Red Hat Satellite 6.17

Using the Satellite REST API

Develop custom applications or integrations by using the Satellite REST API

Last Updated: 2025-11-19

Red Hat Satellite 6.17 Using the Satellite REST API

Develop custom applications or integrations by using the Satellite REST API

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

Legal Notice

Abstract

The Red Hat Satellite Representational State Transfer (REST) API guide explains the concepts behind a REST API and provides example usage for various types of requests. This provides a basis for administrators and developers to write custom scripts and integrate Red Hat Satellite with third-party applications.

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 SATELLITE API

Red Hat Satellite provides a Representational State Transfer (REST) API. The API provides software developers and system administrators with control over their Red Hat Satellite environment outside of the standard web interface. The REST API is useful for developers and administrators who aim to integrate the functionality of Red Hat Satellite with custom scripts or external applications that access the API over HTTP.

1.1. OVERVIEW OF THE SATELLITE API

The benefits of using the REST API are:

- Broad client support – any programming language, framework, or system with support for HTTP protocol can use the API.
- Self-descriptive – client applications require minimal knowledge of the Red Hat Satellite infrastructure because a user discovers many details at runtime.
- Resource-based model – the resource-based REST model provides a natural way to manage a virtualization platform.

You can use the REST API to perform the following tasks:

- Integrate with enterprise IT systems.
- Integrate with third-party applications.
- Perform automated maintenance or error checking tasks.
- Automate repetitive tasks with scripts.

1.2. SATELLITE API COMPARED TO HAMMER CLI

For many tasks, you can use both Hammer and Satellite API. You can use Hammer as a human-friendly interface to Satellite API. For example, to test responses to API calls before applying them in a script, use the **--debug** option to inspect API calls that Hammer issues: **hammer --debug organization list**. In contrast, scripts that use API commands communicate directly with the Satellite API.

For more information, see the *Using the Hammer CLI tool* .

CHAPTER 2. ACCESSING THE BUILT-IN API REFERENCE

You can access the full API reference on your Satellite Server.

Procedure

- In your browser, access the following URL:

  ```
  https://satellite.example.com/apidoc/
  ```

  Replace *satellite.example.com* with the FQDN of your Satellite Server.

CHAPTER 3. API SYNTAX

You can review the basic syntax of API requests and JSON responses.

IMPORTANT

Even though versions 1 and 2 of the Satellite 6 API are available, Red Hat only supports version 2.

3.1. API REQUEST COMPOSITION

The built-in API reference shows the API route, or path, preceded by an HTTP method:

```
HTTP_METHOD API_ROUTE
```

To work with the API, construct a command by using the **curl** command syntax and the API route from the reference document:

```
$ curl \
--request HTTP_METHOD \ 1
--insecure \ 2
--user My_User_Name:My_Password \ 3
--data @My_Input_File.json \ 4
--header "Accept:application/json" \ 5
--header "Content-Type:application/json" \ 6
--output My_Output_File 7
API_ROUTE \ 8
| python3 -m json.tool 9
```

1. To use **curl** for the API call, specify an HTTP method with the **--request** option. For example, **--request POST**.
2. Add the **--insecure** option to skip SSL peer certificate verification check. Red Hat recommends you to configure SSL authentication and use secured calls. For more information, see Section 4.1, "SSL authentication overview".
3. Provide Satellite user credentials with the **--user** option.
4. For **POST** and **PUT** requests, use the **--data** option to pass JSON-formatted data. For more information, see Section 5.1.1, "Passing JSON data to the API request" .
5. 6. When passing the JSON data with the **--data** option, you must specify the following headers with the **--header** option. For more information, see Section 5.1.1, "Passing JSON data to the API request".
7. When downloading content from Satellite Server, specify the output file with the **--output** option.
8. Use the API route in the following format: **https://*satellite.example.com*/katello/api/activation_keys**. In Satellite 6, version 2 of the API is the default. Therefore, it is not necessary to use **v2** in the URL for API calls.
9. Redirect the output to the Python **json.tool** module to make the output easier to read.

3.1.1. Using the GET HTTP method

Use the GET HTTP method to get data from the API about an existing entry or resource. This example requests the number of registered hosts.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/hosts \
| python3 -m json.tool
```

API response

```
{
  "total": 2,
  "subtotal": 2,
  "page": 1,
  "per_page": 20,
  "search": null,
  "sort": {
    "by": null,
    "order": null
  },
  "results":
    output truncated
}
```

The response from the API indicates that there are two results in total, this is the first page of the results, and the maximum results per page is set to 20. For more information, see Section 3.2, "JSON response format".

3.1.2. Using the POST HTTP method

Use the POST HTTP verb to submit data to the API to create an entry or resource. You must submit the data in JSON format. For more information, see Section 5.1.1, "Passing JSON data to the API request" .

API procedure

1. Create a test file, for example, **activation-key.json**, with the following content:

   ```
   {"organization_id":1, "name":"TestKey", "description":"Just for testing"}
   ```

2. Create an activation key by applying the data in the **activation-key.json** file:
   Example request:

   ```
   $ curl \
   --header "Accept:application/json" \
   --header "Content-Type:application/json" \
   --request POST \
   --user My_User_Name:My_Password \
   --data @activation-key.json \
   https://satellite.example.com/katello/api/activation_keys \
   ```

```
| python3 -m json.tool
```

Example response:


```
{
    "id": 2,
    "name": "TestKey",
    "description": "Just for testing",
    "unlimited_hosts": true,
    "auto_attach": true,
    "content_view_id": null,
    "environment_id": null,
    "usage_count": 0,
    "user_id": 3,
    "max_hosts": null,
    "release_version": null,
    "service_level": null,
    "content_overrides": [

    ],
    "organization": {
        "name": "Default Organization",
        "label": "Default_Organization",
        "id": 1
    },
    "created_at": "2024-02-16 12:37:47 UTC",
    "updated_at": "2024-02-16 12:37:48 UTC",
    "content_view": null,
    "environment": null,
    "products": null,
    "host_collections": [

    ],
    "permissions": {
        "view_activation_keys": true,
        "edit_activation_keys": true,
        "destroy_activation_keys": true
    }
}
```


Verification

- In the Satellite web UI, navigate to **Content** > **Lifecycle** > **Activation Keys** to view your activation keys.

3.1.3. Using the PUT HTTP method

Use the PUT HTTP method to change an existing value or append to an existing resource. You must submit the data in JSON format. For more information, see Section 5.1.1, “Passing JSON data to the API request”.

This example updates the **TestKey** activation key created in the previous example.

API procedure

1. Edit the **activation-key.json** file created previously as follows:

```
{"organization_id":1, "name":"TestKey", "description":"Just for testing","max_hosts":"10" }
```

2. Apply the changes in the JSON file:
   Example request:

```
$ curl \
--request PUT \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--user My_User_Name:My_Password \
--data @activation-key.json \
https://satellite.example.com/katello/api/activation_keys/2 \
| python3 -m json.tool
```

   Example response:

```
{
    "id": 2,
    "name": "TestKey",
    "description": "Just for testing",
    "unlimited_hosts": false,
    "auto_attach": true,
    "content_view_id": null,
    "environment_id": null,
    "usage_count": 0,
    "user_id": 3,
    "max_hosts": 10,
    "release_version": null,
    "service_level": null,
    "content_overrides": [

    ],
    "organization": {
        "name": "Default Organization",
        "label": "Default_Organization",
        "id": 1
    },
    "created_at": "2024-02-16 12:37:47 UTC",
    "updated_at": "2024-02-16 12:46:17 UTC",
    "content_view": null,
    "environment": null,
    "products": null,
    "host_collections": [

    ],
    "permissions": {
        "view_activation_keys": true,
        "edit_activation_keys": true,
        "destroy_activation_keys": true
    }
}
```

3. In the Satellite web UI, verify the changes by navigating to **Content** > **Lifecycle** > **Activation Keys**.

3.1.4. Using the DELETE HTTP method

To delete a resource, use the DELETE method with an API route that includes the ID of the resource you want to delete. This example deletes the **TestKey** activation key which ID is 2.

API request

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request DELETE \
--user My_User_Name:My_Password \
https://satellite.example.com/katello/api/activation_keys/2 \
| python3 -m json.tool
```

API response

```
output omitted
    "started_at": "2024-02-16 12:58:17 UTC",
    "ended_at": "2024-02-16 12:58:18 UTC",
    "state": "stopped",
    "result": "success",
    "progress": 1.0,
    "input": {
        "activation_key": {
            "id": 2,
            "name": "TestKey"
output truncated
```

3.2. JSON RESPONSE FORMAT

Calls to the API return results in JSON format. The API call returns the result for a single-option response or for responses collections.

