \
--name "My_Content_View" \
--organization "My_Organization"
```

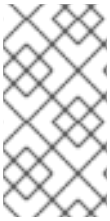
4. Optional: To add a repository to an existing content view, enter the following command:

```
# hammer content-view add-repository \
--name "My_Content_View" \
--organization "My_Organization" \
--repository-id repository_ID
```

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

7.5. COPYING A CONTENT VIEW

You can copy a content view in the Satellite web UI or you can use the Hammer CLI to copy an existing content view into a new content view. To use the CLI instead of the Satellite web UI, see the CLI procedure.



NOTE

A copied content view does not have the same history as the original content view. Version 1 of the copied content view begins at the last version of the original content view. As a result, you cannot promote an older version of a content view from the copied content view.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Select the content view you want to copy.
3. Click the vertical ellipsis icon and click **Copy**.
4. In the **Name** field, enter a name for the new content view and click **Copy content view**.

Verification

- The copied content view appears on the **Content views** page.

CLI procedure

- Copy the content view by using Hammer:

```
# hammer content-view copy --name My_original_CV_name \
--new-name My_new_CV_name
```

Verification

- The Hammer command reports:

```
# hammer content-view copy --id=5 --new-name="mixed_copy"
Content view copied.
```

7.6. VIEWING MODULE STREAMS

In Satellite, you can view the module streams of the repositories in your content views.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to a published version of a **Content View** > **Module Streams** to view the module streams that are available for the Content Types.
2. Use the **Search** field to search for specific modules.
3. To view the information about the module, click the module and its corresponding tabs to include **Details**, **Repositories**, **Profiles**, and **Artifacts**.

CLI procedure

1. List all organizations:

```
# hammer organization list
```

2. View all module streams for your organization:

```
# hammer module-stream list \
--organization-id My_Organization_ID
```

7.7. PROMOTING A CONTENT VIEW

Use this procedure to promote content views across different lifecycle environments. To use the CLI instead of the Satellite web UI, see the CLI procedure.

Permission requirements for content view promotion

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 can promote.

The **promote_or_remove_content_views_to_environment** permission restricts the environments to which a user can promote content views.

With these permissions you can assign users permissions to promote certain content views to certain environments, but not to other environments. For example, you can limit a user so that they are permitted to promote to test environments, but not to production environments.

You must assign both permissions to a user to allow them to promote content views.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Select the content view that you want to promote.
3. Select the version that you want to promote, click the vertical ellipsis icon, and click **Promote**.
4. Select the environment where you want to promote the content view and click **Promote**.

Now the repository for the content view appears in all environments.

CLI procedure

- Promote the content view using Hammer for each lifecycle environment:

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

  Now the database content is available in all environments.

- Alternatively, you can promote content views across all lifecycle environments within an organization using the following Bash script:

```
ORG="My_Organization"
CVV_ID=My_Content_View_Version_ID

for i in $(hammer --no-headers --csv lifecycle-environment list --organization $ORG | awk -F, {'print $1'} | sort -n)
do
  hammer content-view version promote --organization $ORG --to-lifecycle-environment-id $i --id $CVV_ID
done
```

Verification

- Display information about your content view version to verify that it is promoted to the required lifecycle environments:

```
# hammer content-view version info --id My_Content_View_Version_ID
```

Next steps

- To register a host to your content view, see Registering Hosts in *Managing hosts*.

7.8. COMPOSITE CONTENT VIEWS OVERVIEW

A composite content view combines the content from several content views. For example, you might have separate content views to manage an operating system and an application individually. You can use a composite content view 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 also exists for the combined content.

If you want to develop an application that supports different database servers. The *example_application* appears as:

example_software
Application
Database
Operating System

Example of four separate content views:

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

From the previous content views, you can create two composite content views.

Example composite content view for a PostgreSQL database:

Composite content view 1 – *example_software* on PostgreSQL
example_software (Application)
PostgreSQL (Database)
Red Hat Enterprise Linux (Operating System)

Example composite content view for a MariaDB:

Composite content view 2 – *example_software* on MariaDB
example_software (Application)
MariaDB (Database)
Red Hat Enterprise Linux (Operating System)

Each content view is then managed and published separately. When you create a version of the application, you 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.

Repository restrictions

Docker repositories cannot be included more than once in a composite content view. For example, if you attempt to include two content views that contain the same docker repository in a composite content view, Satellite Server reports an error.

7.9. CREATING A COMPOSITE CONTENT VIEW

Use this procedure to create a composite content view. To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Click **Create content view**.
3. In the **Create content view** window, enter a name for the view in the **Name** field. Red Hat Satellite automatically completes the **Label** field from the name you enter.
4. Optional: In the **Description** field, enter a description of the view.
5. On the **Type** tab, select **Composite content view**.
6. Optional: If you want to automatically publish a new version of the composite content view when a content view is republished, select the **Auto publish** checkbox.
7. Click **Create content view**.
8. On the **Content views** tab, select the content views that you want to add to the composite content view, and then click **Add content views**.
9. In the **Add content views** window, select the version of each content view.
10. Optional: If you want to automatically update the content view to the latest version, select the **Always update to latest version** checkbox.
11. Click **Add**, then click **Publish new version**.

12. Optional: In the **Description** field, enter a description of the content view.
13. In the **Publish** window, set the **Promote** switch, then select the lifecycle environment.
14. Click **Next**, then click **Finish**.

CLI procedure

1. Before you create the composite content views, list the version IDs for your existing content views:

```
# hammer content-view version list \
--organization "My_Organization"
```

2. Create a new composite content view. 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 "Example_Composite_Content_View" \
--description "Example composite content view" \
--organization "My_Organization"
```

3. Add a content view to the composite content view. You can identify content view, content view version, and Organization in the commands by either their ID or their name. To add multiple content views to the composite content view, repeat this step for every content view you want to include.

- If you have the **Always update to latest version** option enabled for the content view:

```
# hammer content-view component add \
--component-content-view-id Content_View_ID \
--composite-content-view "Example_Composite_Content_View" \
--latest \
--organization "My_Organization"
```

- If you have the **Always update to latest version** option disabled for the content view:

```
# hammer content-view component add \
--component-content-view-id Content_View_ID \
--composite-content-view "Example_Composite_Content_View" \
--component-content-view-version-id Content_View_Version_ID \
--organization "My_Organization"
```

4. Publish the composite content view:

```
# hammer content-view publish \
--name "Example_Composite_Content_View" \
--description "Initial version of composite content view" \
--organization "My_Organization"
```

5. Promote the composite content view across all environments:

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

7.10. CONTENT FILTER OVERVIEW

Content views also use filters to include or restrict certain Yum content. Without these filters, a content view includes everything from the selected repositories.

There are two types of content filters:

Table 7.1. Filter types

Filter Type	Description
Include	You start with no content, then select which content to add from the selected repositories. Use this filter to combine multiple content items.
Exclude	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 while excluding certain packages. The filter uses all content in the repository except for the content you select.

Include and Exclude filter combinations

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.

Content types

You can filter content based on the following content types:

Table 7.2. Content types

Content Type	Description
RPM	Filter packages based on their name and version number. The **RPM** option filters non-modular RPM packages and errata. Source RPMs are not affected by this filter and will still be available in the content view.

Content Type	Description
Package Group	Filter packages based on package groups. 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.
Module Streams	Select whether to include or exclude specific module streams. The **Module Streams** option filters modular RPMs and errata, but does not filter non-modular content that is associated with the selected module stream.
Container Image Tag	Select whether to include or exclude specific container image tags.

7.11. RESOLVING PACKAGE DEPENDENCIES

Satellite can add dependencies of packages in a content view to the dependent repository when publishing the content view. To configure this, you can enable *dependency solving*.

For example, dependency solving is useful when you incrementally add a single package to a content view version. You might need to enable dependency solving to install that package.

However, dependency solving is unnecessary in most situations. For example:

- When incrementally adding a security errata to a content view, dependency solving can cause significant delays to content view publication without major benefits.
- Packages from a newer erratum might have dependencies that are incompatible with packages from an older content view version. Incrementally adding the erratum by solving dependencies might result in the inclusion of unwanted packages. As an alternative, consider updating the content view.

NOTE

Dependency solving only considers packages within the repositories of the content view. It does not consider packages installed on clients. For example, if a content view includes only AppStream, dependency solving does not include dependent BaseOS content at publish time.

For more information, see Limitations to Repository Dependency Resolution in *Managing content*.

Dependency solving can lead to the following problems:

Significant delay in content view publication

Satellite examines every repository in a content view for dependencies. Therefore, publish time increases with more repositories.

To mitigate this problem, use multiple content views with fewer repositories and combine them into composite content views.

Ignored content view filters on dependent packages

Satellite prioritizes resolving package dependencies over the rules in your filter.
For example, if you create a filter for security purposes but enable dependency solving, Satellite can add packages that you might consider insecure.

To mitigate this problem, carefully test filtering rules to determine the required dependencies. If dependency solving includes unwanted packages, manually identify the core basic dependencies that the extra packages and errata need.

Example 7.2. Combining exclusion filters with dependency solving

For example, you can recreate Red Hat Enterprise Linux 8.3 by using content view filters and include selected errata from a later Red Hat Enterprise Linux 8 minor release. To achieve this, you create filters to exclude most of the errata after the Red Hat Enterprise Linux 8.3 release date, except a few that you need. Then, you enable dependency solving.

In this situation, dependency solving might include more packages than expected. As a result, the host diverges from being a Red Hat Enterprise Linux 8.3 machine.

If you do not need the extra errata and packages, do not configure content view filtering. Instead, enable and use the Red Hat Enterprise Linux 8.3 repository on the **Content** > **Red Hat Repositories** page in the Satellite web UI.

Example 7.3. Excluding packages sometimes makes dependency solving impossible for DNF

If you make a Red Hat Enterprise Linux 8.3 repository with a few excluded packages, **dnf upgrade** can sometimes fail.

Do not enable dependency solving to resolve the problem. Instead, investigate the error from **dnf** and adjust the filters to stop excluding the missing dependency.

Else, dependency solving might cause the repository to diverge from Red Hat Enterprise Linux 8.3.

7.12. ENABLING DEPENDENCY SOLVING FOR A CONTENT VIEW

Use this procedure to enable dependency solving for a content view.

Prerequisites

- Dependency solving is useful only in limited contexts. Before enabling it, ensure you read and understand Section 7.11, “Resolving package dependencies”

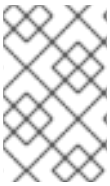
Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. From the list of content views, select the required content view.

3. On the **Details** tab, toggle **Solve dependencies**.

7.13. CONTENT FILTER EXAMPLES

Use any of the following examples with the procedure that follows to build custom content filters.



NOTE

Filters can significantly increase the time to publish a content view. For example, if a content view publish task completes in a few minutes without filters, it can take 30 minutes after adding an exclude or include errata filter.

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 you want to perform system updates on a regular 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 you only require the operating system packages and want to exclude 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.

Example 4

Filter a specific module stream in a content view.

Filter 1:

- **Inclusion Type:** Include
- **Content Type:** Module Stream
- **Filter:** Select only the specific module stream that you want for the content view, for example **ant**, and click **Add Module Stream**.

Filter 2:

- **Inclusion Type:** Exclude
- **Content Type:** Package
- **Filter:** Add a rule to filter any non-modular packages that you want to exclude from the content view. If you do not filter the packages, the content view filter includes all non-modular packages associated with the module stream **ant**. Add a rule to exclude all ***** packages, or specify the package names that you want to exclude.

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

7.14. CREATING A CONTENT FILTER FOR YUM CONTENT

You can filter content views containing Yum content to include or exclude specific packages, package groups, errata, or module streams. Filters are based on a combination of the *name*, *version*, and *architecture*.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

For examples of how to build a filter, see Section 7.13, “Content filter examples”.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Select a content view.
3. On the **Filters** tab, click **Create filter**.
4. Enter a name.
5. From the **Content type** list, select a content type.

6. From the **Inclusion Type** list, select either **Include filter** or **Exclude filter**.
7. Optional: In the **Description** field, enter a description for the filter.
8. Click **Create filter** to create your content filter.
9. Depending on what you enter for **Content Type**, add rules to create the filter that you want.
10. Select if you want the filter to **Apply to subset of repositories** or **Apply to all repositories**.
11. Click **Publish New Version** to publish the filtered repository.
12. Optional: In the **Description** field, enter a description of the changes.
13. Click **Create filter** to publish a new version of the content view.
    You can promote this content view across all environments.

