ONTAP® 9

REST API Developers Guide

July 2019 | 215-14126_B0
doccomments@netapp.com
Updated for ONTAP 9.6
## Contents

- **Deciding whether to use the ONTAP REST API Developers Guide** ........................................... 5
- **Understanding the ONTAP REST API** .................................................................................... 6
  - REST web services foundation .................................................................................. 6
  - ONTAP REST API versioning ............................................................................... 7
  - How to access the ONTAP API .......................................................................... 7
  - Basic operational characteristics ........................................................................ 8
  - Input variables controlling an API request .................................................. 9
  - Interpreting an API response .......................................................................... 12
  - How asynchronous processing works using the Job object ................................ 14
- **Getting started with the ONTAP REST API** ................................................................. 16
  - Hello world ..................................................................................................... 16
  - Accessing the ONTAP API documentation page ........................................ 16
  - Using the NetApp ONTAP Python client library ........................................ 17
    - Preparing to use the ONTAP Python client library ................................ 17
    - Script to retrieve the cluster configuration ................................................ 17
- **Considerations when using the ONTAP REST API** .............................................. 19
  - Object references and access ........................................................................ 19
  - Users of the ONTAPI API and ONTAP CLI .................................................. 20
  - Accessing the ONTAP CLI through the REST API ........................................ 21
  - Security using RBAC .................................................................................... 21
- **ONTAP REST API resource categories** ........................................................................ 22
  - Application .................................................................................................. 22
  - Cloud ......................................................................................................... 23
  - Cluster ......................................................................................................... 23
  - Name services .............................................................................................. 24
  - NAS ............................................................................................................. 25
  - Networking .................................................................................................. 27
  - NVMe .......................................................................................................... 29
  - SAN ............................................................................................................. 30
  - Security ...................................................................................................... 31
  - SnapMirror .................................................................................................. 33
  - Storage ....................................................................................................... 33
  - Support ....................................................................................................... 35
  - SVM ........................................................................................................... 37
- **Workflow processes using the ONTAP REST API** ............................................ 38
  - Preparing to use the workflow processes .................................................. 38
    - Workflow 1: Updating the cluster contact and checking job state ............... 39
- **ONTAP REST API transactions illustrated** ................................................................ 41
  - Retrieving the cluster software version .................................................... 41
- **Where to find additional information** ........................................................................ 43
- **Copyright** ........................................................................................................ 44
Deciding whether to use the ONTAP REST API Developers Guide

The ONTAP 9 REST API Developers Guide describes the REST application programming interface provided with ONTAP 9. You can use the API to administer ONTAP systems and the related storage resources.

Objectives of this guide
This guide provides a detailed description of the ONTAP REST API. It includes the conceptual information needed to understand and access the API as well as workflow examples to complete selected administrative tasks.

Audience assumptions
This guide is suitable for a diverse audience, including software developers automating ONTAP administration tasks and decision makers who only need a high-level understanding of the API.

Note: If you plan to use the API to perform ONTAP administration tasks, you must already have experience administering ONTAP storage systems. This guide does not provide information related to ONTAP administration. You should refer to the ONTAP 9 documentation as needed.

API Compatibility
NetApp makes every effort to keep the ONTAP REST API backwards compatible from release to release. However, there may be instances where it is necessary to make a breaking change to accommodate new product functionality. See ONTAP REST API versioning for more information. Also review the ONTAP 9 Release Notes for any changes to the API.

Related concepts

ONTAP REST API versioning on page 7

Related references

Where to find additional information on page 43

Related information

ONTAP 9 Release Notes
Understanding the ONTAP REST API

Beginning with version 9.6, ONTAP includes a REST web services API. You can use the REST API to deploy and administer ONTAP storage resources. Before using the API, you should understand its design, architectural components, and limitations.

REST web services foundation

Representational State Transfer (REST) is a style for creating distributed web applications. When applied to the design of a web services API, it establishes a set of technologies and best practices for exposing server-based resources and managing their states. It uses mainstream protocols and standards to provide a flexible foundation for administering ONTAP clusters.

Resources and state representation

Resources are the basic components of a web-based system. When creating a REST web services application, early design tasks include:

- Identification of system or server-based resources
  Every system uses and maintains resources. A resource can be a file, business transaction, process, or administrative entity. One of the first tasks in designing an application based on REST web services is to identify the resources.

- Definition of resource states and associated state operations
  Resources are always in one of a finite number of states. The states, as well as the associated operations used to affect the state changes, must be clearly defined.

URI endpoints

Every REST resource must be defined and made available using a well-defined addressing scheme. The endpoints where the resources are located and identified use a Uniform Resource Identifier (URI). The URI provides a general framework for creating a unique name for each resource in the network. The Uniform Resource Locator (URL) is a type of URI used with web services to identify and access resources. Resources are typically exposed in a hierarchical structure similar to a file directory.

HTTP messages

Hypertext Transfer Protocol (HTTP) is the protocol used by the web services client and server to exchange request and response messages about the resources. As part of designing a web services application, HTTP methods are mapped to the resources and corresponding state management actions. HTTP is stateless. Therefore, to associate a set of related requests and responses as part of one transaction, additional information must be included in the HTTP headers carried with the request and response data flows.

JSON formatting

While information can be structured and transferred between a web services client and server in several ways, the most popular option is JavaScript Object Notation (JSON). JSON is an industry standard for representing simple data structures in plain text and is used to transfer state information describing the resources. The ONTAP REST API uses JSON to format the data carried in the body of each HTTP request and response.
ONTAP REST API versioning

The REST API included with ONTAP 9 is assigned a version number. The API version number is independent of the ONTAP version number. You should be aware of the API version included with your release of ONTAP and how this might affect your use of the API.

The ONTAP system software contains versions of the REST API as follows:

**ONTAP 9.6**

Version 1

Related information

*ONTAP 9 Release Notes*

How to access the ONTAP API

You can access the ONTAP REST API in several different ways.

**Network considerations**

You can connect to the REST API through the following interfaces:

- Cluster management LIF
- Node management LIF
- SVM management LIF

The LIF you choose to use must be configured to support the HTTPS management protocol. Also, the firewall configuration in your network must allow the HTTPS traffic.

**Tip:** You should always use the cluster management LIF. This will load balance the API requests across all nodes, and avoid nodes that are offline or experiencing connectivity issues.

**ONTAP API online documentation page**

The ONTAP API online documentation page provides an access point when using a web browser. In addition to providing a way to execute individual API calls directly, the page includes a detailed description of the API, including input parameters and other options for each call. The API calls are organized into functional categories. See *ONTAP REST API resource categories* for more information.

The format of the URL used to access the documentation page for the most recent version of the API is:

https://<cluster_mgmt_ip_address>/docs/api

**Custom software and tools**

You can access the ONTAP API using any of several different programming languages and tools. Popular choices include Python, Java, Curl, and PowerShell. A program, script, or tool that uses the API acts as a REST web services client. Using a programming language enables a deeper understanding of the API and provides an opportunity to automate the ONTAP administration.

