Nutanix API User Guide

The v4 API User Guide describes how the new set of v4 APIs represents the future of Nutanix platform automation by using APIs and software development kits (SDKs).

Nutanix Legacy API Deprecation Note Nutanix Legacy API versions v0.8, v1, v2, and v3 will be deprecated and no longer supported starting with the AOS and PC Upgrade Release planned for Q4-CY2026. Customers with active service contracts will continue to receive support from Nutanix as per the Nutanix EOSL policy. Nutanix strongly recommends that all our customers and partners migrate their applications and tools to the latest v4 APIs and SDKs before the legacy APIs are deprecated. See the Nutanix Legacy API Deprecation Notice, available here: https://download.nutanix.com/misc/LegacyAPI-EOLNotification.pdf

Nutanix v4 API User Guide

Table of Contents

Purpose

This v4 API User Guide describes how the new set of v4 APIs represents the future of Nutanix platform automation by using APIs and software development kits (SDKs).

Audience

This document is intended for new and existing Nutanix API users planning to use the Nutanix v4 REST APIs and SDKs.

Related Documentation

For information about Nutanix v4 APIs and included features, see Nutanix v4 API Introduction.

The snippets and pointers in this guide have demonstrated a number of key points. They are intended as a starting point and prepare users to create custom scripts and apps using the Nutanix v4 APIs and SDKs.

The related resources below are recommended for readers:

- Nutanix v4 API migration guide
- Nutanix v4 API Developer Portal
- Nutanix v4 API Versioning Scheme
- Code sample repository

Overview

The Nutanix v4 APIS were originally released as early access in Prism Central pc.2022.6 and have been redesigned from the ground up.

Nutanix APIs have grown over the past decade and now provide support for multiple endpoints, clusters, products, and operations. Our goal is to provide a comprehensive and consistent set of APIs and SDKs for operating the Nutanix Cloud Platform. The v4 API includes semantics based on open standards and delivers enhanced usability and developer experience. Nutanix SDKs provide libraries, documentation and resources to help developers build applications and solutions on the Nutanix Platform.

Automation can take the form of a custom script or application. In these cases, specific steps in a specific order are often required. Depending on the action required, the only way of achieving this level of custom automation is by using an API or an SDK where the user has complete control over every step in the process. From a customer perspective, some typical API and SDK use cases are:

- Retrieve information about an entity owned by a specific system. On the Nutanix Cloud Platform, this includes virtual machines, disk images, storage containers and overall cluster configuration. These are typically GET requests.
- Create an entity on a managed system. On the Nutanix Cloud Platform, this includes the creation of a managed entity such as a disk image, virtual machine or Nutanix Flow Network Security policy. These are typically POST requests.
- Update or change a managed entity. On the Nutanix Cloud Platform, this includes the modification of cluster configuration and alteration of a managed entity's configuration such as a disk image or virtual machine. These are typically PUT requests.
- Language-specific SDKs for integration of the Nutanix Cloud Platform into an existing environment or the integration of a third party system into the Nutanix Cloud Platform.

Nutanix v4 API and SDK Features

Note: For the purposes of this user guide, Python 3 will be used as the basis for all code samples.

The Nutanix v4 APIs include these significant features.

- **Language-specific SDKs**. The officially-supported languages are Python, Go, Java and JavaScript.
- **OData-compliant resource filtering**. Provides the ability to quickly find a specific resource instance without the need to send multiple requests that then require client-side processing.
- **Result sorting**. Provides the ability to specify a sort key during list requests. For example, a list of Prism Central disk images could be sorted by size (bytes).
- Improved exception handling with more meaningful error messages, along with a detailed per-namespace error code reference.
- A new developer portal covering all supported languages, SDKs and REST API endpoints.
- API versioning support and a defined versioning scheme.
- **Request idempotency via Ntnx-Request-Id header**. Provides support for request idempotency i.e. the ability to ensure a request is not duplicated or run more than once. This is done by providing a UUID-formatted header as Ntnx-Request-Id. Required for REST APIs only; Nutanix SDKs do not require the use of the Nutanix-Request-Id header.
- **Resource Etag via If-Match header**. This ensures mid-air collisions are avoided by providing resource-specific versions. In the event two API consumers modify the same resource, the first request will alter the resource Etag. The next user must request the resource's new Etag and details before submitting their own update request.
- OData Selection Projection. Controls which fields of an entity are displayed in the response. APIs generally return a default set of fields on a GET request. Selection Projection allows the user to specify a subset of fields to be returned, thereby decreasing response time and payload size.
- OData Expansion Projection. Controls which fields of a related entity are returned when requesting another specific entity. For example, listing volume groups using the Volumes namespace accepts an $expand parameter to request the related cluster.
- Batch Operations. Allows users to generate a bulk payload for entity creation, modification, deletion or custom actions. Up to 500 entity actions can be included in a single batch request.
- Rate Limiting. When incoming requests exceed a certain limit, further requests will be temporarily blocked. Rate limits are dictated by the Prism Central type and configuration.

Software Requirements

Recommendations

Where possible, Nutanix recommends using the Nutanix v4 APIs via the new language-specific SDKs, depending on language support. Getting Started describes support languages. The v4 SDKs relieve the user of many manual actions through abstraction, resulting in a more streamlined experience. For example, the SDKs remove the need to manually manage the following actions.

- Connection between clients and Prism Central. When writing custom connection methods, Python developers often do this using the `requests` library.

- Authentication. The Nutanix v4 SDKs remove the need for you to create authentication methods in your code. Python developers might do this using the `HTTPBasicAuth` or `base64.b64encode` libraries.
- Custom exception handling. The v4 SDKs provide meaningful error information in the event of an invalid request or other problem while submitting a request.

In addition, to avoid the manual steps above, the Nutanix v4 SDKs create responses and accept user payloads that are based on fluent, action-relevant information.

For example, the Lifecycle Manager (LCM) SDK action `RecommendationsApi.compute_recommendations` accepts an instance of `RecommendationSpec`. Compared to manually-created JSON payloads, this can be a significantly smoother experience.

However, Nutanix v4 APIs may need to be consumed via HTTP endpoints in situations where an official SDK is not available. When using HTTP endpoints the above actions must be manually managed; this guide demonstrates both approaches.

Request Requirements

The Nutanix v4 Python SDKs are named as follows:

“ntnx__py_client”

Throughout this guide code samples primarily reference the “vmm” namespace “ntnx_vmm_py_client”. The “vmm” namespace exposes operations on entities such as Prism Central virtual machines, images, placement policies and rate limits. Namespaces required for different operations will need to be installed and referenced individually. See https://developers.nutanix.com for information on currently available public Nutanix SDKs.

Getting Started with v4 APIs

As a first step, Nutanix recommends deciding how you will consume the v4 APIs. This will be directly via an HTTP endpoint or language-specific SDK.

Configuring Your Python Development Environment

With Python chosen as the base language for all code samples through this guide, the following steps demonstrate how to configure a demo development environment.

You can skip this section if you are already familiar with Python development environment setup, or if you have a preferred alternative method.