CLI procedure

1. 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 "Example_Content_View" \
--description "My latest filter" \
--inclusion false \
--organization "My_Organization"
```

2. Add a rule to the filter:

```
# hammer content-view filter rule create \
--content-view "Example_Content_View" \
--content-view-filter "Errata Filter" \
--start-date "YYYY-MM-DD" \
--types enhancement,bugfix \
--date-type updated \
--organization "My_Organization"
```

3. Publish the content view:

```
# hammer content-view publish \
--name "Example_Content_View" \
--description "Adding errata filter" \
--organization "My_Organization"
```

4. Promote the view across all environments:

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

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

7.15. DELETING MULTIPLE CONTENT VIEW VERSIONS

You can delete multiple content view versions simultaneously.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Select the content view you want to delete versions of.
3. On the **Versions** tab, select the checkbox of the version or versions you want to delete.
4. Click the vertical ellipsis icon at the top of the list of content views.
5. Click **Delete** to open the deletion wizard that shows any affected environments.
6. If there are no affected environments, review the details and click **Delete**.
7. If there are any affected environments, reassign any hosts or activation keys before deletion.
8. Review the details of the actions.
9. Click **Delete**.

7.16. CLEARING THE SEARCH FILTER

If you search for specific content types by using keywords in the **Search** text box and the search returns no results, click **Clear search** to clear all the search queries and reset the **Search** text box.

If you use a filter to search for specific repositories in the **Type** text box and the search returns no results, click **Clear filters** to clear all active filters and reset the **Type** text box.

7.17. STANDARDIZING CONTENT VIEW EMPTY STATES

If there are no filters listed for a content view, click **Create filter**. A modal opens to show you the next steps to create a filter. Follow these steps to add a new filter to create new content types.

7.18. COMPARING CONTENT VIEW VERSIONS

Use this procedure to compare content view version functionality for Satellite.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.

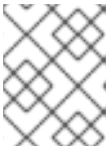
2. Select a content view whose versions you want to compare.
3. On the **Versions** tab, select the checkbox next to any two versions you want to compare.
4. Click **Compare**.
   The **Compare** screen has the pre-selected versions in the version dropdown menus and tabs for all content types found in either version. You can filter the results to show only the same, different, or all content types. You can compare different content view versions by selecting them from the dropdown menus.

7.19. DISTRIBUTING ARCHIVED CONTENT VIEW VERSIONS

The setting **Distribute archived content view versions**enables hosting of non-promoted content view version repositories in the Satellite content web application along with other repositories. This is useful while debugging to see what content is present in your content view versions.

Procedure

1. In the Satellite web UI, navigate to **Administer** > **Settings**.
2. Click the **Content** tab.
3. Set the **Distribute archived content view versions**parameter to **Yes**.
4. Click **Submit**.
   This enables the repositories of content view versions without lifecycle environments to be distributed at **satellite.example.com/pulp/content/*My_Organization*/content_views/*My_Content_View*/*My_Content_View_Version*/**.



NOTE

Older non-promoted content view versions are not distributed once the setting is enabled. Only new content view versions become distributed.

CHAPTER 8. SYNCHRONIZING CONTENT BETWEEN SATELLITE SERVERS

In a Satellite setup with multiple Satellite Servers, you can use Inter-Satellite Synchronization (ISS) to synchronize content from one upstream server to one or more downstream servers.

There are two possible ISS configurations of Satellite, depending on how you deployed your infrastructure. Configure your Satellite for ISS as appropriate for your scenario. For more information, see Inter-Satellite Synchronization scenarios in *Installing Satellite Server in a disconnected network environment*.

To change the Pulp export path, see Hammer content export fails with "Path '/the/path' is not an allowed export path" in the *Red Hat Knowledgebase*.

8.1. CONTENT SYNCHRONIZATION BY USING EXPORT AND IMPORT

There are multiple approaches for synchronizing content by using the export and import workflow:

- You employ the upstream Satellite Server as a content store, which means that you sync the whole Library rather than content view versions. This approach offers the simplest export/import workflow. In such case, you can manage the content view versions downstream. For more information, see Section 8.1.1, “Using an upstream Satellite Server as a content store” .
- You use the upstream Satellite Server to sync content view versions. This approach offers more control over what content is synced between Satellite Servers. For more information, see Section 8.1.2, “Using an upstream Satellite Server to synchronize content view versions” .
- You sync a single repository. This can be useful if you use the content-view syncing approach, but you want to sync an additional repository without adding it to an existing content view. For more information, see Section 8.1.3, “Synchronizing a single repository” .

NOTE

Synchronizing content by using export and import requires the same major, minor, and patch version of Satellite on both the downstream and upstream Satellite Servers.

When you are unable to match upstream and downstream Satellite versions, you can use:

- Syncable exports and imports.
- Inter-Satellite Synchronization (ISS) with your upstream Satellite connected to the Internet and your downstream Satellite connected to the upstream Satellite.

8.1.1. Using an upstream Satellite Server as a content store

In this scenario, you use the upstream Satellite Server as a content store for updates rather than to manage content. You use the downstream Satellite Server to manage content for all infrastructure behind the isolated network. You export the Library content from the upstream Satellite Server and import it into the downstream Satellite Server.

On the upstream Satellite Server

1. Ensure that repositories are using the **Immediate** download policy in one of the following ways:
   a. For existing repositories using **On Demand**, change their download policy on the repository

details page to **Immediate**.

b. For new repositories, ensure that the **Default Red Hat Repository download policy**setting is set to **Immediate** before enabling Red Hat repositories, and that the **Default download policy** is set to **Immediate** for custom repositories.

For more information, see Section 4.9, "Download policies overview" .

2. Enable the content that you want to synchronize. For more information, see Section 4.6, "Enabling Red Hat repositories".
   If you want to sync custom content, first create a custom product and then synchronize repositories.

3. Synchronize the enabled content:

   a. On the first export, perform a **complete** Library export so that all the synchronized content is exported. This generates content archives that you can later import into one or more downstream Satellite Servers. For more information on performing a complete Library export, see Section 8.3, "Exporting the Library environment".

   b. Export all future updates on the upstream Satellite Server incrementally. This generates leaner content archives that contain only a recent set of updates. For example, if you enable and synchronize a new repository, the next exported content archive contains content only from the newly enabled repository. For more information on performing an incremental Library export, see Section 8.6, "Exporting the Library environment incrementally" .

On the downstream Satellite Server

1. Bring the content exported from the upstream Satellite Server over to the hard disk.

2. Place it inside a directory under **/var/lib/pulp/imports**.

3. Import the content to an organization using the procedure outlined in Section 8.15, "Importing into the Library environment".
   You can then manage content using content views or lifecycle environments as you require.

8.1.2. Using an upstream Satellite Server to synchronize content view versions

In this scenario, you use the upstream Satellite Server not only as a content store, but also to synchronize content for all infrastructure behind the isolated network. You curate updates coming from the CDN into content views and lifecycle environments. Once you promote content to a designated lifecycle environment, you can export the content from the upstream Satellite Server and import it into the downstream Satellite Server.

On the upstream Satellite Server

1. Ensure that repositories are using the **Immediate** download policy in one of the following ways:

   a. For existing repositories using **On Demand**, change their download policy on the repository details page to **Immediate**.

   b. For new repositories, ensure that the **Default Red Hat Repository download policy**setting is set to **Immediate** before enabling Red Hat repositories, and that the **Default download policy** is set to **Immediate** for custom repositories.

For more information, see Section 4.9, "Download policies overview" .

2. Enable the content that you want to synchronize. For more information, see Section 4.6, "Enabling Red Hat repositories".
   If you want to sync custom content, first create a custom product and then synchronize repositories.

3. Synchronize the enabled content:

   a. For the first export, perform a **complete** version export on the content view version that you want to export. For more information see, Section 8.7, "Exporting a content view version". This generates content archives that you can import into one or more downstream Satellite Servers.

   b. Export all future updates in the connected Satellite Servers incrementally. This generates leaner content archives that contain changes only from the recent set of updates. For example, if your content view has a new repository, this exported content archive contains only the latest changes. For more information, see Section 8.9, "Exporting a content view version incrementally".

   c. When you have new content, republish the content views that include this content before exporting the increment. For more information, see Chapter 7, *Managing content views*. This creates a new content view version with the appropriate content to export.

On the downstream Satellite Server

1. Bring the content exported from the upstream Satellite Server over to the hard disk.

2. Place it inside a directory under **/var/lib/pulp/imports**.

3. Import the content to the organization that you want. For more information, see Section 8.17, "Importing a content view version". This will create a content view version from the exported content archives and then import content appropriately.

8.1.3. Synchronizing a single repository

In this scenario, you export and import a single repository.

On the upstream Satellite Server

1. Ensure that the repository is using the **Immediate** download policy in one of the following ways:

   a. For existing repositories using **On Demand**, change their download policy on the repository details page to **Immediate**.

   b. For new repositories, ensure that the **Default Red Hat Repository download policy**setting is set to **Immediate** before enabling Red Hat repositories, and that the **Default download policy** is set to **Immediate** for custom repositories.

   For more information, see Section 4.9, "Download policies overview" .

2. Enable the content that you want to synchronize. For more information, see Section 4.6, "Enabling Red Hat repositories".
   If you want to sync custom content, first create a custom product and then synchronize product repositories.

3. Synchronize the enabled content:

   a. On the first export, perform a **complete** repository export so that all the synchronized

content is exported. This generates content archives that you can later import into one or more downstream Satellite Servers. For more information on performing a complete repository export, see Section 8.10, "Exporting a repository".

b. Export all future updates on the upstream Satellite Server incrementally. This generates leaner content archives that contain only a recent set of updates. For more information on performing an incremental repository export, see Section 8.12, "Exporting a repository incrementally".

On the downstream Satellite Server

1. Bring the content exported from the upstream Satellite Server over to the hard disk.
2. Place it inside a directory under **/var/lib/pulp/imports**.
3. Import the content to an organization. See Section 8.19, "Importing a repository".
   You can then manage content using content views or lifecycle environments as you require.

8.2. SYNCHRONIZING A CUSTOM REPOSITORY

When using Inter-Satellite Synchronization Network Sync, Red Hat repositories are configured automatically, but custom repositories are not. Use this procedure to synchronize content from a custom repository on a connected Satellite Server to a disconnected Satellite Server through Inter-Satellite Synchronization (ISS) Network Sync.

Follow the procedure for the connected Satellite Server before completing the procedure for the disconnected Satellite Server.

Connected Satellite Server

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Click on the custom product.
3. Click on the custom repository.
4. Copy the **Published At:** URL.
5. Continue with the procedure on disconnected Satellite Server.

Disconnected Satellite Server

1. Download the **katello-server-ca.crt** file from the connected Satellite Server:

```
# curl http://satellite.example.com/pub/katello-server-ca.crt
```

2. Create an SSL Content Credential with the contents of **katello-server-ca.crt**. For more information on creating an SSL Content Credential, see Section 4.3, "Importing custom SSL certificates".
3. In the Satellite web UI, navigate to **Content** > **Products**.
4. Create your custom product with the following:
   - **Upstream URL**: Paste the link that you copied earlier.

- **SSL CA Cert**: Select the SSL certificate that was transferred from your connected Satellite Server.

For more information on creating a custom product, see Section 4.4, "Creating a custom product".

After completing these steps, the custom repository is properly configured on the disconnected Satellite Server.

8.3. EXPORTING THE LIBRARY ENVIRONMENT

You can export contents of all Yum repositories in the Library environment of an organization to an archive file from Satellite Server and use this archive file to create the same repositories in another Satellite Server or in another Satellite Server organization. The exported archive file contains the following data:

- A JSON file containing content view version metadata.
- An archive file containing all the repositories from the Library environment of the organization.

Satellite Server exports only RPM, Kickstart files, and Docker content included in the Library environment.

Prerequisites

- Ensure that the export directory has free storage space to accommodate the export.
- Ensure that the **/var/lib/pulp/exports** directory has free storage space equivalent to the size of the repositories being exported for temporary files created during the export process.
- Ensure that you set download policy to **Immediate** for all repositories within the Library lifecycle environment you export. For more information, see Section 4.9, "Download policies overview" .
- Ensure that you synchronize products that you export to the required date.

Procedure

1. Use the organization name or ID to export.

```
# hammer content-export complete library --organization="My_Organization"
```

2. Verify that the archive containing the exported version of a content view is located in the export directory:

```
# ls -lh /var/lib/pulp/exports/My_Organization/Export-Library/1.0/2021-03-02T03-35-24-00-00
total 68M
-rw-r--r--. 1 pulp pulp 68M Mar  2 03:35 export-1e25417c-6d09-49d4-b9a5-23df4db3d52a-20210302_0335.tar.gz
-rw-r--r--. 1 pulp pulp 333 Mar  2 03:35 export-1e25417c-6d09-49d4-b9a5-23df4db3d52a-20210302_0335-toc.json
-rw-r--r--. 1 pulp pulp 443 Mar  2 03:35 metadata.json
```

You need all three files, the **tar.gz**, the **toc.json**, and the **metadata.json** file to be able to import.

3. A new content view **Export-Library** is created in the organization. This content view contains all the repositories belonging to this organization. A new version of this content view is published and exported automatically.

Export with chunking

In many cases the exported archive content may be several gigabytes in size. If you want to split it into smaller sizes or chunks. You can use the **--chunk-size-gb** flag directly in the export command to handle this. In the following example, you can see how to specify **--chunk-size-gb=2** to split the archives in **2 GB** chunks.