The format of the base URL used to directly access the most recent version of the API is:

https://<cluster_mgmt_ip_address>/api
To access a specific API version where multiple versions are supported, the format of the URL is:
https://<cluster_mgmt_ip_address>/api/v1

Basic operational characteristics

While REST establishes a common set of technologies and best practices, the details of each API can vary based on the design choices.

Request and response API transaction

Every REST API call is performed as an HTTP request to the ONTAP system which generates an associated response to the client. This request/response pair is considered an API transaction. Before using the API, you should be familiar with the input variables available to control a request and the contents of the response output.

Support for CRUD operations

Each of the resources available through the ONTAP REST API is accessed based on the CRUD model:

- Create
- Read
- Update
- Delete

For some of the resources, only a subset of the operations is supported. You should review the ONTAP API documentation page at your ONTAP cluster for more information about each resource.

Object identifiers

Each resource instance or object is assigned a unique identifier when it is created. In most cases, the identifier is a 128-bit UUID. These identifiers are globally unique within a specific ONTAP cluster. After issuing an API call that creates a new object instance, a URL with the associated id value is returned to the caller in the location header of the HTTP response. You can extract the identifier and use it on subsequent calls when referring to the resource instance.

**Attention:** The content and internal structure of the object identifiers can change at any time. You should only use the identifiers on the applicable API calls as needed when referring to the associated objects.

Object instances and collections

Depending on the resource path and HTTP method, an API call can apply to a specific object instance or a collection of objects.

Synchronous and asynchronous operations

There are two ways that ONTAP performs an HTTP request received from a client.

**Synchronous processing**

ONTAP performs the request immediately and responds with an HTTP status code of 200 or 201 if it is successful.

Every request using the methods GET, HEAD, and OPTIONS is always performed synchronously. In addition, requests that use POST, PATCH, and DELETE are designed to run synchronously if they are expected to complete in less than two seconds.
Asynchronous processing

If an asynchronous request is valid, ONTAP creates a background task to process the request and a job object to anchor the task. The 202 HTTP status is returned to the caller along with the job object. To determine final success or failure, you must retrieve the state of the job.

Requests that use the methods POST, PATCH, and DELETE are designed to run asynchronously if they are expected to take more than two seconds to complete.

Note: The return_timeout query parameter is available with asynchronous API calls and can convert an asynchronous call to complete synchronously. Refer to How asynchronous processing works using the Job object for more information.

Security

The security provided with the REST API is based primarily on the existing security features available with ONTAP. The following security is used by the API:

- Transport Layer Security
  All traffic sent over the network between the ONTAP LIF and client is typically encrypted using TLS, based on the ONTAP configuration settings.

- Client authentication
  The same authentication options available with ONTAP System Manager and the Network Manageability SDK can also be used with the ONTAP REST API.

- HTTP authentication
  At an HTTP level, basic authentication is used for the API transactions. An HTTP header with the user name and password in a base64 string is added to each request.

- ONTAP authorization
  ONTAP implements a role-based authorization model. The account you use when accessing the ONTAP REST API or API documentation page should have the proper authority.

Related concepts

Security using RBAC on page 21

Input variables controlling an API request

You can control how an API call is processed through parameters and variables set in the HTTP request.

HTTP methods

The HTTP methods supported by the ONTAP REST API are shown in the following table.

Note: Not all of the HTTP methods are available at each of the REST endpoints. Also, both PATCH and DELETE can be used on a collection. See Object references and access for more information.

<table>
<thead>
<tr>
<th>HTTP method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Retrieves object properties on a resource instance or collection.</td>
</tr>
<tr>
<td>POST</td>
<td>Creates a new resource instance based on the supplied input.</td>
</tr>
<tr>
<td>PATCH</td>
<td>Updates an existing resource instance based on the supplied input.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Deletes an existing resource instance.</td>
</tr>
</tbody>
</table>
### HTTP methods

<table>
<thead>
<tr>
<th>HTTP method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HEAD</td>
<td>Effectively issues a GET request but only returns the HTTP headers.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Determine what HTTP methods are supported at a specific endpoint.</td>
</tr>
</tbody>
</table>

### Request headers

You must include several headers in the HTTP request.

#### Content-type

If the request body includes JSON, this header must be set to `application/json`.

#### Accept

This header should be set to `application/hal+json`. If it is instead set to `application/json` none of the HAL links will be returned except a link needed to retrieve the next batch of records. If the header is anything else aside from these two values, the default value of the `content-type` header in the response will be `application/hal+json`.

#### Authorization

Basic authentication must be set with the user name and password encoded as a base64 string.

### Request body

The content of the request body varies depending on the specific call. The HTTP request body consists of one of the following:

- JSON object with input variables
- Empty

### Filtering objects

When issuing an API call that uses GET, you can limit or filter the returned objects based on any attribute. For example, you can specify an exact value to match:

```
<field>=<query value>
```

In addition to an exact match, other operators are available to return a set of objects over a range of values. The ONTAP REST API supports the filtering operators shown in the table below.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Not equal to</td>
</tr>
<tr>
<td>*</td>
<td>Greedy wildcard</td>
</tr>
</tbody>
</table>

You can also return a collection of objects based on whether a specific field is set or not set by using the `null` keyword or its negation `!null` as part of the query.
**Note:** Any fields that are not set are generally excluded from matching queries.

**Requesting specific object fields**

By default, issuing an API call using GET returns only the attributes that uniquely identify the object or objects, along with a HAL self link. This minimum set of fields acts as a key for each object and varies based on the object type. You can select additional object properties using the `fields` query parameter in the following ways:

- **Common or standard fields**
  Specify `fields=*` to retrieve the most commonly used object fields. These fields are typically maintained in local server memory or require little processing to access. These are the same properties returned for an object after using GET with a URL path key (UUID).

- **All fields**
  Specify `fields=**` to retrieve all the object fields, including those requiring additional server processing to access.

- **Custom field selection**
  Use `fields=<field_name>` to specify the exact field you want. When requesting multiple fields, the values must be separated using commas without spaces.

**Attention:** As a best practice, you should always identify the specific fields you want. You should only retrieve the set of common fields or all fields when needed. Which fields are classified as common, and returned using `fields=*`, is determined by NetApp based on internal performance analysis. The classification of a field might change in future releases.

**Sorting objects in the output set**

The records in a resource collection are returned in the default order defined by the object. You can change the order using the `order_by` query parameter with the field name and sort direction as follows:

```
order_by=<field name> asc|desc
```

For example, you can sort the `type` field in descending order followed by `id` in ascending order:

```
order_by=type desc, id asc
```

Note the following:

- If you specify a sort field but don’t provide a direction, the values are sorted in ascending order.
- When including multiple parameters, you must separate the fields with a comma.

**Pagination when retrieving objects in a collection**

When issuing an API call using GET to access a collection of objects of the same type, ONTAP attempts to return as many objects as possible based on two constraints. You can control each of these constraints using additional query parameters on the request. The first constraint reached for a specific GET request terminates the request and therefore limits the number of records returned.

**Note:** If a request ends before iterating over all the objects, the response contains the link needed to retrieve the next batch of records.

