



Automate with REST

ONTAP Select

NetApp
February 03, 2026

This PDF was generated from https://docs.netapp.com/us-en/ontap-select-9151/concept_api_rest.html on February 03, 2026. Always check docs.netapp.com for the latest.

Table of Contents

Automate with REST	1
Concepts	1
REST web services foundation	1
How to access the Deploy API	2
Deploy API versioning	2
Basic operational characteristics	3
Request and response API transaction	4
Asynchronous processing using the job object	7
Access with a browser	8
Before you access the API with a browser	8
Access the Deploy documentation page	9
Understand and execute an API call	9
Workflow processes	10
Before you use the API workflows	10
Workflow 1: Create a single-node evaluation cluster on ESXi	10
Access with Python	17
Before you access the API using Python	17
Understand the Python scripts	17
Python code samples	19
Script to create a cluster	19
JSON for script to create a cluster	26
Script to add a node license	31
Script to delete a cluster	34
Common support module	36
Script to resize cluster nodes	41

Automate with REST

Concepts

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 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 improved 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.

How to access the Deploy API

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

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.

Deploy API versioning

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

The current release of the Deploy administration utility includes version 3 of the REST API. Past releases of the Deploy utility include the following API versions:

Deploy 2.8 and later

ONTAP Select Deploy 2.8 and all later releases include version 3 of the REST API.

Deploy 2.7.2 and earlier

ONTAP Select Deploy 2.7.2 and all earlier releases include version 2 of the REST API.

 Versions 2 and 3 of the REST API are not compatible. If you upgrade to Deploy 2.8 or later from an earlier release that includes version 2 of the API, you must update any existing code that directly accesses the API as well as any scripts using the command line interface.

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

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 <code>location</code> 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

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).

Access with a browser

Before you access the API with a browser

There are several things you should be aware of before using the Deploy online documentation page.

Deployment plan

If you intend to issue API calls as part of performing specific deployment or administrative tasks, you should

consider creating a deployment plan. These plans can be formal or informal, and generally contain your goals and the API calls to be used. Refer to Workflow processes using the Deploy REST API for more information.

JSON examples and parameter definitions

Each API call is described on the documentation page using a consistent format. The content includes implementation notes, query parameters, and HTTP status codes. In addition, you can display details about the JSON used with the API requests and responses as follows:

- Example Value

If you click *Example Value* on an API call, a typical JSON structure for the call is displayed. You can modify the example as needed and use it as input for your request.

- Model

If you click *Model*, a complete list of the JSON parameters is displayed, with a description for each parameter.

Caution when issuing API calls

All API operations you perform using the Deploy documentation page are live operations. You should be careful not to mistakenly create, update, or delete configuration or other data.

Access the Deploy documentation page

You must access the ONTAP Select Deploy online documentation page to display the API documentation, as well as to manually issue an API call.

Before you begin

You must have the following:

- IP address or domain name of the ONTAP Select Deploy virtual machine
- User name and password for the administrator

Steps

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

`https://<ip_address>/api/ui`

2. Sign in using the administrator user name and password.

Result

The Deploy documentation web page is displayed with the calls organized by category at the bottom of the page.

Understand and execute an API call

The details of all the API calls are documented and displayed using a common format on the ONTAP Select Deploy online documentation web page. By understanding a single API call, you can access and interpret the details of all the API calls.

Before you begin

You must be signed in to the ONTAP Select Deploy online documentation web page. You must have the

unique identifier assigned to your ONTAP Select cluster when the cluster was created.

About this task

You can retrieve the configuration information describing an ONTAP Select cluster using its unique identifier. In this example, all fields classified as inexpensive are returned. However, as a best practice you should request only the specific fields that are needed.

Steps

1. On the main page, scroll to the bottom and click **Cluster**.
2. Click **GET /clusters/{cluster_id}** to display the details of the API call used to return information about an ONTAP Select cluster.