3.2.1. JSON response format for single objects

You can use single-object JSON responses to work with a single object. API requests to a single object require the unique identifier **:id** of the object.

This is an example of the format for a single-object request for the Satellite domain which ID is 23:

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/domains/23 \
| python3 -m json.tool
```

API response

```
{
    "id": 23,
    "name": "qa.lab.example.com",
    "fullname": "QA",
    "dns_id": 10,
    "created_at": "2024-08-13T09:02:31Z",
    "updated_at": "2024-08-13T09:02:31Z"
}
```


3.2.2. JSON response format for collections

Collections are a list of objects such as hosts and domains. The format for a collection JSON response consists of a metadata fields section and a results section.

This is an example of the format for a collection request for a list of Satellite domains:

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/domains \
| python3 -m json.tool
```

API response


```
{
    "total": 3,
    "subtotal": 3,
    "page": 1,
    "per_page": 20,
    "search": null,
    "sort": {
        "by": null,
        "order": null
    },
    "results": [
        {
            "id": 23,
            "name": "qa.lab.example.com",
            "fullname": "QA",
            "dns_id": 10,
            "created_at": "2024-08-13T09:02:31Z",
            "updated_at": "2024-08-13T09:02:31Z"
        },
        {
            "id": 25,
            "name": "dev.lab.example.com",
            "fullname": "DEVEL",
            "dns_id": 8,
            "created_at": "2024-08-13T08:32:48Z",
            "updated_at": "2024-08-14T07:04:03Z"
        },
        {
```

```
            "id": 32,
            "name": "hr.lab.example.com",
            "fullname": "HR",
            "dns_id": 8,
            "created_at": "2024-08-16T08:32:48Z",
            "updated_at": "2024-08-16T07:04:03Z"
        }
    ]
}
```

3.2.3. JSON response metadata

Satellite API responses contain the following metadata fields:

total

The total number of objects without any search parameters.

subtotal

The number of objects returned with the given search parameters. If there is no search, then subtotal is equal to total.

page

The page number.

per_page

The maximum number of objects returned per page.

limit

The specified number of objects to return in a collection response.

offset

The number of objects skipped before returning a collection.

search

The search string based on **scoped_scoped** syntax.

sort

- **by** – Specifies by what field the API sorts the collection.
- **order** – The sort order, either ASC for ascending or DESC for descending.

results

The collection of objects.

3.3. RELATING API ERROR MESSAGES TO THE API REFERENCE

The API uses a RAILs format to indicate an error:

```
Nested_Resource.Attribute_Name
```

This translates to the following format used in the API reference:

```
Resource[Nested_Resource_attributes][Attribute_Name_id]
```

CHAPTER 4. API CALL AUTHENTICATION

Interaction with the Satellite API requires SSL authentication with Satellite Server CA certificate and authentication with valid Satellite user credentials. You can use the following authentication methods.

4.1. SSL AUTHENTICATION OVERVIEW

Red Hat Satellite uses HTTPS, which provides a degree of encryption and identity verification when communicating with Satellite Server. Satellite 6.17 does not support non-SSL communications.

By default, Satellite Server uses a self-signed certificate. This certificate acts as both the server certificate to verify the encryption key and the certificate authority (CA) to trust the identity of Satellite Server.

You can configure Satellite Server to use a custom SSL certificate. For more information, see Configuring Satellite Server with a custom SSL certificate in *Installing Satellite Server in a connected network environment*. For more information on disconnected Satellite Server, see Configuring Satellite Server with a custom SSL certificate in *Installing Satellite Server in a disconnected network environment*.

4.1.1. Configuring SSL authentication

Configure an SSL authentication for the API requests to Satellite Server.

Procedure

1. Obtain a certificate from your Satellite Server by using one of the following options:
   - If you plan to call the API from a remote server, download the CA certificate:

     ```
     $ curl \
     --output /etc/pki/ca-trust/source/anchors/satellite.example.com-katello-server-ca.crt \
     http://satellite.example.com/pub/katello-server-ca.crt
     ```

   - If you plan to call the API directly on your Satellite Server, copy the certificate to the **/etc/pki/ca-trust/source/anchors** directory:

     ```
     # cp /var/www/html/pub/katello-server-ca.crt /etc/pki/ca-
     trust/source/anchors/satellite.example.com-katello-server-ca.crt
     ```

2. Add the certificate to the list of trusted CAs:

   ```
   $ update-ca-trust extract
   ```

Verification

- Verify that your client trusts the Satellite SSL certificate by entering the API request without the **--cacert** option:

  ```
  $ curl \
  --request GET \
  --user My_User_Name:My_Password \
  https://satellite.example.com/api/v2/hosts
  ```

4.2. HTTP AUTHENTICATION OVERVIEW

All requests to the Satellite API require a valid Satellite user name and password. The API uses Basic HTTP authentication to encode these credentials and add to the **Authorization** header. For more information about Basic authentication, see RFC 2617 HTTP Authentication: Basic and Digest Access Authentication. If a request does not include an appropriate **Authorization** header, the API returns a **401 Authorization Required** error.

IMPORTANT

Basic authentication involves potentially sensitive information, for example, it sends passwords as plain text. The REST API requires HTTPS for transport-level encryption of plain text requests.

Some base64 libraries break encoded credentials into multiple lines and terminate each line with a newline character. This invalidates the header and causes a faulty request. The **Authorization** header requires the encoded credentials to be on a single line within the header.

4.3. TOKEN AUTHENTICATION OVERVIEW

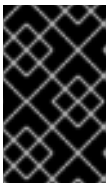
Red Hat Satellite supports Personal Access Tokens that you can use to authenticate API requests instead of using your password. You can set an expiration date for your Personal Access Token and you can revoke it if you decide it should expire before the expiration date.

4.3.1. Creating a Personal Access Token

Use this procedure to create a Personal Access Token.

Procedure

1. In the Satellite web UI, navigate to **Administer** > **Users**.
2. Select a user for which you want to create a Personal Access Token.
3. On the **Personal Access Tokens** tab, click **Add Personal Access Token**
4. Enter a **Name** for you Personal Access Token.
5. Optional: Select the **Expires** date to set an expiration date. If you do not set an expiration date, your Personal Access Token will never expire unless revoked.
6. Click Submit. You now have the Personal Access Token available to you on the **Personal Access Tokens** tab.



IMPORTANT

Ensure to store your Personal Access Token as you will not be able to access it again after you leave the page or create a new Personal Access Token. You can click **Copy to clipboard** to copy your Personal Access Token.

Verification

1. Make an API request to your Satellite Server and authenticate with your Personal Access Token:
   -

```
$ curl \
--user My_Username:My_Personal_Access_Token \
https://satellite.example.com/api/status
```

2. You should receive a response with status **200**, for example:

```
{"satellite_version":"6.17.0","result":"ok","status":200,"version":"3.5.1.10","api_version":2}
```

If you go back to **Personal Access Tokens** tab, you can see the updated **Last Used** time next to your Personal Access Token.

4.3.2. Revoking a Personal Access Token

Use this procedure to revoke a Personal Access Token before its expiration date.

Procedure

1. In the Satellite web UI, navigate to **Administer** > **Users**.
2. Select a user for which you want to revoke the Personal Access Token.
3. On the **Personal Access Tokens** tab, locate the Personal Access Token you want to revoke.
4. Click **Revoke** in the **Actions** column next to the Personal Access Token you want to revoke.