**Limiting the number of objects**

By default, ONTAP returns a maximum of 10,000 objects for a GET request. You can change this limit using the `max_records` query parameter. For example:

```
max_records=20
```
The number of objects actually returned can be less than the maximum in effect, based on
the related time constraint as well as the total number of objects in the system.

**Limiting the time used to retrieve the objects**

By default, ONTAP returns as many objects as possible within the time allowed for the
GET request. The default timeout is 15 seconds. You can change this limit using the
`return_timeout` query parameter. For example:

\[
\text{return_timeout=5}
\]

The number of objects actually returned can be less than the maximum in effect, based on
the related constraint on the number of objects as well as the total number of objects in the
system.

**Narrowing the result set**

If needed, you can combine these two parameters with additional query parameters to
narrow the result set. For example, the following returns up to 10 ems events generated
after the specified time:

\[
\text{time}=> 2018-04-04T15:41:29.140265Z&max_records=10
\]

You can issue multiple requests to page through the objects. Each subsequent API call
should use a new time value based on the latest event in the last result set.

**Size properties**

The input values used with some API calls as well as certain query parameters are numeric. Rather
than provide an integer in bytes, you can optionally use a suffix as shown in the following table.

<table>
<thead>
<tr>
<th>Suffix</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KB</td>
<td>Kilobytes (1024 bytes) or kibibytes</td>
</tr>
<tr>
<td>MB</td>
<td>Megabytes (KB x 1024 bytes) or mebibytes</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabytes (MB x 1024 bytes) or gibibytes</td>
</tr>
<tr>
<td>TB</td>
<td>Terabytes (GB x 1024 bytes) or tebibytes</td>
</tr>
<tr>
<td>PB</td>
<td>Petabytes (TB x 1024 bytes) or pebibytes</td>
</tr>
</tbody>
</table>

**Related concepts**

*Object references and access* on page 19

**Interpreting an API response**

Each API request generates a response back to the client. You should examine the response to
determine whether it was successful and retrieve additional data as needed.

**HTTP status code**

The HTTP status codes used by the ONTAP REST API are described below.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>OK</td>
<td>Indicates success for calls that do not create a new object.</td>
</tr>
<tr>
<td>201</td>
<td>Created</td>
<td>An object is successfully created. The location header in the response includes the unique identifier for the object.</td>
</tr>
<tr>
<td>Code</td>
<td>Meaning</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>202</td>
<td>Accepted</td>
<td>A background job has been started to perform the request, but has not completed yet.</td>
</tr>
<tr>
<td>400</td>
<td>Bad request</td>
<td>The request input is not recognized or is inappropriate.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>User authentication has failed.</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>Access is denied due to an authorization (RBAC) error.</td>
</tr>
<tr>
<td>404</td>
<td>Not found</td>
<td>The resource referred to in the request does not exist.</td>
</tr>
<tr>
<td>405</td>
<td>Method not allowed</td>
<td>The HTTP method in the request is not supported for the resource.</td>
</tr>
<tr>
<td>409</td>
<td>Conflict</td>
<td>An attempt to create an object failed because a different object must be created first or the requested object already exists.</td>
</tr>
<tr>
<td>500</td>
<td>Internal error</td>
<td>A general internal error occurred at the server.</td>
</tr>
</tbody>
</table>

**Response headers**

Several headers are included in the HTTP response generated by the ONTAP.

**Location**

When an object is created, the location header includes the complete URL to the new object including the unique identifier assigned to the object.

**Content-type**

This will normally be `application/hal+json`.

**Response body**

The content of the response body resulting from an API request differs based on the object, processing type, and the success or failure of the request. The response is always rendered in JSON.

- **Single object**
  
  A single object can be returned with a set of fields based on the request. For example, you can use GET to retrieve selected properties of a cluster using the unique identifier.

- **Multiple objects**
  
  Multiple objects from a resource collection can be returned. In all cases, there is a consistent format used, with `num_records` indicating the number of records and `records` containing an array of the object instances. For example, you can retrieve the nodes defined in a specific cluster.

- **Job object**
  
  If an API call is processed asynchronously, a Job object is returned which anchors the background task. For example, the PATCH request used to update the cluster configuration is processed asynchronously and returns a Job object.

- **Error object**
  
  If an error occurs, an Error object is always returned. For example, you will receive an error when attempting to change a field not defined for a cluster.

- **Empty**
  
  In certain cases, no data is returned and the response body includes an empty JSON object.
HAL linking

The ONTAP REST API uses HAL as the mechanism to support Hypermedia as the Engine of Application State (HATEOAS). When an object or attribute is returned that identifies a specific resource, a HAL-encoded link is also included allowing you to easily locate and determine additional details about the resource.

Errors

If an error occurs, an error object is returned in the response body.

Format

An error object has the following format:

```
"error": {
    "message": "<string>",
    "code": <integer>[,
    "target": "<string>"]
}
```

You can use the code value to determine the general error type or category, and the message to determine the specific error. When available, the target field includes the specific user input associated with the error.

Common error codes

The common error codes are described in the following table. Specific API calls can include additional error codes.

<table>
<thead>
<tr>
<th>Error code</th>
<th>HTTP status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>409</td>
<td>An object with the same identifier already exists.</td>
</tr>
<tr>
<td>2</td>
<td>400</td>
<td>The value for a field has an invalid value or is missing, or an extra field was provided.</td>
</tr>
<tr>
<td>3</td>
<td>400</td>
<td>The operation is not supported.</td>
</tr>
<tr>
<td>4</td>
<td>405</td>
<td>An object with the specified identifier cannot be not found.</td>
</tr>
<tr>
<td>6</td>
<td>403</td>
<td>Permission to perform the request is denied.</td>
</tr>
<tr>
<td>8</td>
<td>409</td>
<td>The resource is in use.</td>
</tr>
</tbody>
</table>

How asynchronous processing works using the Job object

After issuing an API request that is designed to run asynchronously, a job object is always created and returned to the caller. The job describes and anchors a background task that processes the request. Depending on the HTTP status code, you must retrieve the state of the job to determine if the request was successful.

Refer to the ONTAP API online documentation page to determine which API calls are designed to be performed asynchronously.

Controlling how a request is processed

You can use the `return_timeout` query parameter to control how an asynchronous API call is processed. There are two possible outcomes when using this parameter.
Timer expires before the request completes
For valid requests, ONTAP returns a 202 HTTP status code along with the job object. You must retrieve the state of the job to determine if the request completed successfully.

Request is completed before the timer expires
If the request is valid and completes successfully before the time expires, ONTAP returns a 200 HTTP status code along with the job object. Because the request is completed synchronously, as indicated by the 200, you do not need to retrieve the job state.

Note: The default value for the return_timeout parameter is zero seconds. Therefore, if you don't include the parameter, the 202 HTTP status code is always returned for a valid request.

Querying the Job object associated with an API request
The Job object returned in the HTTP response contains several properties. You can query the state property in a subsequent API call to determine if the request completed successfully. A Job object is always in one of the following states:

Non-terminal states
- Queued
- Running
- Paused

Terminal states
- Success
- Failure

General procedure for issuing an asynchronous request
You can use the following high-level procedure to complete an asynchronous API call. This example assumes the return_timeout parameter is not used, or that the time expires before the background job completes.