Workflow processes

Before you use the API workflows

You should prepare to review and use the workflow processes.

Understand the API calls used in the workflows

The ONTAP Select 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 verb
The HTTP verb identifies the action performed on a resource. Each API call is executed through a single HTTP verb.
- 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.

Construct a URL to directly access the REST API

In addition to the ONTAP Select documentation page, you can also access the Deploy 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:

`http://deploy.mycompany.com/api`

Workflow 1: Create a single-node evaluation cluster on ESXi

You can deploy a single-node ONTAP Select cluster on a VMware ESXi host managed by vCenter. The cluster is created with an evaluation license.

The cluster creation workflow differs in the following situations:

- The ESXi host is not managed by vCenter (standalone host)
- Multiple nodes or hosts are used within the cluster
- Cluster is deployed in a production environment with a purchased license
- The KVM hypervisor is used instead of VMware ESXi

1. Register vCenter server credential

When deploying to an ESXi host managed by a vCenter server, you must add a credential before registering the host. The Deploy administration utility can then use the credential to authenticate to vCenter.

Category	HTTP verb	Path
Deploy	POST	/security/credentials

Curl

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k  
-d @step01 'https://10.21.191.150/api/security/credentials'
```

JSON input (step01)

```
{  
  "hostname": "vcenter.company-demo.com",  
  "type": "vcenter",  
  "username": "misteradmin@vsphere.local",  
  "password": "mypassword"  
}
```

Processing type

Asynchronous

Output

- Credential ID in the location response header
- Job object

2. Register a hypervisor host

You must add a hypervisor host where the virtual machine containing the ONTAP Select node will run.

Category	HTTP verb	Path
Cluster	POST	/hosts

Curl

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k  
-d @step02 'https://10.21.191.150/api/hosts'
```

JSON input (step02)

```
{  
  "hosts": [  
    {  
      "hypervisor_type": "ESX",  
      "management_server": "vcenter.company-demo.com",  
      "name": "esx1.company-demo.com"  
    }  
  ]  
}
```

Processing type

Asynchronous

Output

- Host ID in the location response header
- Job object

3. Create a cluster

When you create an ONTAP Select cluster, the basic cluster configuration is registered and the node names are automatically generated by Deploy.

Category	HTTP verb	Path
Cluster	POST	/clusters

Curl

The query parameter node_count should be set to 1 for a single-node cluster.

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k  
-d @step03 'https://10.21.191.150/api/clusters? node_count=1'
```

JSON input (step03)

```
{  
  "name": "my_cluster"  
}
```

Processing type

Synchronous

Output

- Cluster ID in the location response header

4. Configure the cluster

There are several attributes you must provide as part of configuring the cluster.

Category	HTTP verb	Path
Cluster	PATCH	/clusters/{cluster_id}

Curl

You must provide the cluster ID.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k -d @step04 'https://10.21.191.150/api/clusters/CLUSTERID'
```

JSON input (step04)

```
{  
  "dns_info": {  
    "domains": ["lab1.company-demo.com"],  
    "dns_ips": ["10.206.80.135", "10.206.80.136"]  
  },  
  "ontap_image_version": "9.5",  
  "gateway": "10.206.80.1",  
  "ip": "10.206.80.115",  
  "netmask": "255.255.255.192",  
  "ntp_servers": {"10.206.80.183"}
```

Processing type

Synchronous

Output

None

5. Retrieve the node name

The Deploy administration utility automatically generates the node identifiers and names when a cluster is created. Before you can configure a node, you must retrieve the assigned ID.

Category	HTTP verb	Path
Cluster	GET	/clusters/{cluster_id}/nodes

Curl

You must provide the cluster ID.

```
curl -iX GET -u admin:<password> -k  
'https://10.21.191.150/api/clusters/CLUSTERID/nodes?fields=id,name'
```

Processing type

Synchronous

Output

- Array records each describing a single node with the unique ID and name

6. Configure the nodes

You must provide the basic configuration for the node, which is the first of three API calls used to configure a node.