```
# hammer content-export complete library \
--chunk-size-gb=2 \
--organization="My_Organization"

Generated /var/lib/pulp/exports/My_Organization/Export-Library/2.0/2021-03-02T04-01-25-00-00/metadata.json

# ls -lh /var/lib/pulp/exports/My_Organization/Export-Library/2.0/2021-03-02T04-01-25-00-00/
```

8.4. EXPORTING THE LIBRARY ENVIRONMENT IN A SYNCABLE FORMAT

You can export contents of all yum repositories, Kickstart repositories and file repositories in the Library environment of an organization to a syncable format that you can use to create your custom CDN and synchronize the content from the custom CDN over HTTP/HTTPS.

You can then serve the generated content on a local web server and synchronize it on the importing Satellite Server or in another Satellite Server organization.

You can use the generated content to create the same repository in another Satellite Server or in another Satellite Server organization by using content import. On import of the exported archive, a regular content view is created or updated on your importing Satellite Server. For more information, see Section 8.17, “Importing a content view version” .

You can export the following content in the syncable format from Satellite Server:

- Yum repositories
- Kickstart repositories
- File repositories

You cannot export Ansible, Deb, or Docker content.

The export contains directories with the packages, **listing** files, and metadata of the repository in Yum format that can be used to synchronize in the importing Satellite Server.

Prerequisites

- Ensure that you set the download policy to **Immediate** for all repositories within the Library lifecycle environment you export. For more information, see Section 4.9, “Download policies overview”.
- Ensure that you synchronize products you export to the required date.

- Ensure that the user exporting the content has the **Content Exporter** role.

Procedure

1. Use the organization name or ID to export:

```
# hammer content-export complete library \
--organization="My_Organization" \
--format=syncable
```

2. Optional: Verify that the exported content is located in the export directory:

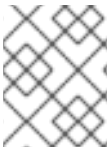
```
# du -sh /var/lib/pulp/exports/My_Organization/Export-My_Repository/1.0/2021-03-02T03-35-24-00-00
```

8.5. IMPORTING SYNCABLE EXPORTS

Procedure

- Use the organization name or ID to import syncable exports:

```
# hammer content-import library
--organization="My_Organization"
--path="My_Path_To_Syncable_Export"
```



NOTE

Syncable exports must be located in one of your **ALLOWED_IMPORT_PATHS** as specified in **/etc/pulp/settings.py**. By default, this includes **/var/lib/pulp/imports**.

8.6. EXPORTING THE LIBRARY ENVIRONMENT INCREMENTALLY

Exporting Library content can be a very expensive operation in terms of system resources. Organizations that have multiple Red Hat Enterprise Linux trees can occupy several gigabytes of space on Satellite Server.

In such cases, you can create an incremental export which contains only pieces of content that have changed since the last export. Incremental exports typically result in smaller archive files than the full exports.

The example below shows incremental export of all repositories in the organization's Library.

Procedure

1. Create an incremental export:

```
# hammer content-export incremental library \
--organization="My_Organization"
```

If you want to create a syncable export, add **--format=syncable**. By default, Satellite creates an importable export.

Next steps

- Optional: View the exported data:

```
# find /var/lib/pulp/exports/My_Organization/Export-Library/
```

8.7. EXPORTING A CONTENT VIEW VERSION

You can export a version of a content view to an archive file from Satellite Server and use this archive file to create the same content view version on another Satellite Server or on another Satellite Server organization. Satellite exports composite content views as normal content views. The composite nature is not retained. On importing the exported archive, a regular content view is created or updated on your downstream Satellite Server. The exported archive file contains the following data:

- A JSON file containing content view version metadata
- An archive file containing all the repositories included into the content view version

You can only export Yum repositories, Kickstart files, and Docker content added to a version of a content view. Satellite does not export the following content:

- Content view definitions and metadata, such as package filters.

Prerequisites

To export a content view, ensure that Satellite Server where you want to export meets the following conditions:

- Ensure that the export directory has free storage space to accommodate the export.
- Ensure that the **/var/lib/pulp/exports** directory has free storage space equivalent to the size of the repositories being exported for temporary files created during the export process.
- Ensure that you set download policy to **Immediate** for all repositories within the content view you export. For more information, see Section 4.9, “Download policies overview” .
- Ensure that you synchronize products that you export to the required date.
- Ensure that the user exporting the content has the **Content Exporter** role.

To export a content view version

1. List versions of the content view that are available for export:

```
# hammer content-view version list \
--content-view="My_Content_View" \
--organization="My_Organization"

---|----------|---------|-------------|----------------------
ID | NAME     | VERSION | DESCRIPTION | LIFECYCLE ENVIRONMENTS
---|----------|---------|-------------|----------------------
5  | view 3.0 | 3.0     |             | Library
4  | view 2.0 | 2.0     |             |
3  | view 1.0 | 1.0     |             |
---|----------|---------|-------------|----------------------
```

Export a content view version

1. Get the version number of desired version. The following example targets version **1.0** for export.

```
# hammer content-export complete version \
--content-view="Content_View_Name" \
--version=1.0 \
--organization="My_Organization"
```

2. Verify that the archive containing the exported version of a content view is located in the export directory:

```
# ls -lh /var/lib/pulp/exports/My_Organization/Content_View_Name/1.0/2021-02-25T18-59-26-00-00/
```

You require all three files, for example, the **tar.gz** archive file, the **toc.json** and **metadata.json** to import the content successfully.

Export with chunking

In many cases, the exported archive content can be several gigabytes in size. You might want to split it smaller sizes or chunks. You can use the **--chunk-size-gb** option with in the **hammer content-export** command to handle this. The following example uses the **--chunk-size-gb=2** to split the archives into **2 GB** chunks.

```
# hammer content-export complete version \
--chunk-size-gb=2 \
--content-view="Content_View_Name" \
--organization="My_Organization" \
--version=1.0
# ls -lh /var/lib/pulp/exports/My_Organization/view/1.0/2021-02-25T21-15-22-00-00/
```

8.8. EXPORTING A CONTENT VIEW VERSION IN A SYNCABLE FORMAT

You can export a version of a content view to a syncable format that you can use to create your custom CDN. After you have exported the content view, you can do either of the following:

- Synchronize the content from your custom CDN over HTTP/HTTPS.
- Import the content using **hammer content-import**. Note that this requires both the Export and Import servers to run Satellite 6.16.

You can then serve the generated content using a local web server on the importing Satellite Server or in another Satellite Server organization.

You cannot directly import Syncable Format exports. Instead, on the importing Satellite Server you must:

- Copy the generated content to an HTTP/HTTPS web server that is accessible to importing Satellite Server.
- Update your CDN configuration to **Custom CDN**.
- Set the CDN URL to point to the web server.

- Optional: Set an SSL/TLS CA Credential if the web server requires it.
- Enable the repository.
- Synchronize the repository.

You can export the following content in a syncable format from Satellite Server:

- Yum repositories
- Kickstart repositories
- File repositories

You cannot export Ansible, Deb, or Docker content.

The export contains directories with the packages, **listing** files, and metadata of the repository in Yum format that can be used to synchronize in the importing Satellite Server.

Prerequisites

- Ensure that you set the download policy to **Immediate** for all repositories within the content view you export. For more information, see Section 4.9, “Download policies overview” .
- Ensure that you synchronize products you export to the required date.
- Ensure that the user exporting the content has the **Content Exporter** role.

To export a content view version

- List versions of the content view that are available for export:

```
# hammer content-view version list \
--content-view="My_Content_View" \
--organization="My_Organization"
```

Procedure

1. Get the version number of desired version. The following example targets version **1.0** for export:

```
# hammer content-export complete version \
--content-view="Content_View_Name" \
--version=1.0 \
--organization="My_Organization" \
--format=syncable
```

2. Optional: Verify that the exported content is located in the export directory:

```
# ls -lh /var/lib/pulp/exports/My_Organization/My_Content_View_Name/1.0/2021-02-25T18-59-26-00-00/
```

8.9. EXPORTING A CONTENT VIEW VERSION INCREMENTALLY

Exporting complete content view versions can be a very expensive operation in terms of system resources. Content view versions that have multiple Red Hat Enterprise Linux trees can occupy several gigabytes of space on Satellite Server.

In such cases, you can create an incremental export which contains only pieces of content that have changed since the last export. Incremental exports typically result in smaller archive files than the full exports.

Procedure

1. Create an incremental export:

```
# hammer content-export incremental version \
--content-view="My_Content_View" \
--organization="My_Organization" \
--version="My_Content_View_Version"
```

If you want to create a syncable export, add **--format=syncable**. By default, Satellite creates an importable export.

Next steps

- Optional: View the exported content view:

```
# find
/var/lib/pulp/exports/My_Organization/My_Exported_Content_View/My_Content_View_Version/
```

- You can import your exported content view version into Satellite Server. For more information, see Section 8.17, “Importing a content view version” .

8.10. EXPORTING A REPOSITORY

You can export the content of a repository in the Library environment of an organization from Satellite Server. You can use this archive file to create the same repository in another Satellite Server or in another Satellite Server organization.

You can export the following content from Satellite Server:

- Ansible repositories
- Kickstart repositories
- Yum repositories
- File repositories
- Docker content

The export contains the following data:

- Two JSON files containing repository metadata.
- One or more archive files containing the contents of the repository from the Library environment of the organization.

You need all the files, **tar.gz**, **toc.json** and **metadata.json**, to be able to import.

Prerequisites

- Ensure that the export directory has enough free storage space to accommodate the export.
- Ensure that the **/var/lib/pulp/exports** directory has enough free storage space equivalent to the size of all repositories that you want to export.
- Ensure that you set download policy to **Immediate** for the repository within the Library lifecycle environment you export. For more information, see Section 4.9, "Download policies overview" .
- Ensure that you synchronize products that you export to the required date.

Procedure

1. Export a repository:

```
# hammer content-export complete repository \
--name="My_Repository" \
--product="My_Product" \
--organization="My_Organization"
```

NOTE

The size of the exported archive depends on the number and size of the packages within the repository. If you want to split the exported archive into chunks, export your repository using the **--chunk-size-gb** argument to limit the size by an integer value in gigabytes, for example **---chunk-size-gb=*2***.

2. Optional: Verify that the exported archive is located in the export directory:

```
# ls -lh /var/lib/pulp/exports/My_Organization/Export-My_Repository/1.0/2022-09-02T03-35-24-00-00/
```

8.11. EXPORTING A REPOSITORY IN A SYNCABLE FORMAT

You can export the content of a repository in the Library environment of an organization to a syncable format that you can use to create your custom CDN and synchronize the content from the custom CDN over HTTP/HTTPS.

You can then serve the generated content using a local web server on the importing Satellite Server or in another Satellite Server organization.

You cannot directly import Syncable Format exports. Instead, on the importing Satellite Server you must:

- Copy the generated content to an HTTP/HTTPS web server that is accessible to importing Satellite Server.
- Update your CDN configuration to **Custom CDN**.
- Set the CDN URL to point to the web server.

- Optional: Set an SSL/TLS CA Credential if the web server requires it.
- Enable the repository.
- Synchronize the repository.

You can export the following content in a syncable format from Satellite Server:

- Yum repositories
- Kickstart repositories
- File repositories

You cannot export Ansible, Deb, or Docker content.

The export contains directories with the packages, **listing** files, and metadata of the repository in Yum format that can be used to synchronize in the importing Satellite Server.

Prerequisites

- Ensure that you set the download policy to **Immediate** for the repository within the Library lifecycle environment you export. For more information, see Section 4.9, "Download policies overview".

Procedure

1. Export a repository using the repository name or ID:

```
# hammer content-export complete repository \
--organization="My_Organization" \
--product="My_Product" \
--name="My_Repository" \
--format=syncable
```

2. Optional: Verify that the exported content is located in the export directory:

```
# du -sh /var/lib/pulp/exports/My_Organization/Export-My_Repository/1.0/2021-03-02T03-35-24-00-00
```

8.12. EXPORTING A REPOSITORY INCREMENTALLY

Exporting a repository can be a very expensive operation in terms of system resources. A typical Red Hat Enterprise Linux tree may occupy several gigabytes of space on Satellite Server.

In such cases, you can use **Incremental Export** to export only pieces of content that changed since the previous export. Incremental exports typically result in smaller archive files than the full exports.

The example below shows incremental export of a repository in the Library lifecycle environment.

Procedure

1. Create an incremental export:

```
# hammer content-export incremental repository \
```

```
--name="My_Repository" \
--organization="My_Organization" \
--product="My_Product"
```

2. Optional: View the exported data:

```
# ls -lh /var/lib/pulp/exports/My_Organization/Export-My_Repository/3.0/2021-03-02T03-35-24-00-00/
total 172K
-rw-r--r--. 1 pulp pulp  20M Mar  2 04:22 export-436882d8-de5a-48e9-a30a-17169318f908-20210302_0422.tar.gz
-rw-r--r--. 1 pulp pulp  333 Mar  2 04:22 export-436882d8-de5a-48e9-a30a-17169318f908-20210302_0422-toc.json
-rw-r--r--. 1 root root  492 Mar  2 04:22 metadata.json
```

8.13. EXPORTING A REPOSITORY INCREMENTALLY IN A SYNCABLE FORMAT

Exporting a repository can be a very expensive operation in terms of system resources. A typical Red Hat Enterprise Linux tree may occupy several gigabytes of space on Satellite Server.

In such cases, you can use **Incremental Export** to export only pieces of content that changed since the previous export. Incremental exports typically result in smaller archive files than full exports.

The procedure below shows an incremental export of a repository in the Library lifecycle environment.

Procedure

1. Create an incremental export:

```
# hammer content-export incremental repository \
--format=syncable \
--name="My_Repository" \
--organization="My_Organization" \
--product="My_Product"
```

2. Optional: View the exported data:

```
# find /var/lib/pulp/exports/Default_Organization/My_Product/2.0/2023-03-09T10-55-48-05-00/ -name "*.rpm"
```

8.14. KEEPING TRACK OF YOUR EXPORTS

Satellite keeps records of all exports. Each time you export content on the upstream Satellite Server, the export is recorded and maintained for future querying. You can use the records to organize and manage your exports, which is useful especially when exporting incrementally.

When exporting content from the upstream Satellite Server for several downstream Satellite Servers, you can also keep track of content exported for specific servers. This helps you track which content was exported and to where.

Use the **--destination-server** argument during export to indicate the target server. This option is available for all **content-export** operations.

Tracking destinations of Library exports

- Specify the destination server when exporting the Library:

```
# hammer content-export complete library \
--destination-server=My_Downstream_Server_1 \
--organization="My_Organization" \
--version=1.0
```

Tracking destinations of content view exports

- Specify the destination server when exporting a content view version:

```
# hammer content-export complete version \
--content-view="Content_View_Name" \
--destination-server=My_Downstream_Server_1 \
--organization="My_Organization" \
--version=1.0
```

Querying export records

- List content exports using the following command:

```
# hammer content-export list --organization="My_Organization"
```

8.15. IMPORTING INTO THE LIBRARY ENVIRONMENT

You can import exported Library content into the Library lifecycle environment of an organization on another Satellite Server. For more information about exporting contents from the Library environment, see Section 8.3, “Exporting the Library environment”.

Prerequisites

- The exported files must be in a directory under **/var/lib/pulp/imports**.
- If there are any Red Hat repositories in the exported content, the importing organization’s manifest must contain subscriptions for the products contained within the export.
- The user importing the content must have the *Content Importer* Role.

Procedure

1. Copy the exported files to a subdirectory of **/var/lib/pulp/imports** on Satellite Server where you want to import.
2. Set the ownership of the import directory and its contents to **pulp:pulp**.