1. Issue an API call that is designed to be performed asynchronously.
2. Receive an HTTP response 202 indicating acceptance of a valid request.
3. Extract the identifier for the Job object from the response body.
4. Within a timed loop, perform the following in each cycle:
   a. Get the current state of the Job.
   b. If the Job is in a non-terminal state, perform loop again.
5. Stop when the Job reaches a terminal state (success, failure).

Related references
Workflow 1: Updating the cluster contact and checking job state on page 39
Getting started with the ONTAP REST API

You can quickly get started using the ONTAP REST API. Accessing the API provides some perspective before you begin using it with the more complex workflow processes on a live system.

Hello world

You can issue a simple command at your workstation's CLI to get started using the ONTAP REST API and confirm its availability.

Before you begin

The Curl utility must be available on your workstation. In addition, you must have the following:

• IP address or host name of the ONTAP cluster management LIF
• User name and password for an account with authority to access the ONTAP REST API (cluster resource).

Step

1. At the command line interface of your local workstation, retrieve the version of the ONTAP software:

   curl -X GET -u username:password -k 'https://<ip_address>/api/cluster?fields=version'

Example


After you finish

The ONTAP version information is displayed in a JSON format.

Accessing the ONTAP API documentation page

You can access the ONTAP API online documentation page to display the documentation for the REST API.

Before you begin

You must have the following:

• IP address or host name of the ONTAP cluster management LIF
• User name and password for an account with authority to access the ONTAP REST API

Steps

1. Type the URL in your browser and press Enter:

   https://<ip_address>/docs/api
2. Sign in using the ONTAP account.
   The ONTAP API documentation page is displayed with the API calls organized in major resource categories at the bottom.

3. To see the details of an individual API call, scroll down to the cluster category and click GET /cluster.

Using the NetApp ONTAP Python client library
The ONTAP Python client library is a package you can install to write scripts to access the ONTAP REST API. It provides support for several underlying services, including connection management, asynchronous processing, exception handling, and error messages. By using the Python client library, you can quickly develop robust code to support the automation of ONTAP deployments.

Preparing to use the ONTAP Python client library
You should prepare the environment before using the Python client library.

Basic requirements
You must use Python 3.5 or later. The following packages are also required:

- requests 2.21.0 or later
- marshmallow 3.0.0rc5 or later

Package name and version
The name of the Python client library package is netapp_ontap. The version associated with the package is a combination of the ONTAP major and minor version numbers the library was generated from, along with a minor version for the client within the ONTAP release. For example, valid version numbers include: 9.6.1, 9.6.2, and 9.7.1.

Installation
You must use pip to install the netapp_ontap package from the Python Package Index (PyPi) website.

Documentation and additional resources
See the NetApp Developer Network for links to documentation and other resources.

Related information
NetApp Developer Network: ONTAP RESTful API

Script to retrieve the cluster configuration
The following script provides a simple example of how to use the Python client library. You can run the script using Python 3 at the CLI to retrieve the ONTAP cluster configuration.

```bash
#!/usr/bin/env python3
#
# Description: Python script to retrieve the cluster configuration.
#
# Usage example:
#   python3 get_cluster.py
```

```bash
```
# (C) Copyright 2019 NetApp, Inc.
# This sample code is provided AS IS, with no support or warranties of any kind, including but not limited for warranties of merchantability or fitness of any kind, expressed or implied. Permission to use, reproduce, modify and create derivatives of the sample code is granted solely for the purpose of researching, designing, developing and testing a software application product for use with NetApp products, provided that the above copyright notice appears in all copies and that the software application product is distributed pursuant to terms no less restrictive than those set forth herein.
#
###--------------------------------------------------------------------

# Global configuration for the library
from netapp_ontap import config

# Support for the connection to ONTAP
from netapp_ontap import HostConnection

# Specific API needed for this script
from netapp_ontap.resources import Cluster

# Optionally suppress warning for unsigned certificate
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

# Create connection to the ONTAP management LIF
conn = HostConnection("10.236.252.97", username="admin",
password="mypassword", verify=False)

# Set connection as the default for all API calls
config.CONNECTION = conn

# Create new cluster object
clus = Cluster()

# Issue REST API call
clus.get()

# Display the cluster configuration
print(clus)
Considerations when using the ONTAP REST API

There are several characteristics of the ONTAP REST API affecting its operation and use. You should be aware of these considerations before issuing an API call.

You should review the resources exposed through the API.

Related references

ONTAP REST API resource categories on page 22

Object references and access

The resource instances or objects exposed through the ONTAP REST API can be referenced and accessed in several different ways.

Object access paths

At a high level, there are two path types when accessing an object:

- Primary
  The object is the primary or direct target of the API call.

- Foreign
  The object is not the primary reference of the API call, but rather is linked to from the primary object. It is therefore a foreign or downstream object, and referenced through a field in the primary object.

Accessing an object using the UUID

Every object is assigned a unique identifier when it is created, which in most cases is a 128-bit UUID. The assigned UUID values are immutable, and are used internally within ONTAP to access and manage the resources. Because of this, the UUID generally provides the fastest and most stable way to access objects.

For many of the resource types, a UUID value can be provided as part of the path key in the URL to access a specific object. For example, you can use the following to access a node instance: /cluster/nodes/{uuid}

Accessing an object using an object property

In addition to a UUID, you can also access an object using an object property. In most cases, it is convenient to use the name property. For example, you can use the following query parameter in the URL string to access a node instance by its name: /cluster/nodes?name=node_one. In addition to a query parameter, a foreign object can be accessed through a property in the primary object.

While you can use the name or other property to access an object instead of the UUID, there are several possible disadvantages:

- The name field is not immutable and can be changed. If the name of an object is changed before accessing an object, the wrong object will be returned or an object access error will fail.

  Note: This issue can occur with a POST or PATCH method on a foreign object or with a GET method on a primary object.

- ONTAP must translate the name field into the corresponding UUID. This is a type of indirect access which can become a performance issue.
In particular, a performance degradation is possible when one or more of the following is true:

- GET method is used
- A large collection of objects is accessed
- A complex or elaborate query is used

**Cluster versus SVM context**

There are several REST endpoints that support both a cluster and SVM. When using one of these endpoints, you can indicate the context of the API call through the `scope=[svm|cluster]` value. Examples of endpoints supporting a dual context include IP interfaces and security roles.

**Note:** The scope value has a default value base on the properties provided for each API call.

**Using PATCH and DELETE on a collection of objects**

Every REST endpoint supporting PATCH or DELETE on a resource instance also supports the same method on a collection of objects. The only requirement is that at least one field must be provided through a query parameter in the URL string. When issuing a PATCH or DELETE over a collection, this is equivalent to doing the following internally:

- Query-based GET to retrieve the collection
- Serial sequence of PATCH or DELETE calls on each object in the collection

The time out for the operation can be set by `return_timeout` with a default of 15 seconds. If not completed before the timeout, the response includes a link to the next object. You must reissue the same HTTP method using the next link to continue the operation.