Verification

1. Make an API request to your Satellite Server and try to authenticate with the revoked Personal Access Token:

```
$ curl \
--user My_Username:My_Personal_Access_Token \
https://satellite.example.com/api/status
```

2. You receive the following error message:

```
{
  "error": {"message":"Unable to authenticate user My_Username"}
}
```

CHAPTER 5. API REQUESTS IN VARIOUS LANGUAGES

You can review the following examples of sending API requests to Red Hat Satellite from curl, Ruby, or Python.

5.1. CALLING THE API IN CURL

You can use **curl** with the Satellite API to perform various tasks.

Red Hat Satellite requires the use of HTTPS, and by default, a certificate for host identification. If you have not added the Satellite Server certificate as described in Section 4.1, "SSL authentication overview", then you can use the **--insecure** option to bypass certificate checks.

For user authentication, you can use the **--user** option to provide Satellite user credentials in the form **--user *My_User_Name:_My_Password***. If you do not include the password, the command prompts you to enter it. To reduce security risks, do not include the password as part of the command, because it then becomes part of your shell history. For simplicity, the examples in this section include the password.

Be aware that if you use the **--silent** option, **curl** does not display a progress meter or any error messages.

Examples in this chapter use the Python **json.tool** module to format the output.

5.1.1. Passing JSON data to the API request

You can pass data to Satellite Server with the API request. The data must be in JSON format. When specifying JSON data with the **--data** option, you must set the following HTTP headers with the **--header** option:

```
--header "Accept:application/json" \
--header "Content-Type:application/json"
```

Use one of the following options to include data with the **--data** option.

JSON-formatted string

Enclose the quoted JSON-formatted data in curly braces **{}**. When passing a value for a JSON type parameter, you must escape quotation marks **"** with backslashes \. For example, within curly braces, you must format **"Example JSON Variable"** as **"Example JSON Variable**\":

```
--data {"id":44, "smart_class_parameter":{"override":"true", "parameter_type":"json", "default_value":"
{\"GRUB_CMDLINE_LINUX\": {\"audit\":\"1\",\"crashkernel\":\"true\"}}"}}
```

JSON-formatted file

The unquoted JSON-formatted data enclosed in a file and specified by the @ sign and the filename. For example:

```
--data @file.json
```

Using external files for JSON formatted data has the following advantages:

- You can use your favorite text editor.

- You can use syntax checker to find and avoid mistakes.
- You can use tools to check the validity of JSON data or to reformat it.

Use the **json_verify** tool to check the validity of the JSON file:

```
$ json_verify < file.json
```

5.1.2. Retrieving a list of resources

This section outlines how to use **curl** with the Satellite 6 API to request information from Satellite. These examples include both requests and responses. Expect different results for each deployment.

5.1.2.1. Listing users

This example is a basic request that returns a list of Satellite resources, Satellite users in this case. Such requests return a list of data wrapped in metadata, while other request types only return the actual object.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/users \
| python3 -m json.tool
```

API response

```
{
    "page": 1,
    "per_page": 20,
    "results": [
        {
            "admin": false,
            "auth_source_id": 1,
            "auth_source_name": "Internal",
            "created_at": "2024-09-21 08:59:22 UTC",
            "default_location": null,
            "default_organization": null,
            "description": "",
            "effective_admin": false,
            "firstname": "",
            "id": 5,
            "last_login_on": "2024-09-21 09:03:25 UTC",
            "lastname": "",
            "locale": null,
            "locations": [],
            "login": "test",
            "mail": "test@example.com",
            "organizations": [
                {
                    "id": 1,
                    "name": "Default Organization"
```

```
            }
          ],
          "ssh_keys": [],
          "timezone": null,
          "updated_at": "2024-09-21 09:04:45 UTC"
        },
        {
          "admin": true,
          "auth_source_id": 1,
          "auth_source_name": "Internal",
          "created_at": "2024-09-20 07:09:41 UTC",
          "default_location": null,
          "default_organization": {
            "description": null,
            "id": 1,
            "name": "Default Organization",
            "title": "Default Organization"
          },
          "description": "",
          "effective_admin": true,
          "firstname": "Admin",
          "id": 4,
          "last_login_on": "2024-12-07 07:31:09 UTC",
          "lastname": "User",
          "locale": null,
          "locations": [
            {
              "id": 2,
              "name": "Default Location"
            }
          ],
          "login": "admin",
          "mail": "admin@example.com",
          "organizations": [
            {
              "id": 1,
              "name": "Default Organization"
            }
          ],
          "ssh_keys": [],
          "timezone": null,
          "updated_at": "2024-11-14 08:19:46 UTC"
        }
      ],
      "search": null,
      "sort": {
        "by": null,
        "order": null
      },
      "subtotal": 2,
      "total": 2
    }
```


5.1.3. Creating and modifying resources

You can use **curl** to manipulate resources on your Satellite Server. API calls to Satellite require data in **json** format. For more information, see Section 5.1.1, “Passing JSON data to the API request” .

5.1.4. Creating a user

Use this procedure to create a user.

API request

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request POST \
--user My_User_Name:My_Password \
--data "{\"firstname\":\"Test
Name\",\"mail\":\"test@example.com\",\"login\":\"test_user\",\"password\":\"password123\",\"auth_sour
ce_id\":1}" \
https://satellite.example.com/api/users \
| python3 -m json.tool
```

5.1.5. Modifying a user

This example modifies given name and login of the **test_user** that was created in API request.

API request

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request PUT \
--user My_User_Name:My_Password \
--data "{\"firstname\":\"New Test
Name\",\"mail\":\"test@example.com\",\"login\":\"new_test_user\",\"password\":\"password123\",\"auth
_source_id\":1}" \
https://satellite.example.com/api/users/8 \
| python3 -m json.tool
```

5.2. CALLING THE API IN RUBY

You can use Ruby with the Satellite API to perform various tasks.

IMPORTANT

These are example scripts and commands. Ensure you review these scripts carefully before use, and replace any variables, user names, passwords, and other information to suit your own deployment.

5.2.1. Creating objects by using Ruby

This script connects to the Red Hat Satellite 6 API, creates an organization, and then creates three lifecycle environments in the organization. If the organization already exists, the script uses that organization. If any of the lifecycle environments already exist in the organization, the script raises an error and quits.

```
#!/usr/bin/ruby

require 'rest-client'
require 'json'

url = 'https://satellite.example.com/api/v2/'
katello_url = "#{url}/katello/api/v2/"

$username = '_My_User_Name_'
$password = '_My_Password_'

org_name = "_My_Organization_"
environments = [ "Development", "Testing", "Production" ]

# Performs a GET by using the passed URL location
def get_json(location)
  response = RestClient::Request.new(
    :method => :get,
    :url => location,
    :user => $username,
    :password => $password,
    :headers => { :accept => :json,
    :content_type => :json }
  ).execute
  JSON.parse(response.to_str)
end

# Performs a POST and passes the data to the URL location
def post_json(location, json_data)
  response = RestClient::Request.new(
    :method => :post,
    :url => location,
    :user => $username,
    :password => $password,
    :headers => { :accept => :json,
    :content_type => :json},
    :payload => json_data
  ).execute
  JSON.parse(response.to_str)
end

# Creates a hash with ids mapping to names for an array of records
def id_name_map(records)
  records.inject({}) do |map, record|
    map.update(record['id'] => record['name'])
  end
end

# Get list of existing organizations
orgs = get_json("#{katello_url}/organizations")
org_list = id_name_map(orgs['results'])

if !org_list.has_value?(org_name)
  # If our organization is not found, create it
  puts "Creating organization: \t#{org_name}"
  org_id = post_json("#{katello_url}/organizations", JSON.generate({"name"=> org_name}))["id"]
```

```
else
  # Our organization exists, so let's grab it
  org_id = org_list.key(org_name)
  puts "Organization \"#{org_name}\" exists"
end

# Get list of organization's lifecycle environments
envs = get_json("#{katello_url}/organizations/#{org_id}/environments")
env_list = id_name_map(envs['results'])
prior_env_id = env_list.key("Library")

# Exit the script if at least one life cycle environment already exists
environments.each do |e|
  if env_list.has_value?(e)
    puts "ERROR: One of the Environments is not unique to organization"
    exit
  end
end

 # Create life cycle environments
environments.each do |environment|
  puts "Creating environment: \t#{environment}"
  prior_env_id = post_json("#{katello_url}/organizations/#{org_id}/environments",
JSON.generate({"name" => environment, "organization_id" => org_id, "prior_id" => prior_env_id}))
["id"]
end
```

5.2.2. Using apipie bindings with Ruby

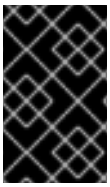
Apipie bindings are the Ruby bindings for apipie documented API calls. They fetch and cache the API definition from Satellite and then generate API calls as needed.