```
# chown -R pulp:pulp /var/lib/pulp/imports/2021-03-02T03-35-24-00-00
```

3. Verify that the ownership is set correctly:

```
# ls -lh /var/lib/pulp/imports/2021-03-02T03-35-24-00-00
total 68M
```

```
-rw-r--r--. 1 pulp pulp 68M Mar  2 04:29 export-1e25417c-6d09-49d4-b9a5-23df4db3d52a-20210302_0335.tar.gz
-rw-r--r--. 1 pulp pulp 333 Mar  2 04:29 export-1e25417c-6d09-49d4-b9a5-23df4db3d52a-20210302_0335-toc.json
-rw-r--r--. 1 pulp pulp 443 Mar  2 04:29 metadata.json
```

4. Identify the Organization that you wish to import into.

5. To import the Library content to Satellite Server, enter the following command:

```
# hammer content-import library \
--organization="My_Organization" \
--path=/var/lib/pulp/imports/2021-03-02T03-35-24-00-00
```

   Note you must enter the full path **/var/lib/pulp/imports/*My_Exported_Library_Dir***. Relative paths do not work.

6. To verify that you imported the Library content, check the contents of the product and repositories. A new content view called **Import-Library** is created in the target organization. This content view is used to facilitate the Library content import.
   By default, this content view is not shown in the Satellite web UI. **Import-Library** is not meant to be assigned directly to hosts. Instead, assign your hosts to **Default Organization View** or another content view as you would normally.

7. The importing Satellite Server extracts the **/var/lib/pulp/imports** directory to **/var/lib/pulp/**. You can empty the **/var/lib/pulp/imports** directory after a successful import.

8.16. IMPORTING INTO THE LIBRARY ENVIRONMENT FROM A WEB SERVER

You can import exported Library content directly from a web server into the Library lifecycle environment of an organization on another Satellite Server. For more information about exporting contents from the Library environment, see Section 8.3, “Exporting the Library environment”.

Prerequisites

- The exported files must be in a syncable format.
- The exported files must be accessible through HTTP/HTTPS.
- If there are any Red Hat repositories in the exported content, the importing organization’s manifest must contain subscriptions for the products contained within the export.
- The user importing the content view version must have the *Content Importer* role.

Procedure

1. Identify the Organization that you wish to import into.

2. To import the Library content to Satellite Server, enter the following command:

```
# hammer content-import library \
--organization="My_Organization" \
--path=http://server.example.com/pub/exports/2021-02-25T21-15-22-00-00/
```

A new content view called **Import-Library** is created in the target organization. This content view is used to facilitate the Library content import.

By default, this content view is not shown in the Satellite web UI. **Import-Library** is not meant to be assigned directly to hosts. Instead, assign your hosts to **Default Organization View** or another content view.

8.17. IMPORTING A CONTENT VIEW VERSION

You can import an exported content view version to create a version with the same content in an organization on another Satellite Server. For more information about exporting a content view version, see Section 8.7, “Exporting a content view version” .

When you import a content view version, it has the same major and minor version numbers and contains the same repositories with the same packages and errata. Custom repositories, products and content views are automatically created if they do not exist in the importing organization.

Prerequisites

- The exported files must be in a directory under **/var/lib/pulp/imports**.
- If there are any Red Hat repositories in the exported content, the importing organization’s manifest must contain subscriptions for the products contained within the export.
- The user importing the content view version must have the *Content Importer* Role.

Procedure

1. Copy the exported files to a subdirectory of **/var/lib/pulp/imports** on Satellite Server where you want to import.
2. Set the ownership of the import directory and its contents to **pulp:pulp**.

```
# chown -R pulp:pulp /var/lib/pulp/imports/2021-02-25T21-15-22-00-00/
```

3. Verify that the ownership is set correctly:

```
# ls -lh /var/lib/pulp/imports/2021-02-25T21-15-22-00-00/
```

4. To import the content view version to Satellite Server, enter the following command:

```
# hammer content-import version \
--organization=My_Organization \
--path=/var/lib/pulp/imports/2021-02-25T21-15-22-00-00/
```

Note that you must enter the full path **/var/lib/pulp/imports/*My_Exported_Version_Dir***. Relative paths do not work.

5. To verify that you imported the content view version successfully, list content view versions for your organization:

```
# hammer content-view version list \
--organization-id=My_Organization_ID
```

6. The importing Satellite Server extracts the **/var/lib/pulp/imports** directory to **/var/lib/pulp/**. You can empty the **/var/lib/pulp/imports** directory after a successful import.

8.18. IMPORTING A CONTENT VIEW VERSION FROM A WEB SERVER

You can import an exported content view version directly from a web server to create a version with the same content in an organization on another Satellite Server. For more information about exporting a content view version, see Section 8.7, “Exporting a content view version” .

When you import a content view version, it has the same major and minor version numbers and contains the same repositories with the same packages and errata. Custom repositories, products, and content views are automatically created if they do not exist in the importing organization.

Prerequisites

- The exported files must be in a syncable format.
- The exported files must be accessible through HTTP/HTTPS.
- If there are any Red Hat repositories in the exported content, the importing organization’s manifest must contain subscriptions for the products contained within the export.
- The user importing the content view version must have the *Content Importer* role.

Procedure

- Import the content view version into Satellite Server:

```
# hammer content-import version \
--organization=My_Organization \
--path=http://server.example.com/pub/exports/2021-02-25T21-15-22-00-00/
```

8.19. IMPORTING A REPOSITORY

You can import an exported repository into an organization on another Satellite Server. For more information about exporting content of a repository, see Section 8.10, “Exporting a repository”.

Prerequisites

- The export files must be in a directory under **/var/lib/pulp/imports**.
- If the export contains any Red Hat repositories, the manifest of the importing organization must contain subscriptions for the products contained within the export.
- The user importing the content must have the *Content Importer* Role.

Procedure

1. Copy the exported files to a subdirectory of **/var/lib/pulp/imports** on Satellite Server where you want to import.
2. Set the ownership of the import directory and its contents to **pulp:pulp**.

```
# chown -R pulp:pulp /var/lib/pulp/imports/2021-03-02T03-35-24-00-00
```

3. Verify that the ownership is set correctly:

```
# ls -lh /var/lib/pulp/imports/2021-03-02T03-35-24-00-00
total 68M
-rw-r--r--. 1 pulp pulp 68M Mar  2 04:29 export-1e25417c-6d09-49d4-b9a5-23df4db3d52a-20210302_0335.tar.gz
-rw-r--r--. 1 pulp pulp 333 Mar  2 04:29 export-1e25417c-6d09-49d4-b9a5-23df4db3d52a-20210302_0335-toc.json
-rw-r--r--. 1 pulp pulp 443 Mar  2 04:29 metadata.json
```

4. Identify the Organization that you wish to import into.

5. To import the repository content to Satellite Server, enter the following command:

```
# hammer content-import repository \
--organization="My_Organization" \
--path=/var/lib/pulp/imports/2021-03-02T03-35-24-00-00
```

   Note that you must enter the full path **/var/lib/pulp/imports/*My_Exported_Repo_Dir***. Relative paths do not work.

6. To verify that you imported the repository, check the contents of the product and repository.

7. The importing Satellite Server extracts the **/var/lib/pulp/imports** directory to **/var/lib/pulp/**. You can empty the **/var/lib/pulp/imports** directory after a successful import.

8.20. IMPORTING A REPOSITORY FROM A WEB SERVER

You can import an exported repository directly from a web server into an organization on another Satellite Server. For more information about exporting the content of a repository, see Section 8.10, “Exporting a repository”.

Prerequisites

- The exported files must be in a syncable format.
- The exported files must be accessible through HTTP/HTTPS.
- If the export contains any Red Hat repositories, the manifest of the importing organization must contain subscriptions for the products contained within the export.
- The user importing the content view version must have the *Content Importer* Role.

Procedure

1. Select the organization into which you want to import.

2. To import the repository to Satellite Server, enter the following command:

```
# hammer content-import repository \
--organization="My_Organization" \
--path=http://server.example.com/pub/exports/2021-02-25T21-15-22-00-00/
```

8.21. EXPORTING AND IMPORTING CONTENT USING HAMMER CLI CHEAT SHEET

Table 8.1. Export

Intent	Command
Fully export an Organization's Library	**hammer content-export complete library --organization="*My_Organization*"**
Incrementally export an Organization's Library (assuming you have exported something previously)	**hammer content-export incremental library --organization="*My_Organization*"**
Fully export a content view version	**hammer content-export complete version --content-view="*My_Content_View*" --version=1.0 --organization="*My_Organization*"**
Export a content view version promoted to the Dev Environment	**hammer content-export complete version --content-view="*My_Content_View*" --organization="*My_Organization*" --lifecycle-environment="Dev"**
Export a content view in smaller chunks (2-GB slabs)	**hammer content-export complete version --content-view="*My_Content_View*" --version=1.0 --organization="*My_Organization*" --chunk-size-gb=2**
Incrementally export a content view version (assuming you have exported something previously)	**hammer content-export incremental version --content-view="*My_Content_View*" --version=2.0 --organization="*My_Organization*"**
Fully export a Repository	**hammer content-export complete repository --product="*My_Product*" --name="*My_Repository*" --organization="*My_Organization*"**
Incrementally export a Repository (assuming you have exported something previously)	**hammer content-export incremental repository --product="*My_Product*" --name="*My_Repository*" --organization="*My_Organization*"**
List exports	**hammer content-export list --content-view="*My_Content_View*" --organization="*My_Organization*"**

Table 8.2. Import

Intent	Command
Import into an Organization's Library	**hammer content-import library --organization="*My_Organization*" --path="/var/lib/pulp/imports/*My_Exported_Library_Dir*"**
Import to a content view version	**hammer content-import version --organization="*My_Organization*" --path="/var/lib/pulp/imports/*My_Exported_Version_Dir*"**
Import a Repository	**hammer content-import repository --organization="*My_Organization*" --path="/var/lib/pulp/imports/*My_Exported_Repo_Dir*"**

CHAPTER 9. MANAGING ACTIVATION KEYS

Activation keys provide a method to automate system registration. You can create multiple keys and associate them with different environments and content views. For example, you might create a basic activation key that enables certain Red Hat repositories and associate it with the appropriate content view.

You can use activation keys during content host registration to improve the speed, simplicity and consistency of the process. Note that activation keys are used only when hosts are registered. If changes are made to an activation key, it is applicable only to hosts that are registered with the amended activation key in the future. The changes are not made to existing hosts.

Activation keys can define the following properties for content hosts:

- Available products and repositories
- A lifecycle environment and a content view
- Host collection membership
- System purpose

Content view conflicts between host creation and registration

When you provision a host, Satellite uses provisioning templates and other content from the content view that you set in the host group or host settings. When the host is registered, the content view from the activation key overwrites the original content view from the host group or host settings. Then Satellite uses the content view from the activation key for every future task, for example, rebuilding a host.

When you rebuild a host, ensure that you set the content view that you want to use in the activation key and not in the host group or host settings.

Using multiple activation keys with a content host

A 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. You can specify the order of precedence by setting a host group parameter as follows:

```
$ hammer hostgroup set-parameter \
--hostgroup "My_Host_Group" \
--name "My_Activation_Key" \
--value "name_of_first_key", "name_of_second_key", ...
```

9.1. BEST PRACTICES FOR ACTIVATION KEYS

- Create an activation key for each use case. This structures, modularizes, and simplifies content management on hosts.
- Use a naming convention for activation keys to indicate the content and lifecycle environment, for example, **red-hat-enterprise-linux-webserver**.
- Automate activation key management by using a Hammer script or an Ansible Playbook.

9.2. CREATING AN ACTIVATION KEY

Create an activation key to assign various attributes to hosts during registration.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Activation Keys** and click **Create Activation Key**.
2. In the **Name** field, enter the name of the activation key.
3. If you want to set a limit, clear the **Unlimited hosts** checkbox, and in the **Limit** field, enter the maximum number of systems you can register with the activation key. If you want unlimited hosts to register with the activation key, ensure the **Unlimited Hosts** checkbox is selected.
4. Optional: In the **Description** field, enter a description for the activation key.
5. From the **Environment** list, select the environment to use.
6. From the **Content View** list, select a content view to use.
7. In the **Repository Sets** tab, override repositories to Enabled or Disabled as desired. For more information, see Section 9.5, “Enabling and disabling repositories on activation key” .
8. Click **Save**.
9. Optional: In the **System Purpose** section, you can configure the activation key to set system purpose attributes on hosts during registration. This helps determine which repositories are available on the host. It also helps with reporting in the Subscriptions service of the Red Hat Hybrid Cloud Console.

CLI procedure

1. Create the activation key:

```
# hammer activation-key create \
--name "My_Activation_Key" \
--unlimited-hosts \
--description "Example Stack in the Development Environment" \
--lifecycle-environment "Development" \
--content-view "Stack" \
--organization "My_Organization"
```

2. Optional: Enter the following command to configure the activation key with system purpose attributes to set on hosts during registration. This helps determine which repositories are available on the host. It also helps with reporting in the Subscriptions service of the Red Hat Hybrid Cloud Console.

```
# hammer activation-key update \
--organization "My_Organization" \
--name "My_Activation_Key" \
--service-level "Standard" \
--purpose-usage "Development/Test" \
--purpose-role "Red Hat Enterprise Linux Server" \
--purpose-addons "addons"
```

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

```
# hammer activation-key product-content \
--content-access-mode-all true \
--name "My_Activation_Key" \
--organization "My_Organization"
```

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

```
# hammer activation-key content-override \
--name "My_Activation_Key" \
--content-label rhel-7-server-satellite-client-6-rpms \
--value 1 \
--organization "My_Organization"
```

9.3. USING ACTIVATION KEYS FOR HOST REGISTRATION

You can use activation keys to complete the following tasks:

- Registering new hosts during provisioning through Red Hat Satellite. The Kickstart provisioning templates in Red Hat Satellite contain commands to register the host using an activation key that is defined when creating a host.
- Registering existing Red Hat Enterprise Linux hosts.

You can register hosts with Satellite using the host registration feature in the Satellite web UI, Hammer CLI, or the Satellite API. For more information, see Registering hosts and setting up host integration in *Managing hosts*.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **Register Host**.
2. From the **Activation Keys** list, select the activation keys to assign to your host.
3. Click **Generate** to create the registration command.
4. Click on the *files* icon to copy the command to your clipboard.
5. Connect to your host using SSH and run the registration command.
6. Check the **/etc/yum.repos.d/redhat.repo** file and ensure that the appropriate repositories have been enabled.

CLI procedure

1. Generate the host registration command using the Hammer CLI:

```
# hammer host-registration generate-command \
--activation-keys "My_Activation_Key"
```

If your hosts do not trust the SSL certificate of Satellite Server, you can disable SSL validation by adding the **--insecure** flag to the registration command.

–

```
# hammer host-registration generate-command \
--activation-keys "My_Activation_Key" \
--insecure true
```

2. Connect to your host using SSH and run the registration command.
3. Check the **/etc/yum.repos.d/redhat.repo** file and ensure that the appropriate repositories have been enabled.

API procedure

1. Generate the host registration command using the Satellite API:

```
# curl -X POST https://satellite.example.com/api/registration_commands \
--user "My_User_Name" \
-H 'Content-Type: application/json' \
-d '{ "registration_command": { "activation_keys": ["My_Activation_Key_1,
My_Activation_Key_2"] }}'
```

If your hosts do not trust the SSL certificate of Satellite Server, you can disable SSL validation by adding the **--insecure** flag to the registration command.