**Users of the ONTAPI API and ONTAP CLI**

ONTAPI is a set of API calls provided through the Network Manageability SDK that can be used to administer ONTAP storage systems. There are differences between the REST API and the ONTAPI API calls, as well as between the REST API and ONTAP CLI. You should understand these differences before using the REST API in a production environment.

**General design differences**

The ONTAP REST API and command line interface have a fundamentally different designs. The CLI commands and parameters do not map directly to the REST API calls. And even where there might be a similarity, the details of the input parameters can be different. For example, numeric units might be specified in bytes or using a suffix (such as KB). You should review *Input variables controlling an API request* as well as the ONTAP online documentation page for more information.

**Data SVMs exposed through the API**

ONTAP supports several types of storage virtual machines (SVMs). However, only the data SVMs are directly exposed through the ONTAP REST API. The configuration information describing the cluster and nodes is available through the REST API, however the cluster and nodes are not treated as separate SVMs.

**Related concepts**

*Input variables controlling an API request* on page 9
Accessing the ONTAP CLI through the REST API

To assist CLI and ONTAPI API users in their transition to the ONTAP REST API, ONTAP provides a REST endpoint to access the CLI. You can use this passthrough feature to execute any CLI command. Usage of the REST endpoint is returned in AutoSupport data so that NetApp can identify gaps in the REST API and make improvements in future releases.

To issue a CLI command, you must make a REST API call that is properly formed based on rules regarding:
- Resource paths
- Field names
- HTTP methods

The base resource path for CLI access is /private/cli. Refer to the ONTAP API online documentation page for details about accessing the CLI through the REST API.

Security using RBAC

The REST API expands the role-based access control (RBAC) capabilities when using ONTAP. You can create user accounts with custom roles to restrict access to the REST endpoints.

Creating roles for the REST endpoints

A REST role is defined through a set of one or more privileges. Each privilege consists of a path to a REST endpoint and the associated access level. Access to each endpoint is provided in one of three levels and determines the HTTP methods that can be used against the resource. The access levels include:
- All
  All HTTP methods can be used
- Read-only
  Only GET can be used
- None
  No access is allowed

Here is an example of two privileges that can be assigned to a REST role:
- access="readonly", path="/api/storage/volumes"
- access="none", path="/api/snapmirror/policies"

Creating a user account with a custom role

At a high level, you can create an account with a custom REST role as follows:

1. Create a user account with access to the HTTP management protocol.
2. Create a REST role with the desired privileges.
3. Associate the user account with the role.
ONTAP REST API resource categories

The resources available through the ONTAP REST API are organized in categories, as displayed on the ONTAP API documentation page. A brief description of each of the resources with the base resource paths is presented below, along with additional usage considerations where appropriate.

Attention: For many of the REST endpoints, you can include a UUID key as part of the path string to access a specific object instance. However, you can also access objects using a property value on a query parameter.

Related concepts

Considerations when using the ONTAP REST API on page 19

Related tasks

Accessing the ONTAP API documentation page on page 16

Application

You can use these API calls to manage the ONTAP application resources.

Applications

The ONTAP applications are arranged based on type, including: templates, applications, components, and Snapshot copies.

- /application/applications
- /application/applications/{uuid}
- /application/applications/{application.uuid}/components
- /application/applications/{application.uuid}/components/{uuid}
- /application/applications/{application.uuid}/components/{component.uuid}/snapshots
- /application/applications/{application.uuid}/components/{component.uuid}/snapshots/{uuid}
- /application/applications/{application.uuid}/components/{component.uuid}/snapshots/{uuid}/restore

Application snapshots

Applications support snapshot copies, which can be created or restored at any time.

- /application/applications/{application.uuid}/snapshots
- /application/applications/{application.uuid}/snapshots/{uuid}
- /application/applications/{application.uuid}/snapshots/{uuid}/restore
- /application/templates
- /application/templates/{name}
Cloud

You can use these API calls to manage connections to object storage resources in the cloud.

**Targets**
A target represents an object storage resource in the cloud. Each target includes the configuration information needed to connect to the storage resource.

- /cloud/targets
- /cloud/targets/{uuid}

Cluster

You can use these API calls to manage clusters and the related resources.

**Clusters**
An ONTAP cluster contains one or more nodes and the related configuration settings which define the storage system.

- /cluster

**Chassis**
The chassis is the hardware framework supporting a cluster.

- /cluster/chassis
- /cluster/chassis/{id}

**Jobs**
Asynchronous API calls are performed by a background task anchored by a job.

- /cluster/jobs
- /cluster/jobs/{uuid}

**Licenses**
The licenses allow you to implement specific ONTAP features and functionality.

- /cluster/licensing/licenses

**Licenses instance**
Each license can be managed as a separate package.

- /cluster/licensing/licenses/{name}

**Nodes**
ONTAP clusters are composed of one or more nodes.

- /cluster/nodes
- /cluster/nodes/{uuid}
Peers
The peer objects represent endpoints and support the cluster peering relationships.

- /cluster/peers
- /cluster/peers/{uuid}

Schedules
Schedules can be used to automate the perform of tasks.

- /cluster/schedules
- /cluster/schedules/{uuid}

Software
An ONTAP cluster includes the cluster software profile, software packages collection, and software history collection.

- /cluster/software
- /cluster/software/download
- /cluster/software/history
- /cluster/software/packages
- /cluster/software/packages/{version}

Name services
You can use these API calls to manage the name services supported by ONTAP.

DNS
DNS supports the integration of the ONTAP cluster in your network.

- /name-services/dns
- /name-services/dns/{svm.uuid}

LDAP
LDAP servers can be used to maintain user information.

- /name-services/ldap
- /name-services/ldap/{svm.uuid}

Name mappings
Name mappings allow you to map identities from one name domain to another. For example, you can map identities from CIFS to UNIX, Kerberos to UNIX, and UNIX to CIFS.

- /name-services/name-mappings
- /name-services/name-mappings/{svm.uuid}/{direction}/{index}
**NIS**

NIS servers can be used to authenticate users and client workstations.

- `/name-services/nis`
- `/name-services/nis/{svm.uuid}`

**NAS**

You can use these API calls to manage the CIFS and NFS settings for the cluster and SVMs.

**Audit**

Certain CIFS and NFS events can be logged for the SVMs, which can help to improve security.

- `/protocols/audit`
- `/protocols/audit/{svm.uuid}`

**CIFS home directory search paths**

Home directories for SMB users on a CIFS server can be located created without creating an individual SMB share for each user. The home directory search path is a set of absolute paths from the root of an SVM.

- `/protocols/cifs/home-directory/search-paths`
- `/protocols/cifs/home-directory/search-paths/{svm.uuid}/[index]`

**CIFS services**

The core configuration of the CIFS server.

- `/protocols/cifs/services`
- `/protocols/cifs/services/{svm.uuid}`

**CIFS shares**

The SMB shares defined at a CIFS server.

- `/protocols/cifs/shares`
- `/protocols/cifs/shares/{svm.uuid}/{name}`

**CIFS shares ACLs**

The access control lists (ACLs) controlling access to folders and files on the CIFS shares.