Creating a Virtual Environment with "venv" and Linux or Mac OS X Bash Terminal

```
# verify python binary location and version
```

which python

returns python path e.g. /usr/bin/python

python --version

returns Python version. Python 3 versions 3.6, 3.7, 3.8 and 3.9 are fully tested and supported

create a working directory for demo code

mkdir -p ~/nutanix/api/v4 cd ~/nutanix/api/v4

create and activate a Python 3 virtual environment

alter the python path if necessary

demo virtual environment will be named venv

python -m venv venv . venv/bin/activate

verify python binary location again

this will be different with the virtual environment activated which python

returns python path e.g. /home/username/nutanix/api/v4/venv/bin/python

```

**Create a `pip` requirements.txt file**

It is common practice when using Python to create a special requirements.txt file.  This file specifies the Python libraries and their versions that are required for a script to run correctly.  In your demo directory, create a requirements.txt containing the following specifications.

Note regarding namespace versions This guide shows requirements.txt installing specific package versions that are correct at the time of writing. In the event "pip" shows an error about unavailable versions, check the error message for a list of available versions. For example: ERROR: Could not find a version that satisfies the requirement ntnx_vmm_py_client==4.1.2 (from versions: 4.0.1a1, 4.0.1b1, 4.0.1, 4.0.2a1, 4.0.3a1, 4.1.1) shows 4.1.1 as the latest available VMM GA package version.

This `requirements.txt` example installs all Nutanix v4 API namespaces available at the time of Prism Central and AOS 7.3 release (June 2025).

```

create `requirements.txt` file
```

touch requirements.txt cat < requirements.txt requests==2.32.4 ruff==0.12.1 ntnx_vmm_py_client==4.1.1 ntnx_lifecycle_py_client==4.1.1
ntnx_prism_py_client==4.1.1 ntnx_clustermgmt_py_client==4.1.1 ntnx_aiops_py_client==4.0.1 ntnx_networking_py_client==4.1.1
ntnx_microseg_py_client==4.1.1 ntnx_iam_py_client==4.0.1 ntnx_opsmgmt_py_client==4.0.2 ntnx_datapolicies_py_client==4.1.1
ntnx_dataprotection_py_client==4.1.1 ntnx_files_py_client==4.0.1 ntnx_licensing_py_client==4.1.1 ntnx_microseg_py_client==4.1.1
ntnx_monitoring_py_client==4.1.1 ntnx_objects_py_client==4.0.1 ntnx_security_py_client==4.0.1 ntnx_volumes_py_client==4.1.1 EOF

```

Installing Requirements

With the `requirements.txt` file created as per our requirements, the script dependencies (requirements) can be installed as follows.

```

# install script requirements from `requirements.txt`
```

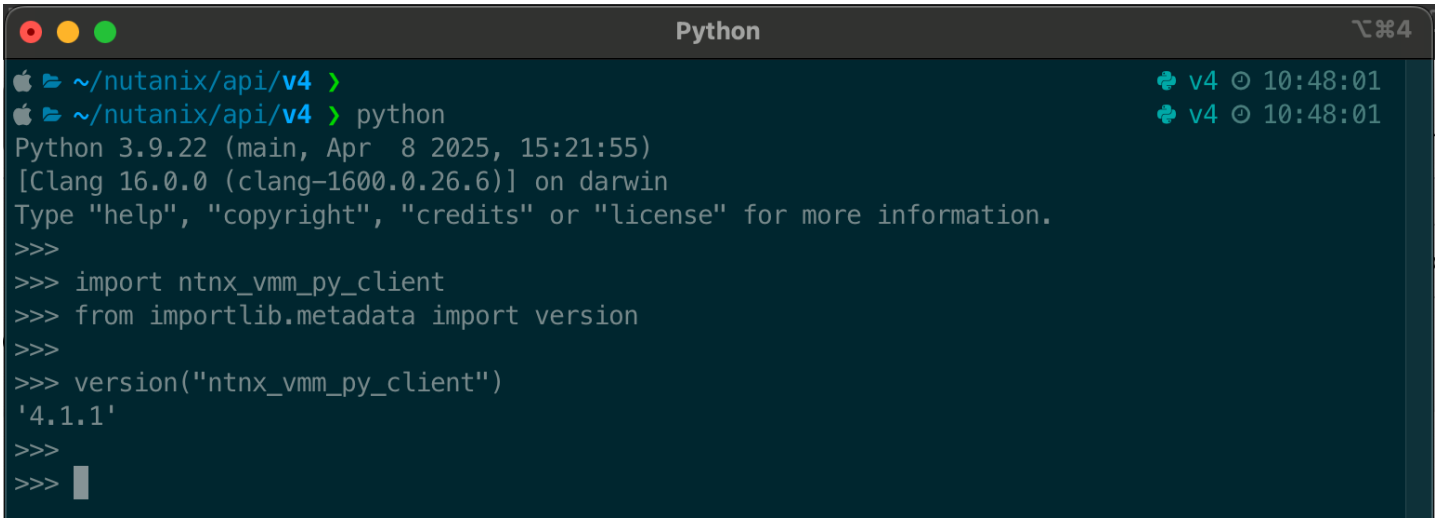
pip install -r requirements.txt --no-cache-dir

```

Note regarding pip install The --no-cache-dir command-line parameter prevents packages being installed from local cache. Without this parameter, pip may install from local cached packages, resulting in unexpected version installation.

**Verifying Installation**

Using the Python REPL (Read, Evaluate, Print, Loop) we can now verify if the Nutanix v4 VMM SDK was installed correctly.  This demo shows the use of Python version 3.10.8 although the supported versions (3.6, 3.7, 3.8 and 3.9) should be used in production environments.




**Preparing Your Python Environment**

To use this guide's Python code snippets, some preparation is required.  The following code snippet is a bootstrap script that can be used for templating your own projects.  The bootstrap script imports the Nutanix v4 SDKs used throughout this guide and instantiates a number of instances that can be used during both SDK and API snippets.

Note: The script snippets shown in this guide do not follow a specific code standard.  The *ruff* library imported earlier can help with code formatting to a particular standard.
```

```
                # common modules

import urllib3

## alter these values to match your environment

## required for both SDKs and APIs

prism_central_ip_address = "0.0.0.0" prism_central_username = "admin" prism_central_password = "password" """ setup testing environment for the Nutanix v4 SDKs """

## import the Nutanix v4 vmm SDK

import ntnx_vmm_py_client

## import the configuration module that manages IP address, username and password information

## to avoid name collisions, each Nutanix v4 SDK module will be imported with dedicated names

from ntnx_vmm_py_client import Configuration as VMMConfiguration

## import the configuration module that manages the API client

## to avoid name collisions, each Nutanix v4 SDK module will be imported with dedicated names

## additional namespaces will need to be imported for different tasks

## see https://developers.nutanix.com for a complete list of available Python SDK namespaces

from ntnx_vmm_py_client import ApiClient as VMMClient

## instantiate our demo environment's SDK configuration

config = VMMConfiguration() config.host = prism_central_ip_address config.username = prism_central_username config.password = prism_central_password

## optional; disable SSL certificate verification and warnings

config.verify_ssl = False urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

## create the client

api_client = VMMClient(configuration=config) """ setup testing environment for the Nutanix v4 SDKs """ import json import requests from requests.auth import HTTPBasicAuth

## setup HTTP basic authentication

prism_central_auth = HTTPBasicAuth(prism_central_username, prism_central_password)
```

Authentication

The Nutanix v4 APIs provide authentication by the following methods.