```
# curl -X POST https://satellite.example.com/api/registration_commands \
--user "My_User_Name" \
-H 'Content-Type: application/json' \
-d '{ "registration_command": { "activation_keys": ["My_Activation_Key_1,
My_Activation_Key_2"], "insecure": true }}'
```

Use an activation key to simplify specifying the environments. For more information, see Managing Activation Keys in *Managing content*.

To enter a password as a command line argument, use **username:password** syntax. Keep in mind this can save the password in the shell history. Alternatively, you can use a temporary personal access token instead of a password. To generate a token in the Satellite web UI, navigate to **My Account** > **Personal Access Tokens**.

2. Connect to your host using SSH and run the registration command.
3. Check the **/etc/yum.repos.d/redhat.repo** file and ensure that the appropriate repositories have been enabled.

Multiple activation keys

You can use multiple activation keys when registering a content host. For example, you can use one activation key to enable specific repositories and another to assign a content view and lifecycle environment.

Settings conflicts

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: **Host Collections**.

- Settings that influence the behavior of the key itself and not the host configuration: **Content Host Limit**.

9.4. SETTING THE SERVICE LEVEL

You can configure an activation key to define a default service level for the new host created with the activation key.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **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**.

CLI procedure

- Set the service level to Premium on your activation key:

```
# hammer activation-key update \
--name "My_Activation_Key" \
--organization "My_Organization" \
--service-level premium
```

9.5. ENABLING AND DISABLING REPOSITORIES ON ACTIVATION KEY

You can enable or disable repositories on an activation key in the Satellite web UI.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Activation Keys**.
2. Select an activation key.
3. Select the **Repository Sets** tab.
4. Optional: Clear the **Limit to Environment** checkbox to view repositories that are available in the lifecycle environment of the activation key.
5. Optional: Use the **Repository type** dropdown menu to filter repositories by type.
6. Optional: Use the **Status** dropdown menu to filter repositories by status.
7. Select the desired repositories or click the **Select All** checkbox to select all repositories.
8. From the **Select Action** list, select **Override to Enabled**, **Override to Disabled**, or **Reset to Default**.

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 packages 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 imports this errata information when synchronizing repositories with Red Hat's Content Delivery Network (CDN). Red Hat Satellite 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

An erratum that applies to one or more content hosts, which means it updates packages present on the content host. Although these errata apply to content hosts, until their state changes to **Installable**, the errata are not ready to be installed. Installable errata are automatically applicable.

Installable

An erratum that applies to one or more content hosts and is available to install on the content host. Installable errata are available to a content host from lifecycle environment and the associated content view, but are not yet installed.

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

10.1. BEST PRACTICES FOR ERRATA

- Use errata to add patches for security issues to a frozen set of content without unnecessarily updating other unaffected packages.
- Automate errata management by using a Hammer script or an Ansible Playbook.
- View errata on the content hosts page and compare the errata of the current content view and lifecycle environment to the Library lifecycle environment, which contains the latest synchronized packages.
  You can only apply errata included in the content view version of the lifecycle of your host. You can view applicable errata as a recommendation to create an incremental content view to provide errata to hosts. For more information, see Section 10.9, "Adding errata to an incremental content view".

10.2. INSPECTING AVAILABLE ERRATA

The following procedure describes how to view and filter the available errata and how to display metadata of the selected advisory. To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Content Types** > **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** checkbox is selected by default to view only applicable errata in the selected repository. Select the **Installable** checkbox 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 Section 10.3, "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 10.11, "Applying errata to multiple hosts".
   - 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.

You can also use the new Host page to view to inspect available errata and select errata to install.

1. In the Satellite web UI, navigate to **Hosts > All Hosts** and select the host you require.
2. If there are errata associated with the host, an Installable Errata card on the new Host page displays an interactive pie chart showing a breakdown of the security advisories, bugfixes, and enhancements.
3. On the new Host page, select the **Content** tab.
4. On the Content page select the **Errata** tab.

5. The page displays installable errata for the chosen host.
6. Click the checkbox for any errata you wish to install.
7. Select **Apply via Remote Execution** to use Remote Execution, or **Apply via customized remote execution** if you want to customize the remote execution.
8. Click **Submit**.

CLI procedure

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

```
# hammer erratum list
```

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

```
# hammer erratum info --id erratum_ID
```

- You can search errata by entering the query with the **--search** option. For example, to view applicable errata for the selected product that contains the specified bugs ordered so that the security errata are displayed on top, enter the following command:

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

10.3. 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"*

Parameter	Description	Example
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*
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"*

10.4. APPLYING INSTALLABLE ERRATA

Use the following procedure to view a list of installable errata and select errata to install.

Procedure

1. In the Satellite web UI, navigate to **Hosts > All Hosts** and select the host you require.
2. If there are errata associated with the host, they are displayed in an Installable Errata card on the new Host page.
3. On the **Content** tab, **Errata** displays installable errata for the chosen host.
4. Click the checkbox for any errata you wish to install.
5. Using the vertical ellipsis icon next to the errata you want to add to the host, select **Apply via Remote Execution** to use Remote Execution. Select **Apply via customized remote execution** if you want to customize the remote execution.
6. Click **Submit**.

10.5. RUNNING CUSTOM CODE WHILE APPLYING ERRATA

You can use custom snippets to run code before and/or after applying errata on hosts.

Prerequisites

- Check your provisioning template to ensure that it supports the custom snippets you want to use.
  You can view all job templates that are in use under **Administer** > **Remote Execution Features**.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **Templates** > **Job Templates**.
2. Click **Create Template**.
3. In the **Name** field, enter a name for your custom snippet. The name must start with the name of a template that supports custom snippets:
   - Append **custom pre** to the name of a template to run code before applying errata on hosts.
   - Append **custom post** to the name of a template to run code after applying errata on hosts.

   If your template is called **Install Errata - Katello Ansible Default**, name your template **Install Errata - Katello Ansible Default custom pre** or **Install Errata - Katello Ansible Default custom post**.
4. On the **Type** tab, select **Snippet**.
5. Click **Submit** to create your custom snippet.

CLI procedure

1. Create a plain text file that contains your custom snippet.
2. Create the template using **hammer**:

```
# hammer template create \
--file "~/My_Snippet" \
--locations "My_Location" \
--name "My_Template_Name_custom_pre" \ --organizations "_My_Organization" \
--type snippet
```

10.6. 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 Configuring Email Notification Preferences in *Administering Red Hat Satellite*.

10.7. LIMITATIONS TO REPOSITORY DEPENDENCY RESOLUTION

With Satellite, using incremental updates to your content views solves some repository dependency problems. However, dependency resolution at a repository level still remains problematic on occasion.

When a repository update becomes available with a new dependency, Satellite retrieves the newest version of the package to solve the dependency, even if there are older versions available in the existing repository package. This can create further dependency resolution problems when installing packages.

Example scenario

A repository on your client has the package **example_repository-1.0** with the dependency **example_repository-libs-1.0**. The repository also has another package **example_tools-1.0**.

A security erratum becomes available with the package **example_tools-1.1**. The **example_tools-1.1** package requires the **example_repository-libs-1.1** package as a dependency.

After an incremental content view update, the **example_tools-1.1**, **example_tools-1.0**, and **example_repository-libs-1.1** are now in the repository. The repository also has the packages **example_repository-1.0** and **example_repository-libs-1.0**. Note that the incremental update to the content view did not add the package **example_repository-1.1**. Because you can install all these packages by using **dnf**, no potential problem is detected. However, when the client installs the **example_tools-1.1** package, a dependency resolution problem occurs because both **example_repository-libs-1.0** and **example_repository-libs-1.1** cannot be installed.

There is currently no workaround for this problem. The larger the time frame, and minor *Y* releases between the base set of packages and the errata being applied, the higher the chance of a problem with dependency resolution.

10.8. CREATING A CONTENT VIEW FILTER FOR ERRATA

You can use 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.

Create a content filter to exclude errata after a certain date. This ensures your production systems in the application lifecycle 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 lifecycle.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Prerequisites

- A content view with the repositories that contain required errata is created. For more information, see Section 7.4, "Creating a content view" .

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Content Views**.
2. Select a content view that you want to use for applying errata.
3. Select **Yum Content** > **Filters** and click **New Filter**.
4. In the **Name** field, enter **Errata Filter**.
5. From the **Content Type** list, select **Erratum – Date and Type**
6. From the **Inclusion Type** list, select **Exclude**.

7. In the **Description** field, enter **Exclude errata items from YYYY-MM-DD**.
8. Click **Save**.
9. For **Errata Type**, select the checkboxes of errata types you want to exclude. For example, select the **Enhancement** and **Bugfix** checkboxes and clear the **Security** checkbox to exclude enhancement and bugfix errata after certain date, but include all the security errata.
10. For **Date Type**, select one of two checkboxes:
    - **Issued On** for the issued date of the erratum.
    - **Updated On** for the date of the erratum's last update.
11. Select the **Start Date** to exclude all errata on or after the selected date.
12. Leave the **End Date** field blank.
13. Click **Save**.
14. Click **Publish New Version** to publish the resulting repository.
15. Enter **Adding errata filter** in the **Description** field.
16. 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.
17. Click the **Versions** tab.
18. Click **Promote** to the right of the published version.
19. Select the environments you want to promote the content view version to.
20. In the **Description** field, enter the description for promoting.
21. Click **Promote Version** to promote this content view version across the required environments.

CLI procedure

1. Create a filter for the errata:

```
# hammer content-view filter create \
--content-view "My_Content_View" \
--description "Exclude errata items from the YYYY-MM-DD" \
--name "My_Filter_Name" \
--organization "My_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 \
--content-view "My_Content_View" \
--content-view-filter="My_Content_View_Filter" \
```

```
--organization "My_Organization" \
--start-date "YYYY-MM-DD" \
--types=security,enhancement,bugfix
```

3. Publish the content view:

```
# hammer content-view publish \
--name "My_Content_View" \
--organization "My_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 "My_Content_View" \
--organization "My_Organization" \
--to-lifecycle-environment "My_Lifecycle_Environment"
```

10.9. ADDING ERRATA TO AN INCREMENTAL CONTENT VIEW

If errata are available but not installable, you can create an incremental content view version to add the errata to your content hosts. For example, if the content view is version 1.0, it becomes content view version 1.1, and when you publish, it becomes content view version 2.0.

IMPORTANT

If your content view version is old, you might encounter incompatibilities when incrementally adding enhancement errata. This is because enhancements are typically designed for the most current software in a repository.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Content Types** > **Errata**.
2. From the **Errata** list, click the name of the errata that you want to apply.
3. Select the content hosts that you want to apply the errata to, and click **Apply to Hosts**. This creates the incremental update to the content view.
4. If you want to apply the errata to the content host, select the **Apply Errata to Content Hosts immediately after publishing** checkbox.
5. Click **Confirm** to apply the errata.

CLI procedure

1. List the errata and its corresponding IDs:

```
# hammer erratum list
```

2. List the different content-view versions and the corresponding IDs:

```
# hammer content-view version list
```

3. Apply a single erratum to content-view version. You can add more IDs in a comma-separated list.

```
# hammer content-view version incremental-update \
--content-view-version-id 319 --errata-ids 34068b
```

10.10. APPLYING ERRATA TO HOSTS

Use these procedures to review and apply errata to hosts.

Prerequisites

- Synchronize Red Hat Satellite repositories with the latest errata available from Red Hat. For more information, see Section 4.7, "Synchronizing repositories".
- Register the host to an environment and content view on Satellite Server. For more information, see Registering Hosts in *Managing hosts*.
- Configure the host for remote execution. For more information about running remote execution jobs, see Configuring and Setting Up Remote Jobs in *Managing hosts*.

The procedure to apply an erratum to a host depends on its operating system.

10.10.1. Applying errata to hosts running Red Hat Enterprise Linux 9

Use this procedure to review and apply errata to a host running Red Hat Enterprise Linux 9.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

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.

CLI procedure

1. List all errata for the host:

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

2. Find the module stream an erratum belongs to:

```
# hammer erratum info --id ERRATUM_ID
```

3. On the host, update the module stream:

```
# dnf upgrade Module_Stream_Name
```

10.10.2. Applying errata to hosts running Red Hat Enterprise Linux 8

Use this procedure to review and apply errata to a host running Red Hat Enterprise Linux 8.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

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.

CLI procedure

1. List all errata for the host:

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

2. Find the module stream an erratum belongs to:

```
# hammer erratum info --id ERRATUM_ID
```

3. On the host, update the module stream:

```
# dnf upgrade Module_Stream_Name
```

10.10.3. Applying errata to hosts running Red Hat Enterprise Linux 7

Use this procedure to review and apply errata to a host running Red Hat Enterprise Linux 7.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

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.

CLI procedure

1. List all errata for the host:

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

2. Apply the most recent erratum to the host. Identify the erratum to apply using the erratum ID. Using **Remote Execution**

```
# hammer job-invocation create \
--feature katello_errata_install \
--inputs errata=ERRATUM_ID1,ERRATUM_ID2 \
--search-query "name = client.example.com"
```

10.11. APPLYING ERRATA TO MULTIPLE HOSTS

Use these procedures to review and apply errata to multiple RHEL hosts.

Prerequisites

- Synchronize Red Hat Satellite repositories with the latest errata available from Red Hat. For more information, see Section 4.7, "Synchronizing repositories".
- Register the hosts to an environment and content view on Satellite Server. For more information, see Registering Hosts in *Managing hosts*.
- Configure the host for remote execution. For more information about running remote execution jobs, see Configuring and Setting Up Remote Jobs in *Managing hosts*.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Content Types** > **Errata**.
2. Click the name of an erratum you want to apply.
3. Click to **Content Hosts** tab.
4. Select the hosts you want to apply errata to and click **Apply to Hosts**.
5. Click **Confirm**.

CLI procedure

1. List all installable errata:

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

2. Apply one of the errata to multiple hosts:

Using **Remote Execution**

```
# hammer job-invocation create \
--feature katello_errata_install \
--inputs errata=ERRATUM_ID \
--search-query "applicable_errata = ERRATUM_ID"
```

The following Bash script applies an erratum to each host for which this erratum is available:

```
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.