- `/protocols/cifs/shares/{svm.uuid}/{share}/acls`
- `/protocols/cifs/shares/{svm.uuid}/{share}/acls/{user|group}/{type}`

**CIFS UNIX symlink mapping**

Both CIFS and UNIX clients can access the same datastore. When UNIX clients create symbolic links, these mappings provide a reference to another file or folder to support the CIFS clients.

- `/protocols/cifs/unix-symlink-mapping`
- `/protocols/cifs/unix-symlink-mapping/{svm.uuid}/{unix_path}`
FPolicy
FPolicy is a file access notification framework used to monitor and manage file access events on the SVMs.

• /protocols/fpolicy
• /protocols/fpolicy/{svm.uuid}

FPolicy engines
The FPolicy engines allow you to identify the external servers that receive the file access notifications.

• /protocols/fpolicy/{svm.uuid}/engines
• /protocols/fpolicy/{svm.uuid}/engines/{name}

FPolicy events
The configuration identifying how file access is monitored and what events are generated.

• /protocols/fpolicy/{svm.uuid}/events
• /protocols/fpolicy/{svm.uuid}/events/{name}

FPolicy policies
A container for elements of the FPolicy framework, including FPolicy engines and events.

• /protocols/fpolicy/{svm.uuid}/policies
• /protocols/fpolicy/{svm.uuid}/policies/{name}

NFS export policies
The policies including rules that describe the NFS exports.

• /protocols/nfs/export-policies
• /protocols/nfs/export-policies/{id}
• /protocols/nfs/export-policies/{policy.id}/rules
• /protocols/nfs/export-policies/{policy.id}/rules/{index}
• /protocols/nfs/export-policies/{policy.id}/rules/{index}/clients
• /protocols/nfs/export-policies/{policy.id}/rules/{index}/clients/{match}

NFS Kerberos interfaces
The configuration settings for an interface to Kerberos.

• /protocols/nfs/kerberos/interfaces
• /protocols/nfs/kerberos/interfaces/{uuid}

NFS Kerberos realms
The configuration settings for Kerberos realms.

• /protocols/nfs/kerberos/realms
• /protocols/nfs/kerberos/realms/{svm.uuid}/[name]

**NFS services**
The core configuration of the NFS server.
• /protocols/nfs/services
• /protocols/nfs/services/{svm.uuid}

**Vscan**
A security feature to protect your data from viruses and other malicious code.
• /protocols/vscan
• /protocols/vscan/{svm.uuid}

**Vscan on-access policies**
The Vscan policies allowing files objects to be actively scanned when accessed by a client.
• /protocols/vscan/{svm.uuid}/on-access-policies
• /protocols/vscan/{svm.uuid}/on-access-policies/[name]

**Vscan On-demand policies**
The Vscan policies allowing files objects to be immediately scanned on demand or according to a set schedule.
• /protocols/vscan/{svm.uuid}/on-demand-policies
• /protocols/vscan/{svm.uuid}/on-demand-policies/[name]

**Vscan scanner pools**
A set of attributes used to manage the connection between ONTAP and an external virus-scanning server.
• /protocols/vscan/{svm.uuid}/scanner-pools
• /protocols/vscan/{svm.uuid}/scanner-pools/[name]

**Vscan server status**
The status of the external virus-scanning server.
• /protocols/vscan/server-status

**Networking**
You can use these API calls to manage the physical and logical networking resources used with the cluster.

**Ethernet broadcast domains**
An Ethernet broadcast domain is a set of physical ports that appear to be part of the same physical network. All of the ports receive a packet when broadcast from one of the ports in the domain. Each broadcast domain is part of an IPspace.
• /network/ethernet/broadcast-domains
• /network/ethernet/broadcast-domains/{uuid}

Ethernet ports
An Ethernet port is a physical or virtual networking endpoint. The ports can be combined into a Link Aggregate Group (LAG) or separated using a Virtual LAN (VLAN).
• /network/ethernet/ports
• /network/ethernet/ports/{uuid}

Fibre Channel interfaces
A Fibre Channel interface is a logical endpoint associated with an SVM.
• /network/fc/interfaces
• /network/fc/interfaces/{uuid}

Fibre Channel ports
A Fibre Channel port is a physical adapter on an ONTAP node used to connect to the Fibre Channel network.
• /network/fc/ports
• /network/fc/ports/{uuid}

IP interfaces
A logical interface (LIF) is an IP address with additional configuration attributes.
• /network/ip/interfaces
• /network/ip/interfaces/{uuid}

IP routes
A routing table is a collection of IP routes used to forward traffic to its destination.
• /network/ip/routes
• /network/ip/routes/{uuid}

IP service policies
The IP service policies define the services available at a specific LIF. Service policies can be configured within the context of an SVM or IPspace.
• /network/ip/service-policies
• /network/ip/service-policies/{uuid}

IPspaces
An IPspace creates a networking space to support one or more SVMs. The IPspaces can be isolated from each other, providing security and privacy.
• /network/ipspaces
• /network/ipspaces/{uuid}
**NVMe**

You can use these API calls to manage resources supporting non-volatile memory express (NVMe).

**Fibre Channel logins**
Fibre Channel logins represent connections formed by Fibre Channel initiators logged in to ONTAP.

- `/network/fc/logins`
- `/network/fc/logins/{interface.uuid}/{initiator.wwpn}`

**NVMe interfaces**
NVMe interfaces are the network interfaces configured to support the NVMe over Fabrics (NVMe-oF) protocol.

- `/protocols/nvme/interfaces`
- `/protocols/nvme/interfaces/{uuid}`

**NVMe services**
An NVMe service defines the properties of the NVMe controller target for an SVM.

- `/protocols/nvme/services`
- `/protocols/nvme/services/{svm.uuid}`

**NVMe subsystem controllers**
The NVMe subsystem controllers represent dynamic connections between hosts and a storage solution.

- `/protocols/nvme/subsystem-controllers`
- `/protocols/nvme/subsystem-controllers/{subsystem.uuid}/{id}`

**NVMe subsystem maps**
An NVMe subsystem map is an association of an NVMe namespace with an NVMe subsystem.

- `/protocols/nvme/subsystem-maps`
- `/protocols/nvme/subsystem-maps/{subsystem.uuid}/{namespace.uuid}`

**NVMe subsystems**
An NVMe subsystem maintains configuration state and namespace access control for a set of NVMe-connected hosts.

- `/protocols/nvme/subsystems`
- `/protocols/nvme/subsystems/{uuid}`
- `/protocols/nvme/subsystems/{subsystem.uuid}/hosts`
- `/protocols/nvme/subsystems/{subsystem.uuid}/hosts/{nqn}`
Namespaces

An NVMe namespace is a collection of addressable logical blocks presented to hosts connected to the SVM using the NVMe over Fabrics protocol.

- /storage/namespaces
- /storage/namespaces/{uuid}

SAN

You can use these API calls to manage storage area networking (SAN) resources.

Fibre Channel logins

Fibre Channel logins represent connections formed by Fibre Channel initiators that have logged in to ONTAP.

- /network/fc/logins
- /network/fc/logins/{interface.uuid}/{initiator.wwpn}

Fibre Channel WWPN aliases

A world wide port name (WWPN) is a 64-bit value uniquely identifying a Fibre Channel port.