- HTTP Basic Authentication i.e. username and password
- Can be used with standard user accounts and directory service accounts
- IAM (Identity and Access Management) API key
- Can be used with users of type “Service Account” only
- Intended for use during automated interaction e.g. script or third-party integration

- All API key configuration is completed using Nutanix v4 IAM APIs or SDKs; service accounts cannot be managed or added to authorization policies in the Prism Central user interface. Additionally, API keys cannot be created in the Nutanix Prism Central user interface.

Authentication: API Key Summary

The recommended API key authentication workflow can be summarized as follows.

1. Create a user of type “service account”
2. Create a new API key, associating the key with the new service account user. The new API key is shown once only so must be securely documented or saved during this step.
3. Create an authorization policy outlining the specific permissions required by the service account user, then associated the new service account with the new authorization policy

During subsequent API requests, specify the *X-Ntnx-Api-Key* header and set the header value to the API key generated in step 2.

Examples of using Nutanix v4 API key authentication are available here:

- Using the Nutanix Python SDK
- Using Nutanix v4 REST APIs

Depending on the type of action being performed, the following account type will be required.

- Read actions. These actions typically do not need administrator privileges. For example, retrieving cluster information or VM details.
- Create, update or delete actions usually require specific IAM or administrator permissions. For example, creating a virtual machine or updating an image.

Regardless of the action being performed, Nutanix recommends consuming the Nutanix v4 APIs and SDKs with a dedicated user account versus the built-in “admin” account.

The code snippets below are identical to those shown in the demo bootstrap script above and use HTTP Basic Authentication.

Authentication: Python SDK

The following code snippet demonstrates authentication using the Python SDK library.

```
                # import required libraries, including Nutanix v4 Python VMM SDK
```

import ntnx_vmm_py_client from ntnx_vmm_py_client import ApiClient as VMMClient from ntnx_vmm_py_client import Configuration as VMMConfiguration config = VMMConfiguration() config.host = "prism_central_ip_or_fqdn" config.username = "prism_central_username" config.password = "prism_central_password"

useful during testing, not recommended in production environments

config.verify_ssl = False

```

**Authentication: Python requests Library (HTTP Basic Authentication)**

The following code snippet demonstrates the use of HTTP basic authentication with the Python `requests` library.  This is identical to using HTTP basic authentication via `HTTPBasicAuth` with any other API that supports it.

### Requests

The Nutanix v4 APIs and SDKs offer simple requests for the management of various actions.

HTTP POST, PUT and DELETE requests must be submitted with an accompanying JSON payload.  This payload includes details about the action to be performed.  This JSON payload is an example of creating a simple Prism Central image using the Nutanix v4 REST API “vmm” namespace.

```
                {
```

"name": "rocky10cloud", "type": "DISK_IMAGE", "description": "Rocky Linux 10 Cloud Image from v4 APIs", "source": { "url": "{{rocky10_url}}", "$objectType": "vmm.v4.content.UrlSource" }, "clusterLocationExtIds": [ "{{cluster_uuid}}" ] }

```

For this demo, we’ll build on the previously described snippets and create a request to list all Prism Central images.

Requests: Python SDK (List Images)

This example shows the use of the `async_req=False` parameter. Setting this parameter to `False` means Python will wait for the request to complete before returning to script or command execution.

```

# build on the previous Nutanix v4 SDK code snippet to request a list of Prism Central images
```

api_client = VMMClient(configuration=config) api_instance = ntnx_vmm_py_client.api.ImagesApi(api_client=api_client) images_list = api_instance.list_images(async_req=False)

print the number of found images

print(images_list.metadata.total_available_results)

in our demo cluster this results in the response "11"

the number of images in your response will be different

```

**Requests: Python `requests` Library (List Images)**

Before continuing with the following Python “requests” code samples, ensure you have configured your environment as per “Configuring Your Python Development Environment”.  This includes preparing your credentials for use with HTTP Basic Authentication.

This example shows the use of the Python `requests` library to generate a list of Prism Central images.

```

build on the previous Nutanix v4 SDK code snippets to look at various Images API response types
```

## assumes a list of Prism Central images is available as `images_list`

print(f"Images list is type {type(images_list)}.")

## returns Images list is type .

## assumes an image has been created and the result is available as `image_create`

print(f"Image creation result type is type {type(image_create)}.")

## returns Image creation result type is type .# get an existing image first

## assumes a list of Prism Central images is available as `images_list`

existing_image = api_instance.get_image_by_id(images_list.data[0].ext_id) print(f"Existing image result type is {type(existing_image)}.")

## returns Existing image result type is .

```

Responses

Responses from Nutanix v4 API and SDK requests differ slightly based on the method chosen. At the core of all responses, however, is almost identical data that can be used to either display information or make a decision regarding what the next action will be.

Note: The code snippets in this section are for demonstration purposes only and will require additional steps before use. See “Using Etag and If-Match headers with Nutanix v4 APIs” for more information.

Responses: Python SDK (List Images)

The Nutanix v4 Python SDK will return specific response types depending on the type of request. This code snippet demonstrates a number of different response types, based on the type of request that has been submitted.

```
```

```
# build on the previous Nutanix v4 SDK code snippets to look at various Images API response types
```

assumes a list of Prism Central images is available as `images_list`

print(f"Images list is type {type(images_list)}.")

returns "Images list is type ."

assumes an image has been created and the result is available as `image_create`

print(f"Image creation result type is type {type(image_create)}.")

returns "Image creation result type is type ."

get an existing image first

assumes a list of Prism Central images is available as `images_list`

existing_image = api_instance.get_image_by_id(images_list.data[0].ext_id) print(f"Existing image result type is {type(existing_image)}.")

returns "Existing image result type is type ."

```

From the previous code snippet:

- Getting an existing Prism Central image via Nutanix v4 Python SDK returns an instance of ntnx_vmm_py_client.models.vmm.v4.content.GetImageApiResponse.GetImageApiResponse
- Listing Prism Central images returns ntnx_vmm_py_client.models.vmm.v4.content.ListImagesApiResponse.ListImagesApiResponse
- Creating a Prism Central image returns an instance of ntnx_vmm_py_client.models.vmm.v4.content.CreateImageApiResponse.CreateImageApiResponse

**Responses: Python SDK Images List Parsing**

After using the Nutanix v4 Python SDKs to obtain a list of Prism Central images, the ntnx_vmm_py_client.models.vmm.v4.content.Ntnx.vmm.v4.images.ListImagesApiResponse.ListImagesApiResponse instance can be iterated as follows.