3. To see if an erratum is applied successfully, find the corresponding task in the output of the following command:

```
# hammer task list
```

4. View the state of a selected task:

```
# hammer task progress --id task_ID
```

10.12. APPLYING ERRATA TO A HOST COLLECTION

Using Remote Execution

```
# hammer job-invocation create \
--feature katello_errata_install \
--inputs errata=ERRATUM_ID1,ERRATUM_ID2,... \
--search-query "host_collection = HOST_COLLECTION_NAME"
```

CHAPTER 11. MANAGING CONTAINER IMAGES

With Satellite, you can import container images from various sources and distribute them to external containers by using content views.

For information about containers for Red Hat Enterprise Linux Atomic Host 7, see Getting Started with Containers in *Red Hat Enterprise Linux Atomic Host 7*.

For information about containers for Red Hat Enterprise Linux 8, see *Red Hat Enterprise Linux 8 Building, running, and managing containers*.

For information about containers for Red Hat Enterprise Linux 9, see *Red Hat Enterprise Linux 9 Building, running, and managing containers*.

11.1. IMPORTING CONTAINER IMAGES

You can import container image repositories from Red Hat Registry or from other image registries.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure with repository discovery

1. In the Satellite web UI, navigate to **Content** > **Products** and click **Repo Discovery**.
2. From the **Repository Type** list, select **Container Images**.
3. In the **Registry to Discover** field, enter the URL of the registry to import images from.
4. In the **Registry Username** field, enter the name that corresponds with your user name for the container image registry.
5. In the **Registry Password** field, enter the password that corresponds with the user name that you enter.
6. In the **Registry Search Parameter** field, enter any search criteria that you want to use to filter your search, and then click **Discover**.
7. Optional: To further refine the **Discovered Repository** list, in the **Filter** field, enter any additional search criteria that you want to use.
8. From the **Discovered Repository** list, select any repositories that you want to import, and then click **Create Selected**.
9. Optional: To change the download policy for this container repository to *on demand*, see Section 4.11, “Changing the download policy for a repository” .
10. Optional: If you want to create a product, from the **Product** list, select **New Product**.
11. In the **Name** field, enter a product name.
12. Optional: In the **Repository Name** and **Repository Label** columns, you can edit the repository names and labels.
13. Click **Run Repository Creation**.

14. When repository creation is complete, you can click each new repository to view more information.

15. Optional: To filter the content you import to a repository, click a repository, and then navigate to **Limit Sync Tags**. Click to edit, and add any tags that you want to limit the content that synchronizes to Satellite.

16. In the Satellite web UI, navigate to **Content** > **Products** and select the name of your product.

17. Select the new repositories and then click **Sync Now** to start the synchronization process.

Procedure with creating a repository manually

1. In the Satellite web UI, navigate to **Content** > **Products**. Click the name of the required product.

2. Click **New repository**.

3. From the **Type** list, select **docker**. Enter the details for the repository, and click **Save**.

4. Select the new repository, and click **Sync Now**.

Next steps

- To view the progress of the synchronization, navigate to **Content** > **Sync Status** and expand the repository tree.

- When the synchronization completes, you can click **Container Image Manifests** to list the available manifests. From the list, you can also remove any manifests that you do not require.

CLI procedure

1. Create the custom **Red Hat Container Catalog** product:

```
# hammer product create \
--description "My_Description" \
--name "Red Hat Container Catalog" \
--organization "My_Organization" \
--sync-plan "My_Sync_Plan"
```

2. Create the repository for the container images:

```
# hammer repository create \
--content-type "docker" \
--docker-upstream-name "rhel7" \
--name "RHEL7" \
--organization "My_Organization" \
--product "Red Hat Container Catalog" \
--url "http://registry.access.redhat.com/"
```

3. Synchronize the repository:

```
# hammer repository synchronize \
--name "RHEL7" \
--organization "My_Organization" \
--product "Red Hat Container Catalog"
```

Additional resources

- For more information about creating a product and repository manually, see Chapter 4, *Importing content*.

11.2. MANAGING CONTAINER NAME PATTERNS

When you use Satellite to create and manage your containers, as the container moves through content view versions and different stages of the Satellite lifecycle environment, the container name changes at each stage. For example, if you synchronize a container image with the name **ssh** from an upstream repository, when you add it to a Satellite product and organization and then publish as part of a content view, the container image can have the following name: **my_organization_production-custom_spin-my_product-custom_ssh**. This can create problems when you want to pull a container image because container registries can contain only one instance of a container name. To avoid problems with Satellite naming conventions, you can set a registry name pattern to override the default name to ensure that your container name is clear for future use.

Limitations

If you use a registry name pattern to manage container naming conventions, because registry naming patterns must generate globally unique names, you might experience naming conflict problems. For example:

- If you set the **repository.docker_upstream_name** registry name pattern, you cannot publish or promote content views with container content with identical repository names to the **Production** lifecycle.
- If you set the **lifecycle_environment.name** registry name pattern, this can prevent the creation of a second container repository with the identical name.

You must proceed with caution when defining registry naming patterns for your containers.

Procedure

To manage container naming with a registry name pattern, complete the following steps:

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Lifecycle Environments**.
2. Create a lifecycle environment or select an existing lifecycle environment to edit.
3. In the **Container Image Registry** area, click the edit icon to the right of **Registry Name Pattern** area.
4. Use the list of variables and examples to determine which registry name pattern you require.
5. In the **Registry Name Pattern** field, enter the registry name pattern that you want to use. For example, to use the **repository.docker_upstream_name**:

```
<%= repository.docker_upstream_name %>
```

6. Click **Save**.

11.3. MANAGING CONTAINER REGISTRY AUTHENTICATION

You can manage the authentication settings for accessing containers images from Satellite. By default, users must authenticate to access containers images in Satellite.

You can specify whether you want users to authenticate to access container images in Satellite in a lifecycle environment. For example, you might want to permit users to access container images from the **Production** lifecycle without any authentication requirement and restrict access the **Development** and **QA** environments to authenticated users.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Lifecycle Environments**.
2. Select the lifecycle environment that you want to manage authentication for.
3. To permit unauthenticated access to the containers in this lifecycle environment, select the **Unauthenticated Pull** checkbox. To restrict unauthenticated access, clear the **Unauthenticated Pull** checkbox.
4. Click **Save**.

11.4. CONFIGURING PODMAN AND DOCKER TO TRUST THE CERTIFICATE AUTHORITY

Podman uses two paths to locate the CA file, namely, **/etc/containers/certs.d/** and **/etc/docker/certs.d/**.

Copy the root CA file to one of these locations, with the exact path determined by the server hostname, and naming the file **ca.crt**

In the following examples, replace **hostname.example.com** with **satellite.example.com** or **capsule.example.com**, depending on your use case.

- You might first need to create the relevant location using:

```
# mkdir -p /etc/containers/certs.d/hostname.example.com
```

or

```
# mkdir -p /etc/docker/certs.d/hostname.example.com
```

- For podman, use:

```
# cp rootCA.pem /etc/containers/certs.d/hostname.example.com/ca.crt
```

- Alternatively, if you are using Docker, copy the root CA file to the equivalent Docker directory:

```
# cp rootCA.pem /etc/docker/certs.d/hostname.example.com/ca.crt
```

You no longer need to use the **--tls-verify=false** option when logging in to the registry:

```
$ podman login hostname.example.com

Username: admin
Password:
Login Succeeded!
```

11.5. USING CONTAINER REGISTRIES

You can use Podman and Docker to fetch content from container registries and push the content to the Satellite container registry. The Satellite registry follows the Open Containers Initiative (OCI) specification, so you can push content to Satellite by using the same methods that apply to other registries. For more information about OCI, see Open Container Initiative Distribution Specification .

Prerequisites

- To push content to Satellite, ensure your Satellite account has the **edit_products** permission.
- To pull content from Satellite, ensure that your Satellite account has the **view_lifecycle_environments**, **view_products**, and **view_content_views** permissions, unless the lifecycle environment allows unauthenticated pull.

Container registries on Capsules

On Capsules with content, the Container Gateway Capsule plugin acts as the container registry. It caches authentication information from Katello and proxies incoming requests to Pulp. The Container Gateway is available by default on Capsules with content.

Considerations for pushing content to the Satellite container registry

- You can only push content to the Satellite Server itself. If you need pushed content on Capsule Servers as well, use Capsule syncing.
- The pushed container registry name must contain only lowercase characters.
- Unless pushed repositories are published in a content view version, they do not follow the registry name pattern. For more information, see Section 11.2, “Managing container name patterns”. This is to ensure that users can push and pull from the same path.
- Users are required to push and pull from the same path. If you use the label-based schema, pull using labels. If you use the ID-based schema, pull using IDs.

Procedure

- Logging in to the container registry:

```
# podman login satellite.example.com
```

- Listing container images:

```
# podman search satellite.example.com/
```

- Pulling container images:

```
# podman pull satellite.example.com/my-image:<optional_tag>
```

- Pushing container images to the Satellite container registry:
  - To indicate which organization, product, and repository the container image belongs to, include the organization and product in the container registry name.
  - You can address the container destination by using one of the following schemas:

```
$ podman push My_Container_Image_Hash
satellite.example.com/My_Organization_Label/My_Product_Label/My_Repository_Name
[:_My_Tag_]
$ podman push My_Container_Image_Hash
satellite.example.com/id/My_Organization_ID/My_Product_ID/My_Repository_Name[:_M
y_Tag_]
```

- After the content push has completed, a repository is created in Satellite.

CHAPTER 12. MANAGING ISO IMAGES

You can use Satellite 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.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

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

CLI procedure

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 the repository in the product:

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

4. Synchronize the repository in the product:

```
# 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. To import files, you can complete the following steps in the Satellite web UI or using the Hammer CLI. However, if the size of the file that you want to upload is larger than 15 MB, you must use the Hammer CLI to upload it to a repository.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products** and click **Create Product**.
2. In the **Name** field, enter a name to identify the product. This name populates the **Label** field.
3. Optional: In the **GPG Key** field, enter a GPG Key for the product.
4. Optional: From the **Sync Plan** list, select a synchronization plan for the product.
5. Optional: 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. Select the new repository.
13. Navigate to **Upload File** and click **Browse**.
14. Select the **.iso** file and click **Upload**.

CLI procedure

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 \
--id repo_ID \
--organization "My_Organization"
```

CHAPTER 13. MANAGING ANSIBLE CONTENT

You can import Ansible collections from several sources to Satellite Server.

For more information about Ansible integration in Satellite, see *Managing configurations by using Ansible integration*.

13.1. SYNCHRONIZING ANSIBLE COLLECTIONS

On Satellite, you can synchronize your Ansible Collections from Private Automation Hub, **console.redhat.com**, and other Satellite instances. Ansible Collections will appear on Satellite as a new repository type in the Satellite web UI menu under **Content** after the sync.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Select the required product name.
3. In the **Products** window, select the name of a product that you want to create a repository for.
4. Click the **Repositories** tab, and then click **New Repository**.
5. In the **Name** field, enter a name for the repository.
   The **Label** field is populated automatically based on the name.
6. From the **Type** list, select **ansible collection**.
7. In the **Upstream URL** field, enter the URL for the upstream collections repository.
   The URL can be any Ansible Galaxy endpoint. For example,
   https://console.redhat.com/api/automation-hub/.
8. Optional: In the **Requirements.yml** field, you can specify the list of collections you want to sync from the endpoint, as well as their versions.
   If you do not specify the list of collections, everything from the endpoint will be synced.

   ```
   ---
   collections:
   - name: my_namespace.my_collection
     version: 1.2.3
   ```

   For more information, see Installing roles and collections from the same requirements.yml file in the *Galaxy User Guide*.
9. Authenticate.
   a. To sync Satellite from **Private Automation Hub**, enter your token in the **Auth Token** field.
      For more information, see Connect Private Automation Hub in *Connect to Hub*.
   b. To sync Satellite from **console.redhat.com**, enter your token in the **Auth Token** field and enter your SSO URL in the the **Auth URL** field.
      For more information, see *Getting started with automation hub*.
   c. To sync Satellite from Satellite, leave both authentication fields blank.

10. Click **Save**.
11. Navigate to the Ansible Collections repository.
12. From the **Select Action** menu, select **Sync Now**.

CHAPTER 14. MANAGING CUSTOM FILE TYPE CONTENT

In Satellite, you might require methods of managing and distributing SSH keys and source code files or larger files such as virtual machine images and ISO files. To achieve this, custom products in Red Hat Satellite include repositories for custom file types. This provides a generic method to incorporate arbitrary files in a product.

You can upload files to the repository and synchronize files 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. You must download the files on clients over HTTP or HTTPS by using **curl -O**.