Category	HTTP verb	Path
Cluster	PATH	/clusters/{cluster_id}/nodes/{node_id}

Curl

You must provide the cluster ID and node ID.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k  
-d @step06 'https://10.21.191.150/api/clusters/CLUSTERID/nodes/NODEID'
```

JSON input (step06)

You must provide the host ID where the ONTAP Select node will run.

```
{  
  "host": {  
    "id": "HOSTID"  
  },  
  "instance_type": "small",  
  "ip": "10.206.80.101",  
  "passthrough_disks": false  
}
```

Processing type

Synchronous

Output

None

7. Retrieve the node networks

You must identify the data and management networks used by the node in the single-node cluster. The internal network is not used with a single-node cluster.

Category	HTTP verb	Path
Cluster	GET	/clusters/{cluster_id}/nodes/{node_id}/networks

Curl

You must provide the cluster ID and node ID.

```
curl -iX GET -u admin:<password> -k 'https://10.21.191.150/api/clusters/CLUSTERID/nodes/NODEID/networks?fields=id,purpose'
```

Processing type

Synchronous

Output

- Array of two records each describing a single network for the node, including the unique ID and purpose

8. Configure the node networking

You must configure the data and management networks. The internal network is not used with a single-node cluster.



Issue the following API call two times, once for each network.

Category	HTTP verb	Path
Cluster	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

You must provide the cluster ID, node ID, and network ID.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k -d @step08 'https://10.21.191.150/api/clusters/CLUSTERID/nodes/NODEID/networks/NETWORKID'
```

JSON input (step08)

You need to provide the name of the network.

```
{
  "name": "sDOT_Network"
}
```

Processing type

Synchronous

Output

None

9. Configure the node storage pool

The final step in configuring a node is to attach a storage pool. You can determine the available storage pools through the vSphere web client, or optionally through the Deploy REST API.

Category	HTTP verb	Path
Cluster	PATCH	/clusters/{cluster_id}/nodes/{node_id}/networks/{network_id}

Curl

You must provide the cluster ID, node ID, and network ID.

```
curl -iX PATCH -H 'Content-Type: application/json' -u admin:<password> -k  
-d @step09 'https://10.21.191.150/api/clusters/ CLUSTERID/nodes/NODEID'
```

JSON input (step09)

The pool capacity is 2 TB.

```
{  
  "pool_array": [  
    {  
      "name": "sDOT-01",  
      "capacity": 2147483648000  
    }  
  ]  
}
```

Processing type

Synchronous

Output

None

10. Deploy the cluster

After the cluster and node have been configured, you can deploy the cluster.

Category	HTTP verb	Path
Cluster	POST	/clusters/{cluster_id}/deploy

Curl

You must provide the cluster ID.

```
curl -iX POST -H 'Content-Type: application/json' -u admin:<password> -k  
-d @step10 'https://10.21.191.150/api/clusters/CLUSTERID/deploy'
```

JSON input (step10)

You must provide the password for the ONTAP administrator account.

```
{  
  "ontap_credentials": {  
    "password": "mypassword"  
  }  
}
```

Processing type

Asynchronous

Output

- Job object

Access with Python

Before you access the API using Python

You must prepare the environment before running the sample Python scripts.

Before you run the Python scripts, you must make sure the environment is configured properly:

- The latest applicable version of Python2 must be installed.
The sample codes have been tested using Python2. They should also be portable to Python3, but have not been tested for compatibility.
- The Requests and urllib3 libraries must be installed.
You can use pip or another Python management tool as appropriate for your environment.
- The client workstation where the scripts run must have network access to the ONTAP Select Deploy virtual machine.

In addition, you must have the following information:

- IP address of the Deploy virtual machine
- User name and password of a Deploy administrator account

Understand the Python scripts

The sample Python scripts allow you to perform several different tasks. You should understand the scripts before using them at a live Deploy instance.