```
for image in images_list.data:
print(f"Image found with name {image.name}.")
```

**Responses: Python `requests` Library (List Images)**

Similar to previous Nutanix APIs, the Nutanix v4 REST APIs will return standards-compliant JSON when used via HTTP endpoint.

The example below shows a request to get the details of a specific Prism Central image.

```
# build on the previous Nutanix v4 Python requests code snippet to get details of a Prism Central image
```

set the Nutanix v4 API URL that will be used for this request

replace image_extid with the ext_id of an image in your environment

endpoint = "https://prism_central_ip_address_or_fqdn:9440/api/vmm/v4.1/content/images/image_extid" image_details = requests.get(endpoint, auth=prism_central_auth, verify=False)

use the json() method to print the image details print(image_details.json())

```

A partial VM images list request is shown below.

```
                {
"data": [
   {
       "$reserved": {
           "$fv": "v4.r1"
       },
       "$objectType": "vmm.v4.content.Image",
       "extId": "e460b4c0-e4e4-4042-8e5b-85590797d51e",
       "sizeBytes": 10737418240,
       "createTime": "2025-06-25T00:42:47.425425Z",
       "lastUpdateTime": "2025-06-25T00:42:47.425425Z",
       "ownerExtId": "f0b701e7-163d-5358-abbc-2edb73b1a38c",
       "name": "rocky10cloud",
       "description": "Rocky Linux 10 Cloud Image from v4 APIs",
       "type": "DISK_IMAGE",
       "clusterLocationExtIds": [
           "0005f2f7-eee7-1995-6145-ac1f6b35fe5e"
       ]
   }
],
"$reserved": {
   "$fv": "v4.r1"
},
"$objectType": "vmm.v4.content.ListImagesApiResponse",
"metadata": {
   "flags": [
       {
           "$reserved": {
               "$fv": "v1.r0"
           },
           "$objectType": "common.v1.config.Flag",
           "name": "hasError",
           "value": false
       },
       {
           "$reserved": {
               "$fv": "v1.r0"
           },
           "$objectType": "common.v1.config.Flag",
           "name": "isPaginated",
           "value": true
       },
       {
           "$reserved": {
               "$fv": "v1.r0"
           },
           "$objectType": "common.v1.config.Flag",
           "name": "isTruncated",
           "value": false
       }
   ],
   "$reserved": {
       "$fv": "v1.r0"
   },
   "$objectType": "common.v1.response.ApiResponseMetadata",
   "links": [
       {
           "$reserved": {
               "$fv": "v1.r0"
           },
           "$objectType": "common.v1.response.ApiLink",
           "href": "https://10.0.0.1:9440/api/vmm/v4.1/content/images?$page=0&$limit=50",
           "rel": "last"
       },
       {
           "href": "https://10.0.0.1:9440/api/vmm/v4.1/content/images?$page=0&$limit=50",
           "rel": "self"
```

```
        },
        {
            "href": "https://10.0.0.1:9440/api/vmm/v4.1/content/images?$page=0&$limit=50",
            "rel": "first"
        }
    ],
    "totalAvailableResults": 8
}

}
```

Responses: Python `requests` Library Images List Parsing

After using the Nutanix v4 REST APIs to obtain a JSON-formatted list of Prism Central images, the `requests.models.Response` instance can be parsed as follows.

```
                # get a list of images, if not already done
image_list = requests.request("GET", url, auth=prism_central_auth, verify=False, timeout=10)
## get information about the type of response print(type(image_list))
## returns class 'requests.models.Response'>
## get information about the type of iterator to use print(type(image_list.json()["data"]))
## returns
## with 'list' type obtained, check for results and print
## the name of the first image
if len(image_list.data) > 0: print(f"Image name: {image_list.data[0].name}")
## returns each image’s name # this list is specific to the user’s Prism Central environment
## Image name: rocky10_cloud - Updated
```

How To Use New v4 API Features

OData Support and Compliance

Introduction

OData (Open Data Protocol) is an ISO/IEC approved OASIS standard that defines a set of best practices for building and consuming RESTful APIs. OData enables REST-based services having resources identified by Uniform Resource Locators (URLs) which provides intuitive and efficient retrieval of data. It allows users to discover and navigate through data sets using standard HTTP calls.

System Query Options

Odata specifications provide system query options which are query string parameters that control the amount and order of the data returned for the resource identified by the URL. The names of all system query options are prefixed with a dollar ($) character.

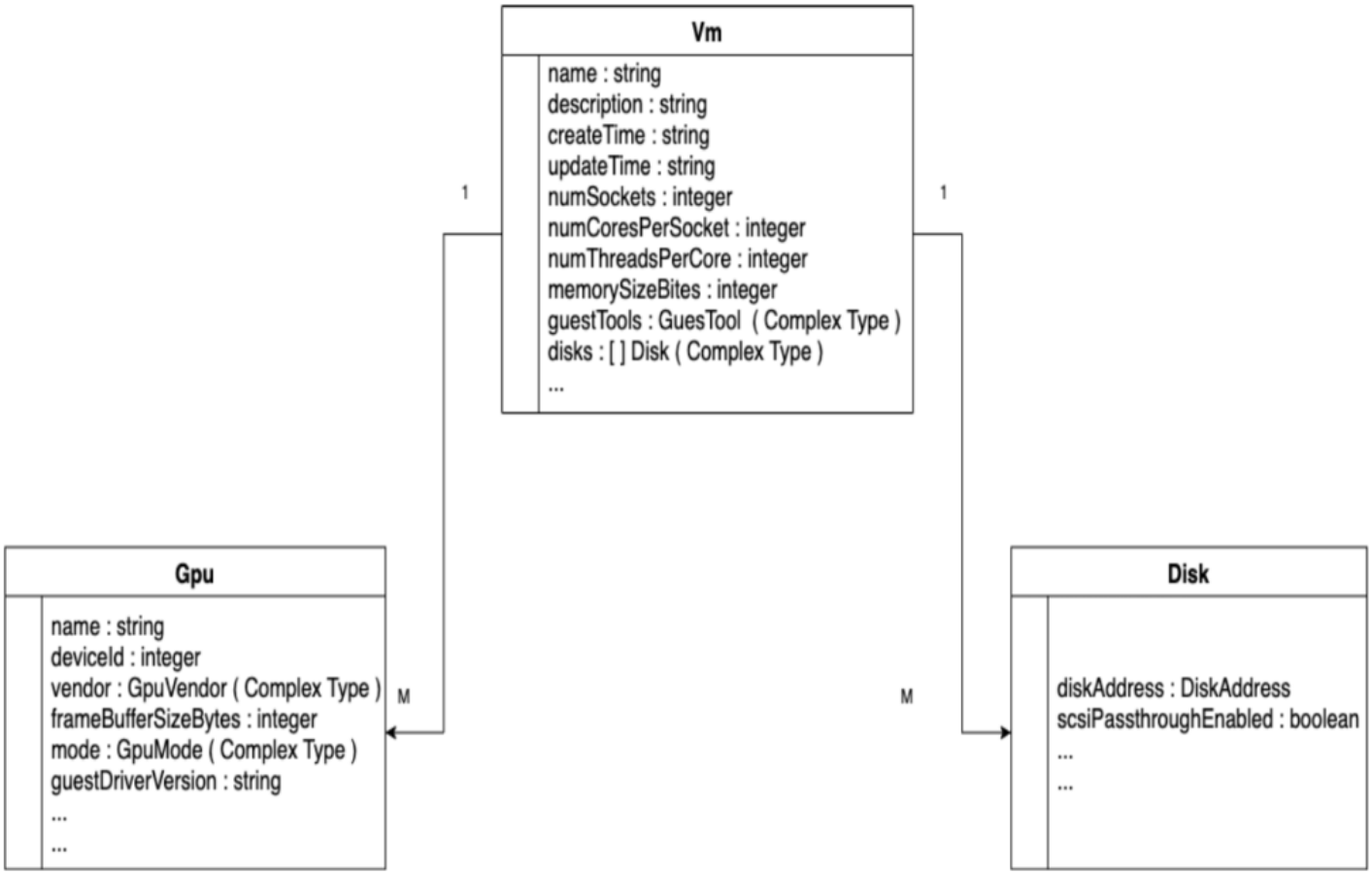
Nutanix v4 APIs support the following system query options.