You can create a file type repository in Satellite Server only in a custom product, but there is flexibility in how you create the repository source. You can create an independent repository source in a directory on Satellite Server, 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.

14.1. CREATING A LOCAL SOURCE FOR A CUSTOM FILE TYPE REPOSITORY

You can create a custom file type repository source, from a directory of files, on the base system where Satellite is installed using **Pulp Manifest**. You can then synchronize the files into a repository and manage the custom file type content like any other content type.

Use this procedure to configure 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 Section 14.2, “Creating a remote source for a custom file type repository”.

Procedure

1. Ensure the Utils repository is enabled.

```
# subscription-manager repos \
--enable=rhel-8-for-x86_64-appstream-rpms \
--enable=rhel-8-for-x86_64-baseos-rpms \
--enable=satellite-utils-6.16-for-rhel-8-x86_64-rpms
```

2. Enable the satellite-utils module:

```
# dnf module enable satellite-utils
```

3. Install the Pulp Manifest package:

```
# satellite-maintain packages install pulp-manifest
```

Note that this command stops the Satellite service and re-runs **satellite-installer**. Alternatively, to prevent downtime caused by stopping the service, you can use the following:

```
# satellite-maintain packages unlock
# satellite-maintain packages install pulp-manifest
# satellite-maintain packages lock
```

4. Create a directory that you want to use as the file type repository, such as:

```
# mkdir -p /var/lib/pulp/local_repos/my_file_repo
```

5. Add the parent folder into allowed import paths:

```
# satellite-installer --foreman-proxy-content-pulpcore-additional-import-paths
/var/lib/pulp/local_repos
```

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

```
# touch /var/lib/pulp/local_repos/my_file_repo/test.txt
```

7. Run the Pulp Manifest command to create the manifest:

```
# pulp-manifest /var/lib/pulp/local_repos/my_file_repo
```

8. Verify the manifest was created:

```
# ls /var/lib/pulp/local_repos/my_file_repo
PULP_MANIFEST test.txt
```

Now, you can import your local source as a custom file type repository. Use the **file:**// URL scheme and the name of the directory to specify an **Upstream URL**, such as **file:///var/lib/pulp/*local_repos*/*my_file_repo***. For more information, see Section 14.3, "Creating a custom file type repository".

If you update the contents of your directory, re-run Pulp Manifest and sync the repository in Satellite. For more information, see Section 4.7, "Synchronizing repositories".

14.2. CREATING A REMOTE SOURCE FOR A CUSTOM FILE TYPE REPOSITORY

You can create a custom file type repository source from a directory of files that is external to Satellite Server using **Pulp Manifest**. You can then synchronize the files into a repository on Satellite Server over HTTP or HTTPS and manage the custom file type content like any other content type.

Use this procedure to configure 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 14.1, "Creating a local source for a custom file type repository".

Prerequisites

- You have a server running Red Hat Enterprise Linux 8 registered to your Satellite or the Red Hat CDN.
- Your server has an entitlement to the Red Hat Enterprise Linux Server and Red Hat Satellite Utils repositories.
- You have installed an HTTP server. For more information about configuring a web server, see Setting up the Apache HTTP web server in Red Hat Enterprise Linux 8 *Deploying different types of servers*.

Procedure

1. On your server, enable the required repositories:

```
# subscription-manager repos \
--enable=rhel-8-for-x86_64-appstream-rpms \
--enable=rhel-8-for-x86_64-baseos-rpms \
--enable=satellite-utils-6.16-for-rhel-8-x86_64-rpms
```

2. Enable the satellite-utils module:

```
# dnf module enable satellite-utils
```

3. Install the Pulp Manifest package:

```
# dnf install pulp-manifest
```

4. 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
```

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

```
# touch /var/www/html/pub/my_file_repo/test.txt
```

6. Run the Pulp Manifest command to create the manifest:

```
# pulp-manifest /var/www/html/pub/my_file_repo
```

7. Verify the manifest was created:

```
# ls /var/www/html/pub/my_file_repo
PULP_MANIFEST test.txt
```

Now, you can import your remote source as a custom file type repository. Use the path to the directory to specify an **Upstream URL**, such as **http://server.example.com/*my_file_repo***. For more information, see Section 14.3, "Creating a custom file type repository" .

If you update the contents of your directory, re-run Pulp Manifest and sync the repository in Satellite. For more information, see Section 4.7, "Synchronizing repositories".

14.3. CREATING A CUSTOM FILE TYPE REPOSITORY

The procedure for creating a custom file type repository is the same as the procedure for creating any custom content, except that when you create the repository, you select the **file** type. You must create a product and then add a custom repository.

To use the CLI instead of the Satellite web UI, see the CLI procedure.

Procedure

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

2. Select a product that you want to create a repository for.
3. On the **Repositories** tab, click **New Repository**.
4. In the **Name** field, enter a name for the repository. Satellite automatically completes the **Label** field based on the name.
5. Optional: In the **Description** field, enter a description for the repository.
6. From the **Type** list, select **file** as type of repository.
7. Optional: In the **Upstream URL** field, enter the URL of the upstream repository to use as a source. If you do not enter an upstream URL, you can manually upload packages. For more information, see Section 14.4, “Uploading files to a custom file type repository” .
8. Select **Verify SSL** to verify that the SSL certificates of the repository are signed by a trusted CA.
9. Optional: In the **Upstream Username** field, enter the user name for the upstream repository if required for authentication. Clear this field if the repository does not require authentication.
10. Optional: In the **Upstream Password** field, enter the corresponding password for the upstream repository. Clear this field if the repository does not require authentication.
11. Optional: In the **Upstream Authentication Token** field, provide the token of the upstream repository user for authentication. Leave this field empty if the repository does not require authentication.
12. From the **Mirroring Policy** list, select the type of content synchronization Satellite Server performs. For more information, see Section 4.12, “Mirroring policies overview” .
13. Optional: In the **HTTP Proxy Policy** field, select an HTTP proxy. By default, it uses the **Global Default** HTTP proxy.
14. Optional: You can clear the **Unprotected** checkbox to require a subscription entitlement certificate for accessing this repository. By default, the repository is published through HTTP.
15. Optional: In the **SSL CA Cert** field, select the SSL CA Certificate for the repository.
16. Optional: In the **SSL Client Cert** field, select the SSL Client Certificate for the repository.
17. Optional: In the **SSL Client Key** field, select the SSL Client Key for the repository.
18. Click **Save** to create the repository.

CLI procedure

1. Create a custom product:

```
# hammer product create \
--description "My_Files" \
--name "My_File_Product" \
--organization "My_Organization" \
--sync-plan "My_Sync_Plan"
```

Table 14.1. Optional parameters for the hammer product create command

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

2. Create a **file** type repository:

```
# hammer repository create \
--content-type "file" \
--name "My_Files" \
--organization "My_Organization" \
--product "My_File_Product"
```

Table 14.2. Optional parameters for the hammer repository create command

Option	Description
--checksum-type *sha_version*	Repository checksum (either **sha256**, **sha384**, or **sha512**)
--download-policy *policy_name*	Download policy for repositories (either **immediate** or **on_demand**)
--gpg-key-id *gpg_key_id*	GPG key numeric identifier
--gpg-key *gpg_key_name*	Key name to search by
--mirror-on-sync *boolean*	Must this repo be mirrored from the source, and stale packages 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-password *repository_password*	Password for the upstream repository user
--upstream-username *repository_username*	Upstream repository user, if required for authentication
--url *My_Repository_URL*	URL of the remote repository
--verify-ssl-on-sync *boolean*	Verify that the upstream SSL certificates of the remote repository are signed by a trusted CA? Set to **true** or **false**, **yes** or **no**, **1** or **0**.

14.4. UPLOADING FILES TO A CUSTOM FILE TYPE REPOSITORY

Use this procedure to upload files to a custom file type repository.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Select a custom product by name.
3. Select a file type repository by name.
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.

CLI procedure

```
# hammer repository upload-content \
--id repo_ID \
--organization "My_Organization" \
--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.

14.5. DOWNLOADING FILES TO A HOST FROM A CUSTOM FILE TYPE REPOSITORY

You can download files to a client over HTTPS using **curl -O**, and optionally over HTTP if the **Unprotected** option for repositories is selected.

Prerequisites

- You have a custom file type repository. For more information, see Section 14.3, “Creating a custom file type repository”.
- You know the name of the file you want to download to clients from the file type repository.
- To use HTTPS you require the following certificates on the client:
  - The **katello-server-ca.crt**. For more information, see Importing the Katello Root CA Certificate in *Administering Red Hat Satellite*.
  - An Organization Debug Certificate. For more information, see Creating an Organization Debug Certificate in *Administering Red Hat Satellite*.

Procedure

1. In the Satellite web UI, navigate to **Content** > **Products**.
2. Select a custom product by name.

3. Select a file type repository by name.
4. Ensure to select the **Unprotected** checkbox to access the repository published through HTTP.
5. Copy the **Published At** URL.
6. On your client, download the file from Satellite Server:
   - For HTTPS:

```
# curl \
--cacert ./_katello-server-ca.crt \
--cert ./_My_Organization_key-cert.pem \
--remote-name \
https://satellite.example.com/pulp/content/My_Organization_Label/Library/custom/My_Product_Label/My_Repository_Label/My_File
```

   - For HTTP:

```
# curl \
--remote-name \
http://satellite.example.com/pulp/content/My_Organization_Label/Library/custom/My_Product_Label/My_Repository_Label/My_File
```

CLI procedure

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" \
--organization-id My_Organization_ID \
--product "My_File_Product"
```

If **Unprotected** is enabled, the output is similar to this:

```
Publish Via HTTP: yes
Published At:
https://satellite.example.com/pulp/content/My_Organization_Label/Library/custom/My_File_Product_Label/My_Files_Label/
```

If **Unprotected** is not enabled, the output is similar to this:

```
Publish Via HTTP: no
Published At:
https://satellite.example.com/pulp/content/My_Organization_Label/Library/custom/My_File_Pr
```

```
oduct_Label/My_Files_Label/
```

3. On your client, download the file from Satellite Server:

- For HTTPS:

```
# curl \
--cacert ./_katello-server-ca.crt \
--cert ./_My_Organization_key-cert.pem \
--remote-name \
https://satellite.example.com/pulp/content/My_Organization_Label/Library/custom/My_Product_Label/My_Repository_Label/My_File
```

- For HTTP:

```
# curl \
--remote-name \
http://satellite.example.com/pulp/content/My_Organization_Label/Library/custom/My_Product_Label/My_Repository_Label/My_File
```

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

Use high-bandwidth, low-latency storage for the **/var/lib/pulp** file system. Red Hat Satellite has many I/O-intensive operations; therefore, high-latency, low-bandwidth storage might have issues with performance degradation.

Procedure

1. Create the NFS share. This example uses a share at **nfs.example.com:/Satellite/pulp**. Ensure this share provides the appropriate permissions to Satellite Server and its **apache** user.
2. Stop Satellite services on your Satellite Server:

```
# satellite-maintain service stop
```

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

```
# satellite-maintain packages install nfs-utils
```

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

```
# mkdir /mnt/temp
# mount -o rw nfs.example.com:/Satellite/pulp /mnt/temp
```

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

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

5. Set the permissions for all files on the share to use the **pulp** user.
6. Unmount the temporary storage location:

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

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

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

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

```
nfs.example.com:/Satellite/pulp    /var/lib/pulp   nfs
rw,hard,intr,context="system_u:object_r:pulpcore_var_lib_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**:

```
# df
Filesystem                      1K-blocks     Used Available Use% Mounted on
...
nfs.example.com:/Satellite/pulp 309506048 58632800 235128224  20% /var/lib/pulp
...
```

Also confirm that the existing content exists at the mount on **var/lib/pulp**:

```
# ls /var/lib/pulp
```

11. Start Satellite services on your Satellite Server:

```
# satellite-maintain service start
```

Satellite Server now uses the NFS share to store content. Run a content synchronization to ensure the NFS share works as expected. For more information, see Section 4.7, “Synchronizing repositories”.

APPENDIX B. IMPORTING KICKSTART REPOSITORIES

Kickstart repositories are not provided by the Content ISO image. To use Kickstart repositories in your disconnected Satellite, you must download a binary DVD ISO file for the version of Red Hat Enterprise Linux that you want to use and copy the Kickstart files to Satellite.

B.1. IMPORTING KICKSTART REPOSITORIES FOR RED HAT ENTERPRISE LINUX 9

Use this procedure to import Kickstart repositories for Red Hat Enterprise Linux 9.

Procedure

1. Navigate to the Red Hat Customer Portal at access.redhat.com/downloads and log in.
2. Click **Red Hat Enterprise Linux**.
3. Select a product variant and a product version from the list. For example, product variant **Red Hat Enterprise Linux for x86_64** and product version **9.0**.
4. Locate the full installation image, for example, **Red Hat Enterprise Linux 9.0 Binary DVD**, and click **Download Now**. Note that you cannot provision hosts using the minimal ISO.
5. When the download completes, copy the ISO image to Satellite Server.
6. On Satellite Server, create a mount point and temporarily mount the ISO image at that location:

```
# mkdir /mnt/iso
# mount -o loop rhel-binary-dvd.iso /mnt/iso
```

   Replace *rhel-binary-dvd.iso* with the name of your ISO image.

7. Create directories for Red Hat Enterprise Linux 9 AppStream and BaseOS Kickstart repositories:

```
# mkdir --parents /var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/appstream/kickstart
# mkdir --parents /var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/baseos/kickstart
```

8. Copy the Kickstart files from the ISO image:

```
# cp -a /mnt/iso/AppStream/* /var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/appstream/kickstart
# cp -a /mnt/iso/BaseOS/* /mnt/iso/images/ /var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/baseos/kickstart
```

   Note that for BaseOS, you must also copy the contents of the **/mnt/iso/images/** directory.

9. Add the following entries to the listing files:
   To the **/var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/appstream/listing** file, append **kickstart** with a new line.

To the **/var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/baseos/listing** file, append **kickstart** with a new line.

