Concepts

ONTAP Select

NetApp
November 08, 2026

This PDF was generated from https://docs.netapp.com/us-en/ontap-select-9191/concept_api_rest.html on November 08, 2026. Always check docs.netapp.com for the latest.

Table of Contents

Concepts . . . 1
- Learn about REST web services for deploying and managing ONTAP Select clusters . . . 1
  - Architecture and classic constraints . . . 1
  - Resources and state representation . . . 1
  - URI endpoints . . . 1
  - HTTP messages . . . 1
  - JSON formatting . . . 2
- Learn about accessing the ONTAP Select Deploy API . . . 2
  - Deploy utility native user interface . . . 2
  - ONTAP Select Deploy online documentation page . . . 2
  - Custom program . . . 2
- ONTAP Select Deploy API basic operational characteristics . . . 2
  - Hypervisor host versus ONTAP Select node . . . 2
  - Object identifiers . . . 3
  - Request identifiers . . . 3
  - Synchronous and asynchronous calls . . . 3
  - Confirm the completion of a long-running job . . . 3
  - Security . . . 4
- Request and response API transaction for ONTAP Select . . . 4
  - Input variables controlling an API request . . . 4
  - Interpret an API response . . . 6
- Asynchronous processing using the Job object for ONTAP Select . . . 7
  - Asynchronous requests described using Job object . . . 7
  - Query the Job object associated with an API request . . . 7
  - General procedure for issuing an asynchronous request . . . 8

Concepts

Learn about REST web services for deploying and managing ONTAP Select clusters

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 deploying and managing ONTAP Select clusters.

Architecture and classic constraints

REST was formally articulated by Roy Fielding in his PhD dissertation at UC Irvine in 2000. It defines an architectural style through a set of constraints, which collectively have improves web-based applications and the underlying protocols. The constraints establish a RESTful web services application based on a client/server architecture using a stateless communication protocol.

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.

Messages are exchanged between the client and server to access and change the state of the resources according to the generic CRUD (Create, Read, Update, and Delete) model.

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 verbs (such as GET and POST) are mapped to the resources and corresponding state management actions.

HTTP is stateless. Therefore, to associate a set of related requests and responses under one transaction, additional information must be included in the HTTP headers carried with the request/response data flows.

JSON formatting

While information can be structured and transferred between a client and server in several ways, the most popular option (and the one used with the Deploy REST API) 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.

Learn about accessing the ONTAP Select Deploy API

Because of the inherent flexibility of REST web services, the ONTAP Select Deploy API can be accessed in several different ways.

> The REST API included with ONTAP Select Deploy is assigned a version number. The API version number is independent of the Deploy release number. The ONTAP Select 9.19.1 Deploy administration utility includes version 3 of the REST API.

Deploy utility native user interface

The primary way you access the API is through the ONTAP Select Deploy web user interface. The browser makes calls to the API and reformats the data according to the design of the user interface. You also access the API through the Deploy utility command line interface.

ONTAP Select Deploy online documentation page

The ONTAP Select Deploy online documentation page provides an alternative access point when using a browser. In addition to providing a way to execute individual API calls directly, the page also includes a detailed description of the API, including input parameters and other options for each call. The API calls are organized into several different functional areas or categories.

Custom program

You can access the Deploy API using any of several different programming languages and tools. Popular choices include Python, Java, and cURL. A program, script, or tool that uses the API acts as a REST web services client. Using a programming language allows you to better understand the API and provides an opportunity to automate the ONTAP Select deployments.

ONTAP Select Deploy API 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. You should be aware of the details and operational characteristics of the ONTAP Select Deploy API before using the API.

Hypervisor host versus ONTAP Select node

A *hypervisor host* is the core hardware platform that hosts an ONTAP Select virtual machine. When an ONTAP Select virtual machine is deployed and active on a hypervisor host, the virtual machine is considered to be an *ONTAP Select node*. With version 3 of the Deploy REST API, the host and node objects are separate and distinct. This allows a one-to-many relationship, where one or more ONTAP Select nodes can run on the same hypervisor host.

Object identifiers

Each resource instance or object is assigned a unique identifier when it is created. These identifiers are globally unique within a specific instance of ONTAP Select Deploy. After issuing an API call that creates a new object instance, 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.



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.

Request identifiers

Every successful API request is assigned a unique identifier. The identifier is returned in the `request-id` header of the associated HTTP response. You can use a request identifier to collectively refer to the activities of a single specific API request-response transaction. For example, you can retrieve all the event messages for a transaction based on the request id.

Synchronous and asynchronous calls

There are two primary ways that a server performs an HTTP request received from a client:

- Synchronous
  The server performs the request immediately and responds with a status code of 200, 201, or 204.
- Asynchronous
  The server accepts the request and responds with a status code of 202. This indicates the server has accepted the client request and started a background task to complete the request. Final success or failure is not immediately available and must be determined through additional API calls.

Confirm the completion of a long-running job

Generally, any operation that can take a long time to complete is processed asynchronously using a background task at the server. With the Deploy REST API, every background task is anchored by a Job object which tracks the task and provides information, such as the current state. A Job object, including its unique identifier, is returned in the HTTP response after a background task is created.

You can query the Job object directly to determine the success or failure of the associated API call. Refer to *asynchronous processing using the Job object* for additional information.

In addition to using the Job object, there are other ways you can determine the success or failure of a request, including:

- Event messages
  You can retrieve all the event messages associated with a specific API call using the request id returned with the original response. The event messages typically contain an indication of success or failure, and can also be useful when debugging an error condition.
- Resource state or status
  Several of the resources maintain a state or status value which you can query to indirectly determine the success or failure of a request.