Common design characteristics

The scripts have been designed with the following common characteristics:

- Execute from command line interface at a client machine
You can run the Python scripts from any properly configured client machine. See *Before you begin* for more information.
- Accept CLI input parameters
Each script is controlled at the CLI through input parameters.
- Read input file
Each script reads an input file based on its purpose. When creating or deleting a cluster, you must provide a JSON configuration file. When adding a node license, you must provide a valid license file.
- Use a common support module
The common support module *deploy_requests.py* contains a single class. It is imported and used by each of the scripts.

Create a cluster

You can create an ONTAP Select cluster using the script *cluster.py*. Based on the CLI parameters and contents of the JSON input file, you can modify the script to your deployment environment as follows:

- Hypervisor
You can deploy to ESXi or KVM (depending on the Deploy release). When deploying to ESXi, the hypervisor can be managed by vCenter or can be a standalone host.
- Cluster size
You can deploy a single-node or multiple-node cluster.
- Evaluation or production license
You can deploy a cluster with an evaluation or purchased license for production.

The CLI input parameters for the script include:

- Host name or IP address of the Deploy server
- Password for the admin user account
- Name of the JSON configuration file
- Verbose flag for message output

Add a node license

If you choose to deploy a production cluster, you must add a license for each node using the script *add_license.py*. You can add the license before or after you deploy the cluster.

The CLI input parameters for the script include:

- Host name or IP address of the Deploy server
- Password for the admin user account
- Name of the license file
- ONTAP user name with privileges to add the license
- Password for the ONTAP user

Delete a cluster

You can delete an existing ONTAP Select cluster using the script *delete_cluster.py*.

The CLI input parameters for the script include:

- Host name or IP address of the Deploy server
- Password for the admin user account
- Name of the JSON configuration file

Python code samples

Script to create a cluster

You can use the following script to create a cluster based on parameters defined within the script and a JSON input file.

```
1 #!/usr/bin/env python
2  #####-----#
3  #
4  # File: cluster.py
5  #
6  # (C) Copyright 2019 NetApp, Inc.
7  #
8  # This sample code is provided AS IS, with no support or warranties of
9  # any kind, including but not limited for warranties of
10 # merchantability
11 # or fitness of any kind, expressed or implied. Permission to use,
12 # reproduce, modify and create derivatives of the sample code is
13 # granted
14 # solely for the purpose of researching, designing, developing and
15 # testing a software application product for use with NetApp products,
16 # provided that the above copyright notice appears in all copies and
17 # that the software application product is distributed pursuant to
18 # terms
19 #
20 import traceback
21 import argparse
22 import json
23 import logging
24
25 from deploy_requests import DeployRequests
26
```

```
27 def add_vcenter_credentials(deploy, config):
28     """ Add credentials for the vcenter if present in the config """
29     log_debug_trace()
30
31
32     vcenter = config.get('vcenter', None)
33     if vcenter and not deploy.resource_exists('/security/credentials',
34                                                 'hostname', vcenter[
35         'hostname']):
36         log_info("Registering vcenter {} credentials".format(vcenter[
36         'hostname']))
37         data = {k: vcenter[k] for k in ['hostname', 'username',
38         'password']}
39         data['type'] = "vcenter"
40         deploy.post('/security/credentials', data)
41
42
43 def add_standalone_host_credentials(deploy, config):
44     """ Add credentials for standalone hosts if present in the config.
45         Does nothing if the host credential already exists on the
46         Deploy.
47     """
48     log_debug_trace()
49
50     hosts = config.get('hosts', [])
51     for host in hosts:
52         # The presense of the 'password' will be used only for
53         # standalone hosts.
54         # If this host is managed by a vcenter, it should not have a
55         host 'password' in the json.
56         if 'password' in host and not deploy.resource_exists(
57             '/security/credentials',
58             'hostname', host['name']):
59             log_info("Registering host {} credentials".format(host[
59             'name']))
60             data = {'hostname': host['name'], 'type': 'host',
61                     'username': host['username'], 'password': host[
62             'password']}
63             deploy.post('/security/credentials', data)
64
65
66 def register_unkown_hosts(deploy, config):
67     ''' Registers all hosts with the deploy server.
68         The host details are read from the cluster config json file.
69     '''
```

```

63         This method will skip any hosts that are already registered.
64         This method will exit the script if no hosts are found in the
65         config.
66         ''
67
68     data = {"hosts": []}
69     if 'hosts' not in config or not config['hosts']:
70         log_and_exit("The cluster config requires at least 1 entry in
71         the 'hosts' list got {}".format(config))
72
73     missing_host_cnt = 0
74     for host in config['hosts']:
75         if not deploy.resource_exists('/hosts', 'name', host['name']):
76             missing_host_cnt += 1
77             host_config = {"name": host['name'], "hypervisor_type":
78             host['type']}
79             if 'mgmt_server' in host:
80                 host_config["management_server"] = host['mgmt_server']
81                 log_info(
82                     "Registering from vcenter {mgmt_server}{}".format(
83                     **host))
84
85             if 'password' in host and 'user' in host:
86                 host_config['credential'] = {
87                     "password": host['password'], "username": host[
88                     'user']}
89
90             log_info("Registering {type} host {name}{}".format(**host))
91             data["hosts"].append(host_config)
92
93
94     def add_cluster_attributes(deploy, config):
95         ''' POST a new cluster with all needed attribute values.
96             Returns the cluster_id of the new config
97         '''
98         log_debug_trace()
99
100        cluster_config = config['cluster']
101        cluster_id = deploy.find_resource('/clusters', 'name',
102        cluster_config['name'])