To the **/var/www/html/pub/satellite-import/content/dist/rhel8/listing** file, append the version number with a new line. For example, for the Red Hat Enterprise Linux 9.0 binary ISO, append **9.0**.

10. Copy the **.treeinfo** files from the ISO image:

```
# cp /mnt/iso/.treeinfo /var/www/html/pub/satellite-
import/content/dist/rhel9/9.0/x86_64/appstream/kickstart/treeinfo
# cp /mnt/iso/.treeinfo /var/www/html/pub/satellite-
import/content/dist/rhel9/9.0/x86_64/baseos/kickstart/treeinfo
```

11. Open the **/var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/baseos/kickstart/treeinfo** file for editing.

12. In the **[general]** section, make the following changes:

    - Change **packagedir = AppStream/Packages** to **packagedir = Packages**
    - Change **repository = AppStream** to **repository = .**
    - Change **variant = AppStream** to **variant = BaseOS**
    - Change **variants = AppStream,BaseOS** to **variants = BaseOS**

13. In the **[tree]** section, change **variants = AppStream,BaseOS** to **variants = BaseOS**.

14. In the **[variant-BaseOS]** section, make the following changes:

    - Change **packages = BaseOS/Packages** to **packages = Packages**
    - Change **repository = BaseOS** to **repository = .**

15. Delete the **[media]** and **[variant-AppStream]** sections.

16. Save and close the file.

17. Verify that the **/var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/baseos/kickstart/treeinfo** file has the following format:

```
[checksums]
images/efiboot.img =
sha256:c01c18acc6778d6e66c8d0872bac59bfd7219ccf3cfa70a5c605c0fb37f33a83
images/install.img =
sha256:ddd08e5a5d92edee150f91ff4f12f39253eae72ff496465cf1b2766fe4a4df49
images/pxeboot/initrd.img =
sha256:a09a8ec89d485d71ed1bdad83584d6d816e67448221172d9aad97886cd70adca
images/pxeboot/vmlinuz =
sha256:6e523d7c3266e26c695923ab12b2873b16b0c61fb2e48ade608ad8998821584b

[general]
; WARNING.0 = This section provides compatibility with pre-productmd treeinfos.
; WARNING.1 = Read productmd documentation for details about new format.
arch = x86_64
family = Red Hat Enterprise Linux
```

```
name = Red Hat Enterprise Linux 9.0.0
packagedir = Packages
platforms = x86_64,xen
repository = .
timestamp = 1571146127
variant = BaseOS
variants = BaseOS
version = 9.0.0

[header]
type = productmd.treeinfo
version = 1.2

[images-x86_64]
efiboot.img = images/efiboot.img
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz

[images-xen]
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz

[release]
name = Red Hat Enterprise Linux
short = RHEL
version = 9.0.0

[stage2]
mainimage = images/install.img

[tree]
arch = x86_64
build_timestamp = 1571146127
platforms = x86_64,xen
variants = BaseOS

[variant-BaseOS]
id = BaseOS
name = BaseOS
packages = Packages
repository = .
type = variant
uid = BaseOS
```

18. Open the **/var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/appstream/kickstart/treeinfo** file for editing.

19. In the **[general]** section, make the following changes:

    - Change **packagedir = AppStream/Packages** to **packagedir = Packages**
    - Change **repository = AppStream** to **repository = .**
    - Change **variants = AppStream,BaseOS** to **variants = AppStream**

20. In the **[tree]** section, change **variants = AppStream,BaseOS** to **variants = AppStream**

21. In the **[variant-AppStream]** section, make the following changes:
    - Change **packages = AppStream/Packages** to **packages = Packages**
    - Change **repository = AppStream** to **repository = .**
22. Delete the following sections from the file: **[checksums]**, **[images-x86_64]**, **[images-xen]**, **[media]**, **[stage2]**, **[variant-BaseOS]**.
23. Save and close the file.
24. Verify that the **/var/www/html/pub/satellite-import/content/dist/rhel9/9.0/x86_64/appstream/kickstart/treeinfo** file has the following format:

```
[general]
; WARNING.0 = This section provides compatibility with pre-productmd treeinfos.
; WARNING.1 = Read productmd documentation for details about new format.
arch = x86_64
family = Red Hat Enterprise Linux
name = Red Hat Enterprise Linux 9.0.0
packagedir = Packages
platforms = x86_64,xen
repository = .
timestamp = 1571146127
variant = AppStream
variants = AppStream
version = 9.0.0

[header]
type = productmd.treeinfo
version = 1.2

[release]
name = Red Hat Enterprise Linux
short = RHEL
version = 9.0.0

[tree]
arch = x86_64
build_timestamp = 1571146127
platforms = x86_64,xen
variants = AppStream

[variant-AppStream]
id = AppStream
name = AppStream
packages = Packages
repository = .
type = variant
uid = AppStream
```

25. If you do not plan to use the mounted binary DVD ISO image, unmount and remove the directory:

```
# umount /mnt/iso
# rmdir /mnt/iso
```

26. In the Satellite web UI, enable the Kickstart repositories.

B.2. IMPORTING KICKSTART REPOSITORIES FOR RED HAT ENTERPRISE LINUX 8

Use this procedure to import Kickstart repositories for Red Hat Enterprise Linux 8.

Procedure

1. Navigate to the Red Hat Customer Portal at access.redhat.com/downloads and log in.
2. Click **Red Hat Enterprise Linux**.
3. Select a product variant and a product version from the list. For example, product variant **Red Hat Enterprise Linux for x86_64** and product version **8.1**.
4. Locate the full installation image, for example, **Red Hat Enterprise Linux 8.1 Binary DVD**, and click **Download Now**.
5. When the download completes, copy the ISO image to Satellite Server.
6. On Satellite Server, create a mount point and temporarily mount the ISO image at that location:

```
# mkdir /mnt/iso
# mount -o loop rhel-binary-dvd.iso /mnt/iso
```

   Replace *rhel-binary-dvd.iso* with the name of your ISO image.

7. Create directories for Red Hat Enterprise Linux 8 AppStream and BaseOS Kickstart repositories:

```
# mkdir --parents /var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/appstream/kickstart
# mkdir --parents /var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/baseos/kickstart
```

8. Copy the Kickstart files from the ISO image:

```
# cp -a /mnt/iso/AppStream/* /var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/appstream/kickstart
# cp -a /mnt/iso/BaseOS/* /mnt/iso/images/ /var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/baseos/kickstart
```

   Note that for BaseOS, you must also copy the contents of the **/mnt/iso/images/** directory.

9. Add the following entries to the listing files:
   To the **/var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/appstream/listing** file, append **kickstart** with a new line.

   To the **/var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/baseos/listing** file, append **kickstart** with a new line.

To the **/var/www/html/pub/satellite-import/content/dist/rhel8/listing** file, append the version number with a new line. For example, for the Red Hat Enterprise Linux 8.1 binary ISO, append **8.1**.

10. Copy the **.treeinfo** files from the ISO image:

```
# cp /mnt/iso/.treeinfo /var/www/html/pub/satellite-
import/content/dist/rhel8/8.1/x86_64/appstream/kickstart/treeinfo
# cp /mnt/iso/.treeinfo /var/www/html/pub/satellite-
import/content/dist/rhel8/8.1/x86_64/baseos/kickstart/treeinfo
```

11. Open the **/var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/baseos/kickstart/treeinfo** file for editing.

12. In the **[general]** section, make the following changes:

    - Change **packagedir = AppStream/Packages** to **packagedir = Packages**
    - Change **repository = AppStream** to **repository = .**
    - Change **variant = AppStream** to **variant = BaseOS**
    - Change **variants = AppStream,BaseOS** to **variants = BaseOS**

13. In the **[tree]** section, change **variants = AppStream,BaseOS** to **variants = BaseOS**.

14. In the **[variant-BaseOS]** section, make the following changes:

    - Change **packages = BaseOS/Packages** to **packages = Packages**
    - Change **repository = BaseOS** to **repository = .**

15. Delete the **[media]** and **[variant-AppStream]** sections.

16. Save and close the file.

17. Verify that the **/var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/baseos/kickstart/treeinfo** file has the following format:

```
[checksums]
images/efiboot.img =
sha256:c01c18acc6778d6e66c8d0872bac59bfd7219ccf3cfa70a5c605c0fb37f33a83
images/install.img =
sha256:ddd08e5a5d92edee150f91ff4f12f39253eae72ff496465cf1b2766fe4a4df49
images/pxeboot/initrd.img =
sha256:a09a8ec89d485d71ed1bdad83584d6d816e67448221172d9aad97886cd70adca
images/pxeboot/vmlinuz =
sha256:6e523d7c3266e26c695923ab12b2873b16b0c61fb2e48ade608ad8998821584b

[general]
; WARNING.0 = This section provides compatibility with pre-productmd treeinfos.
; WARNING.1 = Read productmd documentation for details about new format.
arch = x86_64
family = Red Hat Enterprise Linux
name = Red Hat Enterprise Linux 8.1.0
packagedir = Packages
platforms = x86_64,xen
```

```
repository = .
timestamp = 1571146127
variant = BaseOS
variants = BaseOS
version = 8.1.0

[header]
type = productmd.treeinfo
version = 1.2

[images-x86_64]
efiboot.img = images/efiboot.img
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz

[images-xen]
initrd = images/pxeboot/initrd.img
kernel = images/pxeboot/vmlinuz

[release]
name = Red Hat Enterprise Linux
short = RHEL
version = 8.1.0

[stage2]
mainimage = images/install.img

[tree]
arch = x86_64
build_timestamp = 1571146127
platforms = x86_64,xen
variants = BaseOS

[variant-BaseOS]
id = BaseOS
name = BaseOS
packages = Packages
repository = .
type = variant
uid = BaseOS
```

18. Open the **/var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/appstream/kickstart/treeinfo** file for editing.
19. In the **[general]** section, make the following changes:
    - Change **packagedir = AppStream/Packages** to **packagedir = Packages**
    - Change **repository = AppStream** to **repository = .**
    - Change **variants = AppStream,BaseOS** to **variants = AppStream**
20. In the **[tree]** section, change **variants = AppStream,BaseOS** to **variants = AppStream**
21. In the **[variant-AppStream]** section, make the following changes:
    - Change **packages = AppStream/Packages** to **packages = Packages**

- Change **repository = AppStream** to **repository = .**

22. Delete the following sections from the file: **[checksums]**, **[images-x86_64]**, **[images-xen]**, **[media]**, **[stage2]**, **[variant-BaseOS]**.

23. Save and close the file.

24. Verify that the **/var/www/html/pub/satellite-import/content/dist/rhel8/8.1/x86_64/appstream/kickstart/treeinfo** file has the following format:

```
[general]
; WARNING.0 = This section provides compatibility with pre-productmd treeinfos.
; WARNING.1 = Read productmd documentation for details about new format.
arch = x86_64
family = Red Hat Enterprise Linux
name = Red Hat Enterprise Linux 8.1.0
packagedir = Packages
platforms = x86_64,xen
repository = .
timestamp = 1571146127
variant = AppStream
variants = AppStream
version = 8.1.0

[header]
type = productmd.treeinfo
version = 1.2

[release]
name = Red Hat Enterprise Linux
short = RHEL
version = 8.1.0

[tree]
arch = x86_64
build_timestamp = 1571146127
platforms = x86_64,xen
variants = AppStream

[variant-AppStream]
id = AppStream
name = AppStream
packages = Packages
repository = .
type = variant
uid = AppStream
```

25. If you do not plan to use the mounted binary DVD ISO image, unmount and remove the directory:

```
# umount /mnt/iso
# rmdir /mnt/iso
```

26. In the Satellite web UI, enable the Kickstart repositories.

B.3. IMPORTING KICKSTART REPOSITORIES FOR RED HAT ENTERPRISE LINUX 7

Use this procedure to import Kickstart repositories for Red Hat Enterprise Linux 7.

Procedure

1. Navigate to the Red Hat Customer Portal at access.redhat.com/downloads and log in.
2. Click **Red Hat Enterprise Linux**.
3. Click **Switch to version 7 and below**above the **Product Variant** list.
4. Select a product variant and a product version from the list. For example, product variant **Red Hat Enterprise Linux for x86_64**and product version **7.9**.
5. Locate the full installation image, for example, **Red Hat Enterprise Linux 7.9 Binary DVD**, and click **Download Now**.
6. When the download completes, copy the ISO image to Satellite Server.
7. On Satellite Server, create a mount point and temporarily mount the ISO image at that location:

```
# mkdir /mnt/iso
# mount -o loop rhel-binary-dvd.iso /mnt/iso
```

   Replace *rhel-binary-dvd.iso* with the name of your ISO image.

8. Create Kickstart directories:

```
# mkdir --parents /var/www/html/pub/satellite-
import/content/dist/rhel/server/7/7.9/x86_64/kickstart/
```

9. Copy the Kickstart files from the ISO image:

```
# cp -a /mnt/iso/* /var/www/html/pub/satellite-
import/content/dist/rhel/server/7/7.9/x86_64/kickstart/
```

10. Add the following entries to the listing files:
    To the **/var/www/html/pub/satellite-import/content/dist/rhel/server/7/listing** file, append the version number with a new line. For example, for the Red Hat Enterprise Linux 7.9 ISO, append **7.9**.

    To the **/var/www/html/pub/satellite-import/content/dist/rhel/server/7/7.9/listing** file, append the architecture with a new line. For example, **x86_64**.

    To the **/var/www/html/pub/satellite-import/content/dist/rhel/server/7/7.9/x86_64/listing** file, append **kickstart** with a new line.

11. Copy the **.treeinfo** files from the ISO image:

```
# cp /mnt/iso/.treeinfo /var/www/html/pub/satellite-
import/content/dist/rhel/server/7/7.9/x86_64/kickstart/treeinfo
```

12. If you do not plan to use the mounted binary DVD ISO image, unmount and remove the directory:

```
# umount /mnt/iso
# rmdir /mnt/iso
```

13. In the Satellite web UI, enable the Kickstart repositories.