Security

The Deploy API uses the following security technologies:

- Transport Layer Security
  All traffic sent over the network between the Deploy server and client is encrypted through TLS. Using the HTTP protocol over an unencrypted channel is not supported. TLS version 1.2 is supported.
- HTTP authentication
  Basic authentication is used for every API transaction. An HTTP header, which includes the user name and password in a base64 string, is added to every request.

Request and response API transaction for ONTAP Select

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

Input variables controlling an API request

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

Request headers

You must include several headers in the HTTP request, including:

- content-type
  If the request body includes JSON, this header must be set to application/json.
- accept
  If the response body will include JSON, this header must be set to application/json.
- authorization
  Basic authentication must be set with the user name and password encoded in 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 (such as, the name of a new cluster)
- Empty

Filter 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, there are other operators available to return a set of objects over a range of values. ONTAP Select supports the filtering operators shown below.

Operator	Description
=	Equal to
<	Less than
>	Greater than
⇐	Less than or equal to
>=	Greater than or equal to
	Or
!	Not equal to
*	Greedy wildcard

You can also return a set 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.

Selecting object fields

By default, issuing an API call using GET returns only the attributes that uniquely identify the object or objects. 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:

- Inexpensive fields
  Specify `fields=*` to retrieve the object fields that are maintained in local server memory or require little processing to access.
- Expensive fields
  Specify `fields=**` to retrieve all the object fields, including those requiring additional server processing to access.
- Custom field selection
  Use `fields=FIELDNAME` to specify the exact field you want. When requesting multiple fields, the values must be separated using commas without spaces.



As a best practice, you should always identify the specific fields you want. You should only retrieve the set of inexpensive or expensive fields when needed. The inexpensive and expensive classification is determined by NetApp based on internal performance analysis. The classification for a given field can change at any time.

Sort 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`

When including multiple parameters, you must separate the fields with a comma.

Pagination

When issuing an API call using GET to access a collection of objects of the same type, all matching objects are returned by default. If needed, you can limit the number of records returned using the max_records query parameter with the request. For example:
`max_records=20`

If needed, you can combine this parameter with other query parameters to narrow the result set. For example, the following returns up to 10 system events generated after the specified time:
`time⇒ 2019-04-04T15:41:29.140265Z&max_records=10`

You can issue multiple requests to page through the events (or any object type). Each subsequent API call should use a new time value based on the latest event in the last result set.

Interpret an API response

Each API request generates a response back to the client. You can 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 Deploy REST API are described below.

Code	Meaning	Description
200	OK	Indicates success for calls that do not create a new object.
201	Created	An object is successfully created; the location response header includes the unique identifier for the object.
202	Accepted	A long-running background job has been started to perform the request, but the operation has not completed yet.
400	Bad request	The request input is not recognized or is inappropriate.
403	Forbidden	Access is denied due to an authorization error.
404	Not found	The resource referred to in the request does not exist.
405	Method not allowed	The HTTP verb in the request is not supported for the resource.
409	Conflict	An attempt to create an object failed because the object already exists.
500	Internal error	A general internal error occurred at the server.
501	Not implemented	The URI is known but is not capable of performing the request.

Response headers

Several headers are included in the HTTP response generated by the Deploy server, including:

- request-id
  Every successful API request is assigned a unique request identifier.
- location
  When an object is created, the location header includes the complete URL to the new object including the unique object identifier.

Response body

The content of the response associated with an API request differs based on the object, processing type, and the success or failure of the request. The response body is 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 all 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 POST request used to deploy a cluster 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 create a cluster with a name that already exists.
- Empty
  In certain cases, no data is returned and the response body is empty. For example, the response body is empty after using DELETE to delete an existing host.

Asynchronous processing using the Job object for ONTAP Select

Some of the Deploy API calls, particularly those that create or modify a resource, can take longer to complete than other calls. ONTAP Select Deploy processes these long-running requests asynchronously.

Asynchronous requests described using Job object

After making an API call that runs asynchronously, the HTTP response code 202 indicates the request has been successfully validated and accepted, but not yet completed. The request is processed as a background task which continues to run after the initial HTTP response to the client. The response includes the Job object anchoring the request, including its unique identifier.



You should refer to the ONTAP Select Deploy online documentation page to determine which API calls operate asynchronously.

Query 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 to determine if the request completed successfully. A Job object can be in one of the following states:

- Queued
- Running
- Success

- Failure

There are two techniques you can use when polling a Job object to detect a terminal state for the task, either success or failure:

- Standard polling request
  Current Job state is returned immediately
- Long polling request
  Job state is returned only when one of the following occurs:
  - State has changed more recently than the date-time value provided on the poll request
  - Timeout value has expired (1 to 120 seconds)

Standard polling and long polling use the same API call to query a Job object. However, a long polling request includes two query parameters: `poll_timeout` and `last_modified`.



You should always use long polling to reduce the workload on the Deploy virtual machine.

General procedure for issuing an asynchronous request

You can use the following high-level procedure to complete an asynchronous API call:

1. Issue the asynchronous API call.
2. Receive an HTTP response 202 indicating successful acceptance of the request.
3. Extract the identifier for the Job object from the response body.
4. Within a loop, perform the following in each cycle:
   a. Get the current state of the Job with a long-poll request
   b. If the Job is in a non-terminal state (queued, running), perform loop again.
5. Stop when the Job reaches a terminal state (success, failure).

Copyright information

Copyright © 2026 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.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, 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) (FEB 2014).

Trademark information

NETAPP, the NETAPP logo, and the marks listed at http://www.netapp.com/TM are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.