```

```

103     if not cluster_id:
104         log_info("Creating cluster config named {name}".format(
105             **cluster_config))
106
107         # Filter to only the valid attributes, ignores anything else
108         # in the json
109         data = {k: cluster_config[k] for k in [
110             'name', 'ip', 'gateway', 'netmask', 'ontap_image_version',
111             'dns_info', 'ntp_servers']}
112
113         num_nodes = len(config['nodes'])
114
115         log_info("Cluster properties: {}".format(data))
116
117         resp = deploy.post('/v3/clusters?node_count={}'.format(
118             num_nodes), data)
119         cluster_id = resp.headers.get('Location').split('/')[-1]
120
121     return cluster_id
122
123
124
125
126
127
128
129 def get_node_ids(deploy, cluster_id):
130     ''' Get the the ids of the nodes in a cluster. Returns a list of
131     node_ids.'''
132     log_debug_trace()
133
134     response = deploy.get('/clusters/{}/nodes'.format(cluster_id))
135     node_ids = [node['id'] for node in response.json().get('records')]
136
137     return node_ids
138
139
140
141 def add_node_attributes(deploy, cluster_id, node_id, node):
142     ''' Set all the needed properties on a node '''
143     log_debug_trace()
144
145     log_info("Adding node '{}' properties".format(node_id))
146
147     data = {k: node[k] for k in ['ip', 'serial_number', 'instance_type',
148
149                     'is_storage_efficiency_enabled']} if k
150
151     in node}
152
153     # Optional: Set a serial_number
154     if 'license' in node:
155         data['license'] = {'id': node['license']}
156
157
158     # Assign the host

```

```

142     host_id = deploy.find_resource('/hosts', 'name', node['host_name']
143     ''])
144     if not host_id:
145         log_and_exit("Host names must match in the 'hosts' array, and
146         the nodes.host_name property")
147
148     # Set the correct raid_type
149     is_hw_raid = not node['storage'].get('disks') # The presence of a
150     list of disks indicates sw_raid
151     data['passthrough_disks'] = not is_hw_raid
152
153     # Optionally set a custom node name
154     if 'name' in node:
155         data['name'] = node['name']
156
157     log_info("Node properties: {}".format(data))
158     deploy.patch('/clusters/{}/nodes/{}'.format(cluster_id, node_id),
159     data)
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180