1. $select
2. $expand
3. $filter
4. $orderby

All the system query options are explained with examples using the following example schema.

The schema has the following models.

- **Vm** : This model comprises attributes which describe a Virtual Machine.
- **Disk** : This model comprises attributes of a disk of a VM.
- **Gpu** : This model comprises attributes of a GPU of a VM.



OASIS Schema Example

Relationship between the models

- VM and disk have One-to-Many relation: a VM can have multiple disks.
- VM and GPU have One-to-Many relation: a VM can have multiple GPUs.

Resource Filtering

The $filter system query option allows clients to filter a collection of resources that are addressed by a request URL. The expression specified with $filter is evaluated for each resource in the collection, and only items where the expression evaluates to true are included in the response. Resources for which the expression evaluates to false or to null, or which reference properties that are unavailable due to permissions, are omitted from the response.

The following operations are supported with $filter.

Logical Operator

| Operation | OData Operator | Examples | | --- | --- | --- | | Equals | eq | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=name eq ‘api_v4’ | | Not Equals | ne | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=name ne ‘api_v4’ | | Greater Than | gt | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=numSockets gt 10 | | Greater Than or Equal | ge | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=numSockets ge 21 | | Less Than | lt | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=numSockets lt 20 | | Less Than or Equal | le | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=numSockets le 22 | | And | and | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=numSockets eq 1 and name eq ‘api_v4’ | | Or | or | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=numSockets eq 1 or numSockets eq 22 | | Not | not | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=not startswith(name, ‘pc’) | | In | in | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=name in (‘test_1’) |

String Method

| Operation | OData Method | Examples | Description | | --- | --- | --- | --- | | Contains | contains | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=contains(name, ‘test’) | Returns VMs in which name value contains the string ‘test’ | | Starts With | startswith | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=startswith(name, ‘a’) | Returns VMs in which the name value starts with the string ‘a’ | | Ends With | endswith | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=endswith(name, ‘1’) | Returns VMs in which the name value ends with the string ‘1’ | | Lower case conversion | tolower | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=tolower(name) eq ‘TESTVM’ | Returns VMs in which the name includes both ‘testvm’ and ‘TESTVM’ | | Upper case conversion | toupper | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$filter=toupper(name) eq ‘testvm’ | Returns VMs in which the name includes both ‘testvm’ and ‘TESTVM’ |

Resource filtering: Lambda Functions

Lambda functions are applied on a property which is of array type. Nutanix v4 APIs support only lambda ‘any’ operator. The “any” operator applies a boolean expression for each member of a collection. It returns true if and only if the expression is true for any member of the collection, otherwise it returns false. A lambda expression is made up of lambda literal which can be any allowed ASCII character.

Consider the following scenario.

- Administrator runs a cluster providing a large number of disk images
- Requirement indicates a specific disk image needs to be found on the cluster in order to get information about that disk image
- Administrator does not know the disk image’s unique identifier (`ext_id`)
- Administrator does know part of the disk image’s name: `fileserver_`

OData filtering allows the administrator to send an image list request containing a filter. That filter can specify the request should only return images matching a certain pattern. In this example, images with a name starting with `fileserver_`.

Resource filtering by name: Python SDK

This script snippet shows a Nutanix v4 SDK image list request including the filter parameter.

Note: This snippet builds on previous snippets and assumes the SDK has already been initialized.

```
# build on the previous Nutanix v4 Python requests code snippet to list images with OData filter
```

api_client = VMMClient(configuration=config) api_instance = ntnx_vmm_py_client.api.ImagesApi(api_client=api_client)

request images list, filtering by name

images_list_with_filter = api_instance.list_images(*filter="startswith(name, 'fileserver'*)")

```

**Resource filtering by name: Python `requests` Library**

The script snippet below shows the same request but using the Python `requests` library.  Note the query string added to the images list HTTP endpoint.

```
build on the previous Nutanix v4 Python requests code snippet to list images with OData filter
```

## set the Nutanix v4 API URL that will be used for this request

endpoint = "https://prism_central_ip_address_or_fqdn:9440/api/vmm/v4.1/content/images?$filter=startswith(name, 'fileserver_')" images_list_with_filter = requests.get(endpoint, auth=prism_central_auth, verify=False)

## use the json() method to print the image details

print(images_list_with_filter.json())

```

Sorting and Limiting

The Nutanix v4 APIs support OData-compliant resource sorting and limiting. The $orderby system query option allows clients to request resources in ascending (default) or descending order.

Consider the following scenario.

- Administrator runs a cluster providing a large number of disk images
- Requirement indicates listing a specific number of images but that list should be sorted by name

OData filtering allows the administrator to send an image list request containing a filter. That filter can specify the request should return a maximum of “n” images, sorted by name.

Sorting and limiting: Python SDK

The script snippet below shows a Nutanix v4 SDK image list request including filter parameters. These filters will order disk images by name and return a maximum of 3 images.

Note: This snippet builds on previous snippets and assumes the SDK has already been initialised.

```
# build on the previous Nutanix v4 Python requests code snippet to list a maximum of 3 images, sorted by name
```

api_client = VMMClient(configuration=config) api_instance = ntnx_vmm_py_client.api.ImagesApi(api_client=api_client)

request images list, sorted by name with a maximum of 3 images

returned images_list_filtered_limited = api_instance.list_images(_orderby="name asc", _limit=3)

show number of images returned by the request

print(f"Images found: {len(images_list_filtered_limited.data)}")

with 'list' type obtained, check for results and print

the name of the first image

if len(image_list.data) > 0: print(f"Image name: {image_list.data[0].name}")

the demo environment returns the following, commented for script safety

Image name: rocky10_cloud.qcow2

```

**Sorting and limiting: Python `requests` Library**

The script snippet below shows the same request but using the Python `requests` library.  Note the query string added to the images list HTTP endpoint.

Note these URL querystring parameters: **?$limit=3&orderby=name**