- /network/fc/wwpn-aliases
- /network/fc/wwpn-aliases/{svm.uuid}/{alias}

Fibre Channel Protocol services

A Fibre Channel Protocol (FCP) service defines the properties of a Fibre Channel target for an SVM.

- /protocol/san/fcp/services
- /protocol/san/fcp/services/{svm.uuid}

igroups

An initiator group (igroup) is a collection of Fibre Channel WWPNs (world wide port names), and iSCSI IQNs (qualified names), and iSCSI EUIs (extended unique identifiers) that identify host initiators.

- /protocol/san/igroups
- /protocol/san/igroups/{uuid}
- /protocol/san/igroups/{igroup.uuid}/initiators
- /protocol/san/igroups/{igroup.uuid}/initiators/{name}

iSCSI credentials

The iSCSI credentials object contains authentication credentials which are used by an initiator and ONTAP.

- /protocol/san/iscsi/credentials
- /protocol/san/iscsi/credentials/{svm.uuid}/{initiator}
iSCSI services
An iSCSI service defines the properties of the iSCSI target for an SVM.

- /protocol/san/iscsi/services
- /protocol/san/iscsi/services/{svm.uuid}

iSCSI sessions
An iSCSI session is one or more TCP connections that link an iSCSI initiator with an iSCSI target.

- /protocol/san/iscsi/sessions
- /protocol/san/iscsi/sessions/{svm.uuid}/{tpgroup}/{tsih}

LUN maps
A LUN map is an association between a LUN and an initiator group.

- /protocol/san/lun-maps
- /protocol/san/lun-maps/{lun.uuid}/{igroup.uuid}

LUNs
A LUN is the logical representation of storage in a storage area network (SAN).

- /storage/luns
- /storage/luns/{uuid}

Security
These API calls can be used to manage the cluster and SVM security settings.

Accounts
There is a collection of user accounts for the cluster and SVMs.

- /security/accounts

Accounts name
The configuration for a scoped user account.

- /security/accounts/{owner.uuid}/{name}

Audit
The settings which determine what is logged to the audit log files.

- /security/audit

Audit destinations
These settings control how audit log information is forwarded to remote systems or splunk servers.

- /security/audit/destinations
- /security/audit/destinations/{address}/{port}
Audit messages
You can retrieve the audit log messages.
• /security/audit/messages

LDAP authentication
These settings are used to retrieve and manage the cluster LDAP server configuration.
• /security/authentication/cluster/ldap

NIS authentication
These settings are used to retrieve and manage the cluster NIS server configuration.
• /security/authentication/cluster/nis

SAML service provider
You can display and manage the configuration for the SAML service provider.
• /security/authentication/cluster/saml-sp

Password authentication
This includes the API call used to change the password for a user account.
• /security/authentication/password

Certificates
The APIs calls can be used to install, display, and delete certificates used by ONTAP.
• /security/certificates
• /security/certificates/{uuid}
• /security/certificates/{ca.uuid}/sign

Key managers
A key manager allows client modules within ONTAP to securely stored keys.
• /security/key-managers
• /security/key-managers/{uuid}
• /security/key-managers/{uuid}/key-servers
• /security/key-managers/{uuid}/key-servers/{server}

Login messages
Used to display and manage the login messages used by ONTAP.
• /security/login/messages
• /security/login/messages/{uuid}

Roles
The roles provide a way to assign privileges to user accounts.
• /security/roles

Roles instance
Specific instance of a role.
• /security/roles/{owner.uuid}/name

Privileges for a role instance
Manage the privileges for a specific role.
• /security/roles/{owner.uuid}/name/privileges
• /security/roles/{owner.uuid}/name/privileges/{path}

SnapMirror
You can use these API calls to manage the SnapMirror data protection technology.

Policies
The SnapMirror policies are applied to relationships, and control the configuration attributes and behavior of each relationship.
• /snapmirror/policies
• /snapmirror/policies/{uuid}

Relationships
Both asynchronous and synchronous relationships establish the connectivity needed to transfer data.
• /snapmirror/relationships
• /snapmirror/relationships/{uuid}

Relationships transfers
You can manage the SnapMirror transfers over existing SnapMirror relationships.
• /snapmirror/relationships/{relationship.uuid}/transfers
• /snapmirror/relationships/{relationship.uuid}/transfers/{uuid}

Storage
You can use these API calls to manage the physical and logical storage.

Aggregates
An aggregate consists of one or more RAID groups.
• /storage/aggregates
• /storage/aggregates/{uuid}
• /storage/aggregates/{aggregate.uuid}/cloud-stores
• /storage/aggregates/{aggregate.uuid}/cloud-stores/{target.uuid}
**Aggregate plexes**
A physical copy of the WAFL storage within an aggregate.
- `/storage/aggregates/{aggregate.uuid}/plexes`
- `/storage/aggregates/{aggregate.uuid}/plexes/{name}`

**Disks**
The physical disks in the cluster.
- `/storage/disks`
- `/storage/disks/{name}`

**FlexCaches**
The persistent cache of an origin volume.
- `/storage/flexcache/flexcaches`
- `/storage/flexcache/flexcaches/{uuid}`

**FlexCache origins**
Origin volume of a FlexCache.
- `/storage/flexcache/origins`
- `/storage/flexcache/origins/{uuid}`

**Ports**
Storage ports of the cluster.
- `/storage/ports`
- `/storage/ports/{node.uuid}/{name}`

**QOS policies**
Quality of service policy configuration.
- `/storage/qos/policies`
- `/storage/qos/policies/{policy.uuid}`

**Qtrees**
A logically divided file system.
- `/storage/qtrees`
- `/storage/qtrees/{volume.uuid}/{id}`

**Quota reports**
Report on quotas, which is a technique for restricting or tracking files or space usage.
- `/storage/quota/reports`
- `/storage/quota/reports/{volume.uuid}/{index}`
Quota rules
The rules used to enforce the quotas.
- /storage/quota/rules
- /storage/quota/rules/{rule.uuid}

Shelves
Shelves in the cluster.
- /storage/shelves
- /storage/shelves/{uid}

Snapshot policies
Snapshots are created based on policies.
- /storage/snapshot-policies
- /storage/snapshot-policies/{uuid}

Volumes
Logical containers used to serve data to clients.
- /storage/volumes
- /storage/volumes/{uuid}
- /storage/volumes/{uuid}/metrics

Volumes Snapshots
Snapshots for a volume.
- /storage/volumes/{volume.uuid}/snapshots
- /storage/volumes/{volume.uuid}/snapshots/{uuid}

Support
You can use these API calls to manage the ONTAP features used to support a cluster.

AutoSupport
AutoSupport collects configuration and status details as well as errors, and reports the information to NetApp.
- /support/autosupport

AutoSupport messages
Each node maintains AutoSupport messages that can be generated and retrieved.
- /support/autosupport/messages

Configuration backup
The settings control how ONTAP configuration data is backed up.
**EMS**
The event management system (EMS) collects events and sends notifications to one or more destinations.