```
#!/usr/bin/ruby

require 'apipie-bindings'

org_name = "_My_Organization_"
environments = [ "Development", "Testing", "Production" ]

# Create an instance of apipie bindings
@api = ApipieBindings::API.new({
  :uri => 'https://satellite.example.com/',
  :username => 'admin',
  :password => 'changeme',
  :api_version => 2
})

# Performs an API call with default options
def call_api(resource_name, action_name, params = {})
  http_headers = {}
  apipie_options = { :skip_validation => true }
  @api.resource(resource_name).call(action_name, params, http_headers, apipie_options)
end

# Creates a hash with IDs mapping to names for an array of records
```

```
def id_name_map(records)
  records.inject({}) do |map, record|
    map.update(record['id'] => record['name'])
  end
end

# Get list of existing organizations
orgs = call_api(:organizations, :index)
org_list = id_name_map(orgs['results'])

if !org_list.has_value?(org_name)
  # If our organization is not found, create it
  puts "Creating organization: \t#{org_name}"
  org_id = call_api(:organizations, :create, {'organization' => { :name => org_name }})['id']
else
  # Our organization exists, so let's grab it
  org_id = org_list.key(org_name)
  puts "Organization \"#{org_name}\" exists"
end

# Get list of organization's life cycle environments
envs = call_api(:lifecycle_environments, :index, {'organization_id' => org_id})
env_list = id_name_map(envs['results'])
prior_env_id = env_list.key("Library")

# Exit the script if at least one life cycle environment already exists
environments.each do |e|
  if env_list.has_value?(e)
    puts "ERROR: One of the Environments is not unique to organization"
    exit
  end
end

 # Create life cycle environments
environments.each do |environment|
  puts "Creating environment: \t#{environment}"
  prior_env_id = call_api(:lifecycle_environments, :create, {"name" => environment, "organization_id"
=> org_id, "prior_id" => prior_env_id })['id']
end
```

5.3. CALLING THE API IN PYTHON

You can use Python with the Satellite API to perform various tasks.



IMPORTANT

These are example scripts and commands. Ensure you review these scripts carefully before use, and replace any variables, user names, passwords, and other information to suit your own deployment.

Example scripts in this section do not use SSL verification for interacting with the REST API.

5.3.1. Creating objects by using Python

This script connects to the Red Hat Satellite 6 API, creates an organization, and then creates three environments in the organization. If the organization already exists, the script uses that organization. If any of the environments already exist in the organization, the script raises an error and quits.

```
#!/usr/bin/python3

import json
import sys

try:
    import requests
except ImportError:
    print("Please install the python-requests module.")
    sys.exit(-1)

# URL to your Satellite Server
URL = "https://satellite.example.com"
FOREMAN_API = f"{URL}/api/"
KATELLO_API = f"{URL}/katello/api/"
POST_HEADERS = {'content-type': 'application/json'}
# Default credentials to login to Satellite 6
USERNAME = "admin"
PASSWORD = "changeme"
# Ignore SSL for now
SSL_VERIFY = False

# Name of the organization to be either created or used
ORG_NAME = "MyOrg"
# Name for life cycle environments to be either created or used
ENVIRONMENTS = ["Development", "Testing", "Production"]


def get_json(location):
    """
    Performs a GET by using the passed URL location
    """
    r = requests.get(location, auth=(USERNAME, PASSWORD), verify=SSL_VERIFY)
    return r.json()


def post_json(location, json_data):
    """
    Performs a POST and passes the data to the URL location
    """
    result = requests.post(
        location,
        data=json_data,
        auth=(USERNAME, PASSWORD),
        verify=SSL_VERIFY,
        headers=POST_HEADERS
    )
    return result.json()


def main():
    """
```

```
    Main routine that creates or re-uses an organization and
    life cycle environments.
    If life cycle environments already
    exist, exit out.
    """

    # Check if our organization already exists
    org = get_json(f"{FOREMAN_API}/organizations/{ORG_NAME}")

    # If our organization is not found, create it
    if org.get('error', None):
        org_id = post_json(
            f"{FOREMAN_API}/organizations/",
            json.dumps({"name": ORG_NAME})
        )["id"]
        print("Creating organization:\t" + ORG_NAME)
    else:
        # Our organization exists, so let's grab it
        org_id = org['id']
        print(f"Organization '{ORG_NAME}' exists.")

    # Now, let's fetch all available life cycle environments for this org...
    envs = get_json(
        f"{KATELLO_API}/organizations/{org_id}/environments/"
    )

    # ...and add them to a dictionary, with respective 'Prior' environment
    prior_env_id = 0
    env_list = {}
    for env in envs['results']:
        env_list[env['id']] = env['name']
        prior_env_id = env['id'] if env['name'] == "Library" else prior_env_id

    # Exit the script if at least one life cycle environment already exists
    if all(environment in env_list.values() for environment in ENVIRONMENTS):
        print("ERROR: One of the Environments is not unique to organization")
        sys.exit(-1)

    # Create life cycle environments
    for environment in ENVIRONMENTS:
        new_env_id = post_json(
            f"{KATELLO_API}/organizations/{org_id}/environments/",
            json.dumps({
                "name": environment,
                "organization_id": org_id,
                "prior": prior_env_id
            })
        )["id"]

        print("Creating environment:\t" + environment)
        prior_env_id = new_env_id


if __name__ == "__main__":
    main()
```

5.3.2. Retrieving resource information by using Python

This is an example script that uses Python for various API requests.

```
#!/usr/bin/env python3

import json
import sys

try:
    import requests
except ImportError:
    print("Please install the python-requests module.")
    sys.exit(-1)

HOSTNAME = "satellite.example.com"
# URL for the API to your Satellite Server
FOREMAN_API = f"https://{HOSTNAME}/api/"
KATELLO_API = f"https://{HOSTNAME}/katello/api/v2/"

POST_HEADERS = {'content-type': 'application/json'}
# Default credentials to login to Satellite 6
USERNAME = "admin"
PASSWORD = "password"
# Ignore SSL for now
SSL_VERIFY = False
#SSL_VERIFY = "./path/to/CA-certificate.crt" # Put the path to your CA certificate here to allow
SSL_VERIFY


def get_json(url):
    # Performs a GET by using the passed URL location
    r = requests.get(url, auth=(USERNAME, PASSWORD), verify=SSL_VERIFY)
    return r.json()

def get_results(url):
    jsn = get_json(url)
    if jsn.get('error'):
        print("Error: " + jsn['error']['message'])
    else:
        if jsn.get('results'):
            return jsn['results']
        elif 'results' not in jsn:
            return jsn
        else:
            print("No results found")
    return None

def display_all_results(url):
    results = get_results(url)
    if results:
        print(json.dumps(results, indent=4, sort_keys=True))

def display_info_for_hosts(url):
    hosts = get_results(url)
    if hosts:
```

```
        print(f"{'ID':10}{'Name':40}{'IP':30}{'Operating System':30}")
        for host in hosts:
            print(f"{str(host['id']):10}{host['name']:40}{str(host['ip']):30}
{str(host['operatingsystem_name']):30}")

def display_info_for_subs(url):
    subs = get_results(url)
    if subs:
        print(f"{'ID':10}{'Name':90}{'Start Date':30}")
        for sub in subs:
            print(f"{str(sub['id']):10}{sub['name']:90}{str(sub['start_date']):30}")

def main():
    host = HOSTNAME
    print(f"Displaying all info for host {host} ...")
    display_all_results(FOREMAN_API + 'hosts/' + host)

    print(f"Displaying all facts for host {host} ...")
    display_all_results(FOREMAN_API + f'hosts/{host}/facts')

    host_pattern = 'example'
    print(f"Displaying basic info for hosts matching pattern '{host_pattern}'...")
    display_info_for_hosts(FOREMAN_API + 'hosts?per_page=1&search=name~' + host_pattern)

    print(f"Displaying basic info for subscriptions")
    display_info_for_subs(KATELLO_API + 'subscriptions')

    environment = 'production'
    print(f"Displaying basic info for hosts in environment {environment}...")
    display_info_for_hosts(FOREMAN_API + 'hosts?search=environment=' + environment)


if __name__ == "__main__":
    main()
```

CHAPTER 6. API CHEAT SHEET

You can review the following examples of how to use the Red Hat Satellite API to perform various tasks. You can use the API on Satellite Server via HTTPS on port 443.

For example, in Ruby, you can specify the Satellite Server URL as follows:

```
url = 'https://satellite.example.com/api/v2/'
katello_url = 'https://satellite.example.com/katello/api/v2/'
```

You can use these values to fully automate your scripts, removing any need to verify which ports to use.

The following examples use **curl** for sending API requests. For more information, see Section 5.1, “Calling the API in curl”.

6.1. WORKING WITH HOSTS

6.1.1. Listing hosts

This example returns a list of registered hosts.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts \
| python3 -m json.tool
```

API response

```
{
      ...
      "total" => 2,
   "subtotal" => 2,
       "page" => 1,
   "per_page" => 1000,
     "search" => nil,
       "sort" => {
          "by" => nil,
       "order" => nil
   },
    "results" => [
      ...
}
```

6.1.2. Requesting information for a host

This request returns information for the host **satellite.example.com**.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts/satellite.example.com \
| python3 -m json.tool
```

API response

```
{
    "all_puppetclasses": [],
    "architecture_id": 1,
    "architecture_name": "x86_64",
    "build": false,
    "capabilities": [
        "build"
    ],
    "certname": "satellite.example.com",
    "comment": null,
    "compute_profile_id": null,
    ...
}
```

6.1.3. Listing host facts

This request returns all facts for the host **satellite.example.com**.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts/satellite.example.com/facts \
| python3 -m json.tool
```

API response

```
{
    ...
    "results": {
        "satellite.example.com": {
            "augeasversion": "1.0.0",
            "bios_release_date": "01/01/2007",
            "bios_version": "0.5.1",
            "blockdevice_sr0_size": "1073741312",
            "facterversion": "1.7.6",
            ...
}
```

6.1.4. Searching for hosts with matching patterns

This query returns all hosts that match the pattern "example".

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts?search=example \
| python3 -m json.tool
```

API response

```
{
    ...
    "results": [
        {
            "name": "satellite.example.com",
            ...
        }
    ],
    "search": "example",
    ...
}
```

6.1.5. Searching for hosts in an environment

This query returns all hosts in the **production** environment.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts?search=environment=production \
| python3 -m json.tool
```

API response

```
{
    ...
    "results": [
        {
            "environment_name": "production",
            "name": "satellite.example.com",
            ...
        }
    ],
    "search": "environment=production",
    ...
}
```

6.1.6. Searching for hosts with a specific fact value

This query returns all hosts with a model name **RHV Hypervisor**.

API request

-

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts?search=model=\"RHV+Hypervisor\" \
| python3 -m json.tool
```

API response

```
{
    ...
    "results": [
        {
            "model_id": 1,
            "model_name": "RHV Hypervisor",
            "name": "satellite.example.com",
            ...
        }
    ],
    "search": "model=\"RHV Hypervisor\"",
    ...
}
```

6.1.7. Deleting a host

This request deletes a host with a name *host1.example.com*.

API request

```
$ curl \
--request DELETE \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts/host1.example.com \
| python3 -m json.tool
```

6.1.8. Downloading a full-host boot disk image

This request downloads a full boot disk image for a host by its ID.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
--output My_Image.iso \
https://satellite.example.com/api/bootdisk/hosts/host_ID?full=true
```

6.2. WORKING WITH LIFECYCLE ENVIRONMENTS

Satellite divides application life cycles into lifecycle environments, which represent each stage of the application life cycle. Lifecycle environments are linked to from an environment path. To create linked lifecycle environments with the API, use the **prior_id** parameter.

You can find the built-in API reference for lifecycle environments at **https://*satellite.example.com*/apidoc/v2/lifecycle_environments.html**. The API routes include **/katello/api/environments** and **/katello/api/organizations/:organization_id/environments**.

6.2.1. Listing lifecycle environments

Use this API call to list all the current lifecycle environments on your Satellite for the default organization with ID **1**.

API request

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/katello/api/organizations/1/environments \
| python3 -m json.tool`
```

API response

```
    output omitted
  "description": null,
  "id": 1,
  "label": "Library",
  "library": true,
  "name": "Library",
  "organization": {
      "id": 1,
      "label": "Default_Organization",
      "name": "Default Organization"
  },
  "permissions": {
      "destroy_lifecycle_environments": false,
      "edit_lifecycle_environments": true,
      "promote_or_remove_content_views_to_environments": true,
      "view_lifecycle_environments": true
  },
  "prior": null,
  "successor": null,
  output truncated
```

6.2.2. Creating linked lifecycle environments

Use this example to create a path of lifecycle environments. This procedure uses the default Library environment with ID **1** as the starting point for creating lifecycle environments.

API procedure

1. Choose an existing lifecycle environment that you want to use as a starting point. List the environment by using its ID. In this case, the environment with ID **1**:
   Example request:

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/katello/api/environments/1 \
| python3 -m json.tool
```

Example response:

```
  output omitted
   "id": 1,
   "label": "Library",
 output omitted
    "prior": null,
    "successor": null,
  output truncated
```

2. Create a JSON file, for example, **life-cycle.json**, with the following content:

```
{"organization_id":1,"label":"api-dev","name":"API Development","prior":1}
```

3. Create a lifecycle environment by using the **prior** option set to **1**.
Example request:

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request POST \
--user My_User_Name:My_Password \
--data @life-cycle.json \
https://satellite.example.com/katello/api/environments \
| python3 -m json.tool
```

Example response:

```
    output omitted
  "description": null,
  "id": 2,
  "label": "api-dev",
  "library": false,
  "name": "API Development",
  "organization": {
      "id": 1,
      "label": "Default_Organization",
      "name": "Default Organization"
  },
  "permissions": {
      "destroy_lifecycle_environments": true,
      "edit_lifecycle_environments": true,
      "promote_or_remove_content_views_to_environments": true,
      "view_lifecycle_environments": true
  },
 "prior": {
      "id": 1,
```

```
        "name": "Library"
    },
    output truncated
```

In the command output, you can see the ID for this lifecycle environment is **2**, and the lifecycle environment before this one is **1**. Use the lifecycle environment with ID **2** to create a successor to this environment.

4. Edit the previously created **life-cycle.json** file to update the **label**, **name**, and **prior** values.

```
{"organization_id":1,"label":"api-qa","name":"API QA","prior":2}
```

5. Create a lifecycle environment using the **prior** option set to **2**.
Example request:

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request POST \
--user My_User_Name:My_Password \
--data @life-cycle.json \
https://satellite.example.com/katello/api/environments \
| python3 -m json.tool
```

Example response:

```
    output omitted
  "description": null,
  "id": 3,
   "label": "api-qa",
   "library": false,
   "name": "API QA",
   "organization": {
      "id": 1,
      "label": "Default_Organization",
      "name": "Default Organization"
   },
   "permissions": {
      "destroy_lifecycle_environments": true,
      "edit_lifecycle_environments": true,
      "promote_or_remove_content_views_to_environments": true,
      "view_lifecycle_environments": true
   },
  "prior": {
      "id": 2,
      "name": "API Development"
   },
   "successor": null,
   output truncated
```

In the command output, you can see the ID for this lifecycle environment is **3**, and the lifecycle environment before this one is **2**.