```
build on the previous Nutanix v4 Python requests code snippet to list a maximum of 3 images, sorted by name
```

## set the Nutanix v4 API URL that will be used for this request

endpoint = "https://prism_central_ip_address_or_fqdn:9440/api/vmm/v4.1/content/images?$limit=3&$orderby=name"

images_list_filtered_limited = requests.get(endpoint, auth=prism_central_auth, verify=False)

## use the json() method to print the image details

print(images_list_filtered_limited.json())

## show how many images were found

print(f"Images found: {len(images_list_filtered_limited.json()['data'])}")

## iterate over the images, showing each name

for image in images_list_filtered_limited.json()["data"]: print(f"Image name: {image['name']}")

## the demo environment returns the following, commented for script safety
```

Image name: rocky10_cloud.qcow2

```

**Selection Projection using $select**

The $select system query option allows clients to request a specific set of properties for each entity or complex type.

Examples:

| Operation | URL | | --- | --- | | Returns results a list of storage container names only | https://{{pc_ip}}:9440/api/clustermgmt/v4.1/config/storage-containers?$select=name | | Returns results which has storage container name and extId | https://{{pc_ip}}:9440/api/clustermgmt/v4.1/config/storage-containers?$select=name,containerExtId | | Returns storage container name, extId and replication factor | https://{{pc_ip}}:9440/api/clustermgmt/v4.1/config/storage-containers?$select=name,containerExtId,replicationFactor |

**Expansion Projection using $expand**

Expansion projections allow related entities to be included inline with a retrieved resource. OData provides an $expand query option to support expansion projections. A request with a $expand query option enables the reading of entries of an entity together with an associated entity.

Examples:

| Operation | URL | | --- | --- | | Returns results a list of volume groups without any expansion projection | https://{{pc_ip}}:9440/api/volumes/v4.1/config/volume-groups | | Returns a list of volume groups including the related cluster projection | https://{{pc_ip}}:9440/api/volumes/v4.1/config/volume-groups?$expand=cluster | | Returns a list of volume groups including the related metadata and cluster projections | https://{{pc_ip}}:9440/api/volumes/v4.1/config/volume-groups?$expand=metadata,cluster |

**Pagination**

The Nutanix v4 APIs have Pagination support.  When exposing large data sets through a list API, pagination is used to provide a mechanism to paginate a list of resources. Pagination can be achieved using:

- $page
- $limit

| Parameter | Type | Description | Example | | --- | --- | --- | --- | | $page | Integer | The page or offset from where resources will be fetched in the list | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$page=2 | | $limit | Integer | The limit of resources to be fetched from the given offset | https://{{pc_ip}}:9440/api/vmm/v4.1/ahv/config/vms?$page=2&$limit=100 |

Note: Pagination uses a zero-based index. The formula to use to see which elements are returned in the list given a page and limit is the following.

```

                starting index = page * limit

ending index = (limit * (page+1)) -1

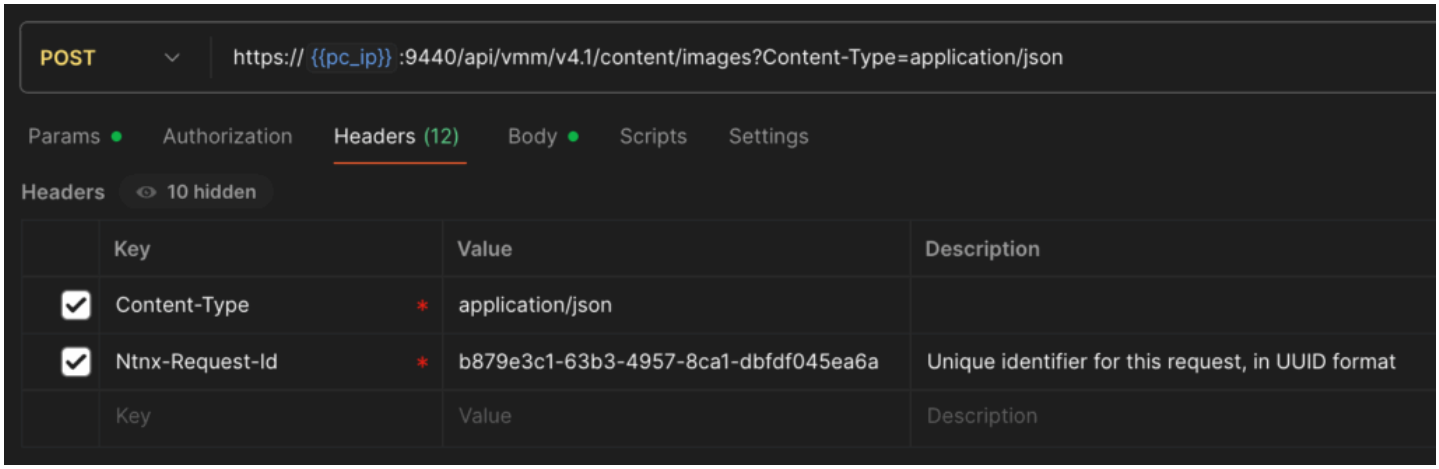
```

### Request Idempotency

To ensure each API request is only carried out once, the Nutanix v4 APIs now implement standard request headers, identified by the Ntnx-Request-Id header. The Ntnx-Request-Id header is mandatory on nearly all POST, PUT and DELETE requests.  Some asynchronous requests do not require the Ntnx-Request-Id header, as indicated in the Nutanix v4 API documentation.

NTNX-Request-Id is a unique identifier associated with each request. The provided value must be opaque and preferably in Universal Unique Identifier(UUID) format. This identifier is also used as an idempotence token for safely retrying requests in case of network errors. All the supported Nutanix v4 SDKs add this auto-generated request identifier to each request.

The method used to specify custom headers will vary depending on the API client.  Using Postman, however, an example is shown below.
```



For a complete explanation and demonstration of `Ntnx-Request-Id` header usage, see the article titled Using Request Id Headers with Nutanix v4 APIs.

Request Idempotency: Nutanix v4 SDKs

Note: The Nutanix v4 SDKs handle the `Ntnx-Request-Id` header automatically. Users should not manually add or specify the `Ntnx-Request-Id` header when using the Nutanix v4 SDKs.

Resource Etag

In the event of multiple users updating the same resource, it is possible for those updates to clash. This situation is known as a mid-air collision and results in incorrect or unknown results as the resource's configuration is not as expected.

To prevent this, the Nutanix v4 APIs now implement standard Etags, submitted via `If-Match` request headers. The `Etag` header is mandatory for all operations on existing entities and helps to eliminate the possibility of a mid-air collision.

Consider the following scenario.

- Administrator needs to update an existing disk image
- Administrator knows the `ext_id` of the existing disk image
- To prevent a mid-air collision, the disk image must be updated whilst specifying the image's current `Etag` as the value of the `If-Match` header

To complete this process, the steps are as follows.