- /support/ems

**EMS destinations**
The EMS destinations determine how and where notifications are sent.

- /support/ems/destinations

**EMS destinations instance**
An EMS destination instance is defined by type and location.

- /support/ems/destinations/{name}

**EMS events**
This is a live collection of system events for the cluster.

- /support/ems/events

**EMS filters**
The EMS filters collectively identify the events that require additional processing.

- /support/ems/filters

**EMS filters instance**
An EMS filter instance is a collection of rules that are applied to the events.

- /support/ems/filters/{name}

**EMS rules for filter instance**
A list of rules can be managed for a specific instance of an EMS filter.

- /support/ems/filters/{name}/rules

**EMS rules instance for filter instance**
An individual rule for a specific instance of an EMS filter.

- /support/ems/filters/{name}/rules/{index}

**EMS messages**
Provides access to the EMS event catalog.

- /support/ems/messages
SVM

You can use these API calls to manage storage virtual machines (SVMs).

**Peer permissions**

Peer permissions can be assigned which enable the SVM peering relationships.

- `/svm/peer-permissions`
- `/svm/peer-permissions/{cluster.uuid}/{svm.uuid}`

**Peers**

The peering relationships establish connectivity among the SVMs.

- `/svm/peers`
- `/svm/peers/{peer.uuid}`

**SVMs**

You can manage the SVMs that are bound to a cluster.

- `/svm/svms`
- `/svm/svms/{uuid}`
Workflow processes using the ONTAP REST API

You can use the ONTAP REST API to administer ONTAP systems and perform other storage administrative tasks. Each of the tasks is described using a workflow process consisting of one or more API calls.

Preparing to use the workflow processes

You should prepare before using the workflow processes.

Understanding the API calls used in the workflows

The ONTAP API online documentation page includes the details of every REST API call. Rather than repeat those details here, each API call used in the workflow samples includes only the information you need to locate the call on the documentation page. After locating a specific API call, you can review the complete details of the call, including the input parameters, output formats, HTTP status codes, and request processing type.

The following information is included for each API call within a workflow to help locate the call on the documentation page:

- **Category**
  The API calls are organized on the documentation page into functionally related areas or categories. To locate a specific API call, scroll to the bottom of the page and click the applicable API category.

- **HTTP method**
  The HTTP method identifies the action performed on a resource. Each API call is executed through a single HTTP method.

- **Path**
  The path determines the specific resource which the action applies to as part of performing a call. The path string is appended to the core URL to form the complete URL identifying the resource.

Constructing a URL to directly access the REST API

In addition to the ONTAP documentation page, you can also access the ONTAP REST API directly through a programming language such as Python. In this case, the core URL is slightly different than the URL used when accessing the online documentation page. When accessing the API directly, you must append /api to the domain and port string. For example:

https://ontap.mycompany.com/api

Related concepts

*How to access the ONTAP API* on page 7

Related references

*ONTAP REST API resource categories* on page 22
Workflow 1: Updating the cluster contact and checking job state

This workflow updates the cluster contact information. Because the request is processed asynchronously, it also illustrates how to determine if the background job completes successfully.

1. Update the contact information for the cluster

You can issue an API call to update the contact information. Because the request runs asynchronously, you must confirm success of the associated job.

<table>
<thead>
<tr>
<th>Category</th>
<th>HTTP method</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>PATCH</td>
<td>/cluster</td>
</tr>
</tbody>
</table>

Processing type

Asynchronous

Curl

curl -X PATCH -H 'Content-Type: application/json' -u admin:password -k -d @step01 'https://10.222.81.101/api/cluster'

JSON input (step01)

```json
{
  "contact": "support@company-demo.com"
}
```

JSON output

A job object with the associated identifier is returned.

```json
{
  "job": {
    "uuid": "d877f5bb-3aa7-11e9-b6c6-005056a78c89",
    "_links": {
      "self": {
        "href": "/api/cluster/jobs/d877f5bb-3aa7-11e9-b6c6-005056a78c89"
      }
    }
  }
}
```

2. Retrieve the status of the job

You must retrieve the state of the job to determine if the asynchronous request has completed successfully.

<table>
<thead>
<tr>
<th>Category</th>
<th>HTTP method</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster</td>
<td>GET</td>
<td>/cluster/jobs/{uuid}</td>
</tr>
</tbody>
</table>

Processing type

Synchronous
Curl

curl -X GET -H 'Content-Type: application/json' -u admin:password -k 'https://10.222.81.101/api/cluster/jobs/uuid'

Output

A job object with the state and other fields is returned.

```
{
    "uuid": "d877f5bb-3aa7-11e9-b6c6-005056a78c89",
    "description": "PATCH /api/cluster",
    "state": "success",
    "message": "success",
    "code": 0,
    "_links": {
        "self": {
            "href": "/api/cluster/jobs/d877f5bb-3aa7-11e9-b6c6-005056a78c89"
        }
    }
}
```
ONTAP REST API transactions illustrated

Each API transaction consists of an HTTP request and the associated response. You can review the following transaction examples to improve your understanding of how the ONTAP REST API works.

Retrieving the cluster software version

This illustration shows how to retrieve the version of the ONTAP software used by the cluster.

HTTP request

The request sent from the client to the server consists of the following:

- GET verb
- URL path for the cluster
- Query parameter (fields)
- Request headers, including authorization

HTTP response

The response sent from the server to the client consists of the following:

- Status code 200
• Response headers
• Response body containing the cluster software version
Where to find additional information

You should review the following additional resources as part of using the ONTAP REST API to administer ONTAP clusters and the related storage resources.

ONTAP 9 resources

- **ONTAP 9 Documentation Center**
  Provides all of the documentation for ONTAP 9.

- **NetApp ONTAP Resources**
  Provides documentation and other helpful links needed to plan, administer, and support ONTAP 9.

NetApp resources

- **NetApp Support**
  Access troubleshooting tools, documentation, and technical support assistance.

- **NetApp Interoperability Matrix Tool**
  Access requirements and compatibility information related to using ONTAP 9 and the ONTAP REST API.

- **NetApp library of technical reports and white papers**
  Access technical reports, white papers, and other documents.
Copyright

Copyright © 2019 NetApp, Inc. All rights reserved. Printed in the U.S.

No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

Data contained herein pertains to a commercial item (as defined in FAR 2.101) and is proprietary to NetApp, Inc. The U.S. Government has a non-exclusive, non-transferrable, non-sublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b).
Trademark

NETAPP, the NETAPP logo, and the marks listed on the NetApp Trademarks page are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.

How to send comments about documentation and receive update notifications

You can help us to improve the quality of our documentation by sending us your feedback. You can receive automatic notification when production-level (GA/FCS) documentation is initially released or important changes are made to existing production-level documents.

If you have suggestions for improving this document, send us your comments by email.

doccomments@netapp.com

To help us direct your comments to the correct division, include in the subject line the product name, version, and operating system.

If you want to be notified automatically when production-level documentation is released or important changes are made to existing production-level documents, follow Twitter account @NetAppDoc.

You can also contact us in the following ways:

• NetApp, Inc., 1395 Crossman Ave., Sunnyvale, CA 94089 U.S.
• Telephone: +1 (408) 822-6000
• Fax: +1 (408) 822-4501
• Support telephone: +1 (888) 463-8277