6.2.3. Updating a lifecycle environment

You can update a lifecycle environment using a PUT command. This example request updates a description of the lifecycle environment with ID **3**.

API request

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request POST \
--user My_User_Name:My_Password \
--data '{"description":"Quality Acceptance Testing"}' \
https://satellite.example.com/katello/api/environments/3 \
| python3 -m json.tool
```

API response

```
  output omitted
"description": "Quality Acceptance Testing",
"id": 3,
"label": "api-qa",
"library": false,
"name": "API QA",
"organization": {
    "id": 1,
    "label": "Default_Organization",
    "name": "Default Organization"
},
"permissions": {
    "destroy_lifecycle_environments": true,
    "edit_lifecycle_environments": true,
    "promote_or_remove_content_views_to_environments": true,
    "view_lifecycle_environments": true
},
"prior": {
    "id": 2,
    "name": "API Development"
},
output truncated
```

6.2.4. Deleting a lifecycle environment

You can delete a lifecycle environment if it has no successor. Therefore, delete them in reverse order using a command in the following format:

API request

```
$ curl \
--request DELETE \
--user My_User_Name:My_Password \
https://satellite.example.com/katello/api/environments/:id
```

6.3. UPLOADING CONTENT TO SATELLITE SERVER

You can use the Satellite API to upload and import large files to your Satellite Server. This process involves four steps:

1. Create an upload request.
2. Upload the content.
3. Import the content.
4. Delete the upload request.

The maximum file size that you can upload is 2 MB. For information about uploading larger content, see API procedure.

Procedure

1. Assign the package name to the variable **name**:
   Example request:

```
$ export name=jq-1.6-2.el7.x86_64.rpm
```

2. Assign the checksum of the file to the variable **checksum**:
   Example request:

```
$ export checksum=$(sha256sum $name|cut -c 1-65)
```

3. Assign the file size to the variable **size**:
   Example request:

```
$ export size=$(du -bs $name|cut -f 1)
```

4. Create an upload request that returns the upload ID of the request by using **size** and **checksum**.
   Example request:

```
$ curl \
--header 'Content-Type: application/json' \
--request POST \
--user My_User_Name:My_Password \
--data "{\"size\": \"$size\", \"checksum\":\"$checksum\"}" \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads
```

   where 76, in this case, is an example Repository ID.

   Example request:

```
{"upload_id":"37eb5900-597e-4ac3-9bc5-2250c302fdc4"}
```

5. Assign the upload ID to the variable **upload_id**:

```
$ export upload_id=37eb5900-597e-4ac3-9bc5-2250c302fdc4
```

6. Assign the path of the package you want to upload to the variable **path**:

```
$ export path=/root/jq/jq-1.6-2.el7.x86_64.rpm
```

7. Upload your content. Ensure you use the correct MIME type when you upload data. The API uses the **application/json** MIME type for the requests to Satellite unless stated otherwise. Combine the upload ID, MIME type, and other parameters to upload content.
Example request:

```
$ curl \
--user My_User_Name:My_Password \
--header Accept:application/json \
--header Content-Type:multipart/form-data \
--request PUT \
--data-urlencode size=$size \
--data-urlencode offset=0 \
--data-urlencode content@${path} \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id
```

8. After you have uploaded the content to your Satellite Server, you need to import it into the appropriate repository. Until you complete this step, Satellite Server does not detect the new content.
Example request:

```
$ curl \
--header "Content-Type:application/json" \
--request PUT \
--user My_User_Name:My_Password \
--data "{\"uploads\":[{\"id\": \"$upload_id\", \"name\": \"$name\", \"checksum\": \"$checksum\"
}]}" \
https://satellite.example.com/katello/api/v2/repositories/76/import_uploads
```

9. After you have successfully uploaded and imported your content, you can delete the upload request. This frees any temporary disk space that data is using during the upload.
Example request:

```
$ curl \
--header 'Content-Type: application/json' \
--request DELETE \
--user My_User_Name:My_Password \
--data "{}" \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id
```

6.3.1. Uploading content larger than 2 MB

The following example demonstrates how to split a large file into chunks, create an upload request, upload the individual files, import them to Satellite, and then delete the upload request. Note that this example uses sample content, host names, user names, repository ID, and file names.

API procedure

1. Assign the package name to the variable **name**:

```
$ export name=bpftool-3.10.0-1160.2.1.el7.centos.plus.x86_64.rpm
```

2. Assign the checksum of the file to the variable **checksum**:

```
$ export checksum=$(sha256sum $name|cut -c 1-65)
```

3. Assign the file size to the variable **size**:

```
$ export size=$(du -bs $name|cut -f 1)
```

4. The following command creates a new upload request and returns the upload ID of the request using **size** and **checksum**.
   Example request:

```
$ curl \
--header 'Content-Type: application/json' \
--request POST \
--user My_User_Name:My_Password \
--data "{\"size\": \"$size\", \"checksum\":\"$checksum\"}" \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads
```

   where 76, in this case, is an example Repository ID.

   Example output

```
{"upload_id":"37eb5900-597e-4ac3-9bc5-2250c302fdc4"}
```

5. Assign the upload ID to the variable **upload_id**:

```
$ export upload_id=37eb5900-597e-4ac3-9bc5-2250c302fdc4
```

6. Split the file in 2MB chunks:

```
$ split \
--bytes 2MB \
--numeric-suffixes \
--suffix-length=1 \
bpftool-3.10.0-1160.2.1.el7.centos.plus.x86_64.rpm bpftool
```

   View the file chunks:

```
$ ls -l bpftool[0-9]
```

   Example output:

```
-rw-r--r--. 1 root root 2000000 Mar 31 14:15 bpftool0
-rw-r--r--. 1 root root 2000000 Mar 31 14:15 bpftool1
-rw-r--r--. 1 root root 2000000 Mar 31 14:15 bpftool2
-rw-r--r--. 1 root root 2000000 Mar 31 14:15 bpftool3
-rw-r--r--. 1 root root  868648 Mar 31 14:15 bpftool4
```

7. Assign the prefix of the split files to the variable path.

```
$ export path=/root/tmp/bpftool
```

8. Upload the file chunks. The offset starts at 0 bytes for the first chunk and increases by 2000000 bytes for each file. Note the use of the offset parameter and how it relates to the file size. Note also that the indexes are used after the path variable, for example, ${path}0, ${path}1. Example requests:

```
$ curl \
--user My_User_Name:My_Password \
--header Accept:application/json \
--header Content-Type:multipart/form-data \
--request PUT \
--data-urlencode size=$size \
--data-urlencode offset=0 \
--data-urlencode content@${path}0 \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id

$ curl \
--user My_User_Name:My_Password \
--header Accept:application/json \
--header Content-Type:multipart/form-data \
--request PUT \
--data-urlencode size=$size \
--data-urlencode offset=2000000 \
--data-urlencode content@${path}1 \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id

$ curl \
--user My_User_Name:My_Password \
--header Accept:application/json \
--header Content-Type:multipart/form-data \
--request PUT \
--data-urlencode size=$size \
--data-urlencode offset=4000000 \
--data-urlencode content@${path}2 \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id

$ curl \
--user My_User_Name:My_Password \
--header Accept:application/json \
--header Content-Type:multipart/form-data \
--request PUT \
--data-urlencode size=$size \
--data-urlencode offset=6000000
--data-urlencode content@${path}3 \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id

$ curl \
--user My_User_Name:My_Password \
--header Accept:application/json \
--header Content-Type:multipart/form-data \
--request PUT \
--data-urlencode size=$size \
--data-urlencode offset=8000000 \
--data-urlencode content@${path}4 \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id
```

9. Import the complete upload to the repository:

```
$ curl \
--header "Content-Type:application/json" \
--request PUT \
--user My_User_Name:My_Password \
--data "{\"uploads\":[{\"id\": \"$upload_id\", \"name\": \"$name\", \"checksum\": \"$checksum\"
}]}" \
https://satellite.example.com/katello/api/v2/repositories/76/import_uploads
```

10. Delete the upload request:

```
$ curl \
--header 'Content-Type: application/json' \
--request DELETE \
--user My_User_Name:My_Password \
--data "{}" \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads/$upload_id
```

6.3.2. Uploading duplicate content

You can reuse existing content in Satellite instead of uploading duplicate content to Satellite through the API.

API procedure

- Upload content to Satellite:

```
$ curl \
--header 'Content-Type: application/json' \
--request POST \
--user My_User_Name:My_Password \
--data "{\"size\": \"$size\", \"checksum\":\"$checksum\"}" \
https://satellite.example.com/katello/api/v2/repositories/76/content_uploads
```

The call will return a content unit ID instead of an upload ID, similar to this:

```
{"content_unit_href":"/pulp/api/v3/content/file/files/c1bcdfb8-d840-4604-845e-
86e82454c747/"}
```

You can copy this output and call import uploads directly to add the content to a repository:

API response

```
$ curl \
--header "Content-Type:application/json" \
--request PUT \
--user My_User_Name:My_Password \
--data "{\"uploads\":[{\"content_unit_id\": \"/pulp/api/v3/content/file/files/c1bcdfb8-d840-4604-845e-
86e82454c747/\", \"name\": \"$name\", \ \"checksum\": \"$checksum\" }]}" \
https://satellite.example.com/katello/api/v2/repositories/76/import_uploads
```

Note that the call changes from using **upload_id** to using **content_unit_id**.

6.4. USING EXTENDED SEARCHES

You can find search parameters that you can use to build your search queries in the Satellite web UI. For more information, see Building search queries in *Administering Red Hat Satellite*.

For example, you can search for hosts.

Procedure

1. In the Satellite web UI, navigate to **Hosts** > **All Hosts**.
2. Click the **Search** field to display a list of search parameters.
3. Locate the search parameters that you want to use. For this example, locate **os_title** and **model**.
4. Combine the search parameters in your API query as follows:
   Example request:

```
$ curl \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/hosts?
search=os_title=\"RedHat+7.7\",model=\"PowerEdge+R330\" \
| python3 -m json.tool
```

   Example response:

```
{
  ...
  "results": [
      {
          "model_id": 1,
          "model_name": "PowerEdge R330",
          "name": "satellite.example.com",
          "operatingsystem_id": 1,
          "operatingsystem_name": "RedHat 7.7",
          ...
      }
  ],
  "search": "os_title=\"RedHat 7.7\",model=\"PowerEdge R330\"",
  "subtotal": 1,
  "total": 11
}
```

6.5. USING SEARCHES WITH PAGINATION CONTROL

You can use the **per_page** and **page** pagination parameters to limit the search results that an API search query returns. The **per_page** parameter specifies the number of results per page and the **page** parameter specifies which page, as calculated by the **per_page** parameter, to return.

The default number of items to return is set to 1000 when you do not specify any pagination parameters, but the **per_page** value has a default of 20 which applies when you specify the **page** parameter.

This example returns a list of activation keys for an organization with ID **1** in pages. The list contains 30 keys per page and returns the second page.

API request

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/katello/api/activation_keys?
organization_id=1&amp;per_page=30&amp;page=2
```

6.5.1. Returning multiple pages

You can use a **for** loop structure to get multiple pages of results. This example returns pages 1 to 3 of Content Views with 5 results per page.

Bash script

```
$ for i in seq 1 3; do \
  curl \
  --request GET \
  --user My_User_Name:My_Password \
  https://satellite.example.com/katello/api/content_views?per_page=5&amp;page=$i; \
done
```

6.6. OVERRIDING SMART CLASS PARAMETERS

You can search for Smart Parameters by using the API and supply a value to override a Smart Parameter in a Class. You can find the full list of attributes that you can modify in the built-in API reference at **https://*satellite.example.com*/apidoc/v2/smart_class_parameters/update.html**.

Procedure

1. Find the ID of the Smart Class parameter you want to change:

   - List all Smart Class Parameters.
     Example request:

     ```
     $ curl \
     --request GET \
     --user My_User_Name:My_Password \
     https://satellite.example.com/api/smart_class_parameters
     ```

   - If you know the Puppet class ID, for example 5, you can restrict the scope: Example request:

     ```
     $ curl \
     --request GET \
     --user My_User_Name:My_Password \
     https://satellite.example.com/api/puppetclasses/5/smart_class_parameters
     ```

     Both calls accept a search parameter. You can view the full list of searchable fields in the Satellite web UI. Navigate to **Configure** > **Smart variables** and click in the search query box to reveal the list of fields.

Two particularly useful search parameters are **puppetclass_name** and **key**, which you can use to search for a specific parameter. For example, use the **--data** option to pass URL encoded data.

Example request:

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
--data 'search=puppetclass_name = access_insights_client and key = authmethod' \
https://satellite.example.com/api/smart_class_parameters
```

Satellite supports standard scoped-search syntax.

2. When you find the ID of the parameter, list the full details including current override values.
Example request:

```
$ curl \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/smart_class_parameters/63
```

3. Enable overriding of parameter values.
Example request:

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request PUT \
--user My_User_Name:My_Password \
--data '{"smart_class_parameter":{"override":true}}' \
https://satellite.example.com/api/smart_class_parameters/63
```

Note that you cannot create or delete the parameters manually. You can only modify their attributes. Satellite creates and deletes parameters only upon class import from Capsules.

4. Add custom override matchers.
Example request:

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request PUT \
--user My_User_Name:My_Password \
--data '{"smart_class_parameter":{"override_value":
{"match":"hostgroup=Test","value":"2.4.6"}}}' \
https://satellite.example.com/api/smart_class_parameters/63
```

For more information about override values, see **https://*satellite.example.com*/apidoc/v2/override_values.html**.

5. You can delete override values.
Example request:

```
$ curl \
--request DELETE \
--user My_User_Name:My_Password \
https://satellite.example.com/api/smart_class_parameters/63/override_values/3
```

6.7. MODIFYING A SMART CLASS PARAMETER BY USING AN EXTERNAL FILE

You can modify a Puppet Smart Class parameter by using an external file.

Using external files simplifies working with JSON data. You can use an editor with syntax highlighting to avoid and locate mistakes. This example uses a MOTD Puppet manifest.

API procedure

1. Search for the Puppet Class by name, **motd** in this case.
   Example request:

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/smart_class_parameters?search=puppetclass_name=motd
\
| python3 -m json.tool
```

2. Examine the following output. Each Smart Class Parameter has an ID that is global for the same Satellite instance. The **content** parameter of the **motd** class has **id=3**. Do not confuse this with the Puppet Class ID that displays before the Puppet Class name.
   Example response:

```
{
 "avoid_duplicates": false,
 "created_at": "2024-02-06 12:37:48 UTC", # Remove this line.
  "default_value": "", # Add a new value here.
  "description": "",
 "hidden_value": "",
 "hidden_value?": false,
 "id": 3,
 "merge_default": false,
 "merge_overrides": false,
 "override": false, # Set the override value to true.
  "override_value_order": "fqdn\nhostgroup\nos\ndomain",
 "override_values": [], # Remove this line.
  "override_values_count": 0,
 "parameter": "content",
 "parameter_type": "string",
 "puppetclass_id": 3,
 "puppetclass_name": "motd",
 "required": false,
 "updated_at": "2024-02-07 11:56:55 UTC", # Remove this line.
  "use_puppet_default": false,
```

```
  "validator_rule": null,
  "validator_type": ""
}
```

3. Use the parameter ID **3** to get the information specific to the **motd** parameter and redirect the output to a file, for example, **output_file.json**.
Example request:

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request GET \
--user My_User_Name:My_Password \
https://satellite.example.com/api/smart_class_parameters/3 \
| python3 -m json.tool > output_file.json
```

4. Copy the file created in the previous step to a new file for editing, for example, **changed_file.json**:

```
$ cp output_file.json changed_file.json
```

5. Modify the required values in the file. In this example, change the content parameter of the **motd** module, which requires changing the **override** option from **false** to **true**:

```
{
 "avoid_duplicates": false,
  "created_at": "2024-02-06 12:37:48 UTC", # Remove this line.
   "default_value": "", # Add a new value here.
   "description": "",
  "hidden_value": "",
  "hidden_value?": false,
  "id": 3,
  "merge_default": false,
  "merge_overrides": false,
  "override": false, # Set the override value to true.
   "override_value_order": "fqdn\nhostgroup\nos\ndomain",
  "override_values": [], # Remove this line.
   "override_values_count": 0,
  "parameter": "content",
  "parameter_type": "string",
  "puppetclass_id": 3,
  "puppetclass_name": "motd",
  "required": false,
  "updated_at": "2024-02-07 11:56:55 UTC", # Remove this line.
   "use_puppet_default": false,
  "validator_rule": null,
  "validator_type": ""
}
```

6. After editing the file, verify that it looks as follows and then save the changes:

```
{
 "avoid_duplicates": false,
  "default_value": "No Unauthorized Access Allowed",
```

```
  "description": "",
  "hidden_value": "",
  "hidden_value?": false,
  "id": 3,
  "merge_default": false,
  "merge_overrides": false,
  "override": true,
  "override_value_order": "fqdn\nhostgroup\nos\ndomain",
  "override_values_count": 0,
  "parameter": "content",
  "parameter_type": "string",
  "puppetclass_id": 3,
  "puppetclass_name": "motd",
  "required": false,
  "use_puppet_default": false,
  "validator_rule": null,
  "validator_type": ""
}
```

7. Submit the file to Satellite:

```
$ curl \
--header "Accept:application/json" \
--header "Content-Type:application/json" \
--request PUT \
--user My_User_Name:My_Password \
--data @changed_file.json \
https://satellite.example.com/api/smart_class_parameters/3
```

6.8. DELETING OPENSCAP REPORTS

In Satellite Server, you can delete one or more OpenSCAP reports. However, when you delete reports, you must delete one page at a time. If you want to delete all OpenSCAP reports, use the bash script that follows.

API Procedure

1. List all OpenSCAP reports. Note the IDs of the reports that you want to delete.
   Example request:

```
$ curl \
--user My_User_Name:My_Password \
https://satellite.example.com/api/v2/compliance/arf_reports/ \
| python3 -m json.tool
```

   Example response:

```
{
    "page": 1,
    "per_page": 20,
    "results": [
        {
            "created_at": "2024-05-16 13:27:09 UTC",
            "failed": 0,
```

```
            "host": "host1.example.com",
            "id": 404,
            "othered": 0,
            "passed": 0,
            "updated_at": "2024-05-16 13:27:09 UTC"
        },
        {
            "created_at": "2024-05-16 13:26:07 UTC",
            "failed": 0,
            "host": "host2.example.com,
            "id": 405,
            "othered": 0,
            "passed": 0,
            "updated_at": "2024-05-16 13:26:07 UTC"
        },
        {
            "created_at": "2024-05-16 13:25:07 UTC",
            "failed": 0,
            "host": "host3.example.com",
            "id": 406,
            "othered": 0,
            "passed": 0,
            "updated_at": "2024-05-16 13:25:07 UTC"
        },
        {
            "created_at": "2024-05-16 13:24:07 UTC",
            "failed": 0,
            "host": "host4.example.com",
            "id": 407,
            "othered": 0,
            "passed": 0,
            "updated_at": "2024-05-16 13:24:07 UTC"
        },
    ],
    "search": null,
    "sort": {
        "by": null,
        "order": null
    },
    "subtotal": 29,
    "total": 29
```


2. Using an ID from the previous step, delete the OpenSCAP report. Repeat for each ID that you want to delete.
   Example request:

```
$ curl \
--user My_User_Name:My_Password \
--header "Content-Type: application/json" \
--request DELETE \
https://satellite.example.com/api/v2/compliance/arf_reports/405
```

   Example response:

```
HTTP/1.1 200 OK
```

```
Date: Thu, 18 May 2024 07:14:36 GMT
Server: Apache/2.4.6 (Red Hat Enterprise Linux)
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Foreman_version: 3.11.0.76
Foreman_api_version: 2
Apipie-Checksum: 2d39dc59aed19120d2359f7515e10d76
Cache-Control: max-age=0, private, must-revalidate
X-Request-Id: f47eb877-35c7-41fe-b866-34274b56c506
X-Runtime: 0.661831
X-Powered-By: Phusion Passenger 4.0.18
Set-Cookie: request_method=DELETE; path=/
Set-Cookie: _session_id=d58fe2649e6788b87f46eabf8a461edd; path=/; secure; HttpOnly
ETag: "2574955fc0afc47cb5394ce95553f428"
Status: 200 OK
Vary: Accept-Encoding
Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
```

Example BASH script to delete all OpenSCAP reports

```
#!/bin/bash

# this script removes all ARF reports from your Satellite Server

# settings
USER="My_User_Name"
PASS="My_Password"
URI="https://satellite.example.com"

# check amount of reports
 while [ $(curl --user $USER:$PASS $URI/api/v2/compliance/arf_reports/ | python3 -m json.tool | grep \"\total\": | cut --fields=2 --delimiter":" | cut --fields=1 --delimiter"," | sed "s/ //g") -gt 0 ]; do

# fetch reports
 for i in $(curl --user $USER:$PASS $URI/api/v2/compliance/arf_reports/ | python3 -m json.tool | grep \"\id\": | cut --fields=2 --delimiter":" | cut --fields=1 --delimiter"," | sed "s/ //g")

# delete reports
  do
  curl --user $USER:$PASS --header "Content-Type: application/json" --request DELETE $URI/api/v2/compliance/arf_reports/$i
  done
done
```

APPENDIX A. API RESPONSE CODES

The Red Hat Satellite 6 API provides HTTP response status codes for API calls. The following codes are common for all resources in the Satellite API.

Table A.1. API response codes

Response	Explanation
200 OK	For a successful request action: show, index, update, or delete (GET, PUT, DELETE requests).
201 Created	For a successful create action (POST request).
301 Moved Permanently	Redirect when Satellite is restricted to use HTTPS and HTTP is attempted.
400 Bad Request	A required parameter is missing or the search query has invalid syntax.
401 Unauthorized	Failed to authorize the user, for example, due to incorrect credentials.
403 Forbidden	The user has insufficient permissions to perform the action or read the resource, or the action is unsupported in general.
404 Not Found	The record with the given ID does not exist. It can appear in show and delete actions when the requested record does not exist; or in create, update and delete actions when one of the associated records does not exist.
409 Conflict	Could not delete the record due to existing dependencies, for example, host groups that still contain hosts.
415 Unsupported Media Type	The content type of the HTTP request is not JSON.
422 Unprocessable Entity	Failed to create an entity due to some validation errors. Applies to create or update actions only.
500 Internal Server Error	Unexpected internal server error.
503 Service Unavailable	The server is not running.

APPENDIX B. CREATING A COMPLETE PERMISSION TABLE

Use the Satellite CLI to create a permission table.

Procedure

1. Start the Satellite console with the following command:

```
# foreman-rake console
```

2. Insert the following code into the console:

```
f = File.open('/tmp/table.html', 'w')

result = Foreman::AccessControl.permissions {|a,b| a.security_block <=>
b.security_block}.collect do |p|
      actions = p.actions.collect { |a| "<li>#{a}</li>" }
      "<tr><td>#{p.name}</td><td><ul>#{actions.join('')}</ul></td><td>#{p.resource_type}</td>
</tr>"
end.join("\n")

f.write(result)
```

   The above syntax creates a table of permissions and saves it to the **/tmp/table.html** file.

3. Press **Ctrl** + **D** to exit the Satellite console.

4. Insert the following text at the first line of **/tmp/table.html**:

```
<table border="1"><tr><td>Permission name</td><td>Actions</td><td>Resource type</td>
</tr>
```

5. Append the following text at the end of **/tmp/table.html**:

```
</table>
```

6. Open **/tmp/table.html** in a web browser to view the table.