1. Request the existing image:
2. Python SDK (assuming API instances are already setup as per previous snippets): api_instance.api.get_image_by_id(image_ext_id: str)
3. REST APIs: GET https://{{prism_central_ip_address_or_fqdn}}:9440/api/vmm/v4.1/content/images/{{image_extid}}
4. Obtain the image's Etag via the response headers:
5. Python SDK Etag property: print(api_client.get_etag(existing_image))
6. REST API Etag header: print(existing_image.headers["Etag"])
7. Generate image update request, using previously obtained Etag as the value of the `If-Match` header

Note: The code snippets in this section are for demonstration purposes only and will require additional steps before use.

Resource ETag: Python SDK

The following snippet uses the Nutanix v4 Python SDK to obtain an existing image by `ext_id`, then obtain the image's `Etag` from the response.

```
# build on the previous Nutanix v4 SDK code snippets to obtain an existing image's Etag

## get the existing image details

## replace ext_id with the ext_id of your image

existing_image = api_instance.get_image_by_id("ext_id")

## show the existing image's Etag

## the v4 APIs provide a get_etag function to make this simple print(api_client.get_etag(existing_image))
```

```

**Resource ETag: Python `requests` Library**

The following snippet uses the Python `requests` library to obtain an existing image by `ext_id`, then obtaining the image's `Etag` from the response.

```
build on the previous Nutanix v4 Python requests code snippet to obtain an existing image's Etag

set the Nutanix v4 API URL that will be used for this request

replace ext_id with the ext_id of your image

endpoint = "https://prism_central_ip_address_or_fqdn:9440/api/vmm/v4.1/content/images/ext_id"

get the existing image details

existing_image = requests.get(endpoint, auth=prism_central_auth, verify=False)

show the existing image's Etag

print(existing_image.headers["Etag"])
```

**Batch Operations**

The Nutanix v4 release candidate (RC) APIs and SDKs include support for asynchronous batch operations. Batch operations allow users to construct a multi-operation payload that can be applied to a collection of related entities.

For example, in an environment where there are a large number of virtual machines that need to be updated at the same time, a batch operation payload can be constructed so that all VMs have the required changes made with only a single update request on the user side.

Batch operations return standard v4 API and SDK task identifiers that allow users to monitor the status of a batch operation, including up to 90 days of batch operation history.

Note: Up to 500 entities can be included in a single batch operation.

**Batch Operations: Python SDK**

The following Python SDK code snippet shows the creation of a batch operation payload for creating 10 virtual machines. This code snippet assumes script preparation steps have been completed i.e. Prism Central IP address & credentials (etc).

- **ActionType.CREATE** will send a POST request. See v4 API: Submit Batch CREATE Operation (Python SDK) for a detailed example.
- **ActionType.MODIFY** will send a PUT request. See v4 API: Submit Batch MODIFY Operation (Python SDK) for a detailed example. Includes full usage of all required headers and parameters.

```
""" Use the Nutanix v4 SDKs to update submit a batch operation This demo creates 50 individual virtual machine

import ntnx_prism_py_client import ntnx_vmm_py_client

from ntnx_prism_py_client import Configuration from ntnx_prism_py_client import ApiClient

from ntnx_prism_py_client import BatchSpecPayload from ntnx_prism_py_client import BatchSpec from ntnx_prism_py_client import BatchSpecMetadata from ntnx_prism_py_client import ActionType from ntnx_prism_py_client import BatchesApi

setup configuration here i.e. Prism Central IP, credentials (etc)

batches_instance = BatchesApi(api_client=prism_client)

the number of virtual machines to create plus an empty list that will be populated with each VM payload

vm_count = 10 batch_spec_payload_list = [] for i in range(int(vm_count)): vm_payload = { "name": f"newvm_{i}", "description": f"new batch operation vm #{i}", "memory_size_bytes": 1024 * 1024 * 1024, # obtain the cluster ID using clustermgmt namespace first "cluster": ntnx_vmm_py_client.AhvConfigClusterReference("cluster_ext_id_here"), } # add the VM batch payload to the list of VMs to be created
```
```

batch_spec_payload_list.append(BatchSpecPayload(data=vm_payload))

batch_spec = BatchSpec(metadata=BatchSpecMetadata(action=ActionType.CREATE, name=f"multi_operation", # URI to the v4 vms API uri="/api/vmm/v4.1/ahv/config/vms", stop_on_error=True, chunk_size=1,), payload=batch_spec_payload_list,)

submit the request

print(f"Submitting batch operation to create {vm_count} VMs ...") batch_response = batches_instance.submit_batch(async_req=False, body=batch_spec)

```

The response is an instance of **ntnx_prism_py_client.models.prism.v4.operations.SubmitBatchApiResponse**, as follows.

```

```
>>> print(batch_response)
```

{'_object_type': 'prism.v4.operations.SubmitBatchApiResponse', '_reserved': {'$fv': 'v4.r0'}, '_unknown_fields': {}, 'data': {'_object_type': 'prism.v4.config.TaskReference', '_reserved': {'$fv': 'v4.r0'}, '_unknown_fields': {}, 'ext_id': 'ZXJnb24=:6b08f012-ed56-4489-5f87-fce4e897bfdf'}}

```

#### Rate Limiting

The Nutanix v4 APIs will enforce rate limiting.  Whenever the incoming requests exceed a certain limit, further requests will be temporarily blocked. Once the number of incoming requests goes down , additional requests can be sent as normal.

The configured rate limit will vary based on the Prism Central configuration.

| Prism Central Version | Prism Central Memory (GB) | Prism Central CPU | Prism Central Disk Size | Rate Limit (requests per second) | | --- | --- | --- | --- | --- | | X-Small | 18 | 4 | 100 | 30 | | Small | 26 | 6 | 500 | 40 | | Large | 44 | 10 | 2500 | 60 | | Extra Large | 60 | 14 | 2500 | 80 |

#### Complete Code Samples: Request Idempotency and Resource Etag

These code samples demonstrate Prism Central image updates with end-to-end usage of the Nutanix v4 REST APIs and Python SDKs.  Before using these code samples, ensure your Python environment is configured as per “Configuring Your Python Environment”.

Note: These code samples assume there is at least one existing image in the configured Prism Central environment.

Both code samples can be downloaded from the Nutanixdev GitHub:

- Update Prism Central image with Resource Etag: Python SDK (Ntnx-Request-Id is not required when using the Nutanix v4 SDKs)
- Update Prism Central image with request idempotency and Resource Etag: Python `requests` library

##### Update Prism Central image with Resource Etag: Nutanix Python SDK

This complete code sample uses the Nutanix v4 Python SDKs and resource Etag to update an existing Prism Central image.

This code sample can be run using the following syntax:

```

```
python .py
```

```

```
""" Use the Nutanix v4 SDKs to update a Prism Central image Requires Prism Central pc.2024.3 or later and AOS
```

""" import getpass import argparse import urllib3 import sys

import ntnx_vmm_py_client from ntnx_vmm_py_client import ApiClient as VMMClient from ntnx_vmm_py_client import Configuration as VMMConfiguration from ntnx_vmm_py_client.rest import ApiException as VMMException

""" suppress warnings about insecure connections consider the security implications before doing this in a production environment """ urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

""" setup the command line parameters for this example only two parameters are required - the Prism Central IP address or FQDN - the Prism Central username; the script will prompt for the user's password so that it never needs to be stored in plain text """
```

parser = argparse.ArgumentParser() parser.add_argument("pc_ip", help="Prism Central IP address or FQDN") parser.add_argument("username", help="Prism Central username") args = parser.parse_args()

get the cluster password

cluster_password = getpass.getpass(prompt="Enter your Prism Central password: ", stream=None,) pc_ip = args.pc_ip username = args.username

make sure the user enters a password

if not cluster_password: while not cluster_password: print("Password cannot be empty. \ Enter a password or Ctrl-C/Ctrl-D to exit.") cluster_password = getpass.getpass(prompt="Enter your Prism Central password: ", stream=None)

if **name** == "**main**": config = VMMConfiguration() config.host = pc_ip config.username = username config.password = cluster_password config.max_retry_attempts = 1 config.backoff_factor = 3 config.verify_ssl = False

try: api_client = VMMClient(configuration=config) api_instance = ntnx_vmm_py_client.api.ImagesApi(api_client=api_client) # get a list of existing images images_list = api_instance.list_images(async_req=False) if images_list.metadata.total_available_results > 0: print(f"Images found: {len(images_list.data)}") else: print("No images found.") sys.exit()

```
# images have been found - update the first image in the list
# to begin, we must retrieve that image's details
existing_image = api_instance.get_image_by_id(images_list.data[0].ext_id)
# get the existing image's Etag
existing_image_etag = api_client.get_etag(existing_image)
# create a new Prism Central image
new_image = ntnx_vmm_py_client.models.vmm.v4.content.Image.Image()
new_image.data = existing_image.data
new_image.name = f"{existing_image.data.name} - Updated"
new_image.type = existing_image.data.type

# add the existing image's Etag as a new request header
api_client.add_default_header(header_name="If-Match", header_value=existing_image_etag)
# update the image using an asynchronous request (will wait until completion before returning)
image_update = api_instance.update_image_by_id(body=new_image, extId=existing_image.data.ext_id, async_req=False)
print(image_update)
```

except VMMException as e: print(f"Unable to authenticate using the supplied credentials. \ Check your username and/or password, then try again. \ Exception details: {e}")

```
```

Update Prism Central image with request idempotency and resource Etag: Python `requests` Library

```
```

```
"""" Use the Nutanix v4 REST APIs to update a Prism Central image
```

Requires Prism Central pc.2024.3 or later and AOS 7.0 or later """" import requests import urllib3 import getpass import argparse import uuid import json import sys from base64 import b64encode

"""" suppress warnings about insecure connections consider the security implications before doing this in a production environment """" urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

"""" setup the command line parameters for this example only two parameters are required - the Prism Central IP address or FQDN - the Prism Central username; the script will prompt for the user's password so that it never needs to be stored in plain text """" parser = argparse.ArgumentParser() parser.add_argument("pc_ip", help="Prism Central IP address or FQDN") parser.add_argument("username", help="Prism Central username") args = parser.parse_args()

get the cluster password

cluster_password = getpass.getpass(prompt="Enter your Prism Central password: ", stream=None,) pc_ip = args.pc_ip username = args.username

make sure the user enters a password

if not cluster_password: while not cluster_password: print("Password cannot be empty. Enter a password or Ctrl-C/Ctrl-D to exit.") cluster_password = getpass.getpass(prompt="Enter your Prism Central password: ", stream=None)

try: """" setup the HTTP Basic Authorization header based on the supplied username and password """" encoded_credentials = b64encode(bytes(f"{username}:{cluster_password}", encoding="ascii")).decode("ascii") auth_header = f"Basic {encoded_credentials}"

```
# setup the URL that will be used for the API request
url = f"https://{pc_ip}:9440/api/vmm/v4.1/content/images"
""" setup the request headers
note the use of {auth_header} i.e. the Basic Authorization credentials we setup earlier """
headers = {
   "Accept": "application/json",
   "Content-Type": "application/json",
   "Authorization": f"{auth_header}",
   "cache-control": "no-cache"
}

# submit the request
try:
   response = requests.request( "GET", url, headers=headers, verify=False, timeout=10 )
   if response.ok:
       # show a total count of images found
       print( f'Total images found: {response.json()["metadata"]["totalAvailableResults"]}')
   else:
       print(f"An error occurred while connecting to {pc_ip}.")
       print(response.text)
       sys.exit()

   # images have been found - update the first image in the list
   # to begin, we must retrieve that image's details
   existing_image_ext_id = response.json()["data"][0]["extId"]
   # get the existing image details
   url = f"https://{pc_ip}:9440/api/vmm/v4.1/content/images/{existing_image_ext_id}"
   existing_image = requests.get(url, headers=headers, verify=False, timeout=10)
   # get the existing image's resource Etag
   existing_image_etag = existing_image.headers["Etag"]
   # create a new UUID to be used as the value of the Ntnx-Request-Id header
   request_id = str(uuid.uuid1())
   # create new headers that include the existing image's resource Etag and Ntnx-Request-Id
   headers = {
       "Accept": "application/json",
       "Content-Type": "application/json",
       "Authorization": f"{auth_header}",
       "cache-control": "no-cache",
       "If-Match": existing_image_etag,
       "Ntnx-Request-Id": request_id
   }
   # create the image update's JSON request payload
   update_payload = existing_image.json()["data"]
   # alter the image's name
   update_payload["name"] = f'{update_payload["name"]} - Updated'
   # update the image
   url = f"https://{pc_ip}:9440/api/vmm/v4.1/content/images/{existing_image_ext_id}"
   update = requests.put(url, headers=headers, data=json.dumps(update_payload), verify=False, timeout=10)
   print(update.json())
except Exception as ex:
   print(f"An {type(ex).__name__} exception occurred while \

connecting to {pc_ip}.\n Argument: {ex.args}.")
```

catching all exceptions like this should be generally be avoided

except Exception as e: print(f"{e}")

```

Nutanix v4 API Early Access ("EA") and Release Candidate ("RC") Do not use EA and RC features in a production environment. Your use of any EA and RC technology is subject to the Nutanix License and Service Agreement: https://www.nutanix.com/legal/eula

### Copyright
```

Our decision to link to or reference an external site should not be considered an endorsement of any content on such a site. Certain information contained in this post may relate to, or be based on, studies, publications, surveys and other data obtained from third-party sources and our own internal estimates and research. While we believe these third-party studies, publications, surveys and other data are reliable as of the date of this paper, they have not independently verified unless specifically stated, and we make no representation as to the adequacy, fairness, accuracy, or completeness of any information obtained from a third-party.

All code samples are unofficial, are unsupported and will require extensive modification before use in a production environment. This content may reflect an experiment in a test environment. Results, benefits, savings, or other outcomes described depend on a variety of factors including use case, individual requirements, and operating environments, and this publication should not be construed as a promise or obligation to deliver specific outcomes.

This content may reflect an experiment in a test environment. Results, benefits, savings, or other outcomes described depend on a variety of factors including use case, individual requirements, and operating environments, and this publication should not be construed as a promise or obligation to deliver specific outcomes.