```

```

181         deploy.patch('/clusters/{}/nodes/{}/networks/{}'.format
182             (cluster_id, node_id, network_id), data)
183
184     def add_node_storage(deploy, cluster_id, node_id, node):
185         ''' Set all the storage information on a node '''
186         log_debug_trace()
187
188         log_info("Adding node '{}' storage properties".format(node_id))
189         log_info("Node storage: {}".format(node['storage']['pools']))
190
191         data = {'pool_array': node['storage']['pools']} # use all the
192             json properties
193         deploy.post(
194             '/clusters/{}/nodes/{}/storage/pools'.format(cluster_id,
195                 node_id), data)
196
197         if 'disks' in node['storage'] and node['storage']['disks']:
198             data = {'disks': node['storage']['disks']}
199             deploy.post(
200                 '/clusters/{}/nodes/{}/storage/disks'.format(cluster_id,
201                     node_id), data)
202
203     def create_cluster_config(deploy, config):
204         ''' Construct a cluster config in the deploy server using the
205             input json data '''
206         log_debug_trace()
207
208         cluster_id = add_cluster_attributes(deploy, config)
209
210         node_ids = get_node_ids(deploy, cluster_id)
211         node_configs = config['nodes']
212
213         for node_id, node_config in zip(node_ids, node_configs):
214             add_node_attributes(deploy, cluster_id, node_id, node_config)
215             add_node_networks(deploy, cluster_id, node_id, node_config)
216             add_node_storage(deploy, cluster_id, node_id, node_config)
217
218     def deploy_cluster(deploy, cluster_id, config):
219         ''' Deploy the cluster config to create the ONTAP Select VMs. '''
220         log_debug_trace()
221         log_info("Deploying cluster: {}".format(cluster_id))

```

```

222
223     data = {'ontap_credential': {'password': config['cluster'][
224         'ontap_admin_password']}}
224     deploy.post('/clusters/{}/deploy?inhibit_rollback=true'.format(
225         cluster_id),
225         data, wait_for_job=True)
226
227
228 def log_debug_trace():
229     stack = traceback.extract_stack()
230     parent_function = stack[-2][2]
231     logging.getLogger('deploy').debug('Calling %s()' %
231         parent_function)
232
233
234 def log_info(msg):
235     logging.getLogger('deploy').info(msg)
236
237
238 def log_and_exit(msg):
239     logging.getLogger('deploy').error(msg)
240     exit(1)
241
242
243 def configure_logging(verbose):
244     FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
245     if verbose:
246         logging.basicConfig(level=logging.DEBUG, format=FORMAT)
247     else:
248         logging.basicConfig(level=logging.INFO, format=FORMAT)
249         logging.getLogger('requests.packages.urllib3.connectionpool').
249             setLevel(
250                 logging.WARNING)
251
252
253 def main(args):
254     configure_logging(args.verbose)
255     deploy = DeployRequests(args.deploy, args.password)
256
257     with open(args.config_file) as json_data:
258         config = json.load(json_data)
259
260         add_vcenter_credentials(deploy, config)
261
262         add_standalone_host_credentials(deploy, config)
263

```

```

264     register_unkown_hosts(deploy, config)
265
266     cluster_id = create_cluster_config(deploy, config)
267
268     deploy_cluster(deploy, cluster_id, config)
269
270
271 def parseArgs():
272     parser = argparse.ArgumentParser(description='Uses the ONTAP
273     Select Deploy API to construct and deploy a cluster.')
274     parser.add_argument('-d', '--deploy', help='Hostname or IP address
275     of Deploy server')
276     parser.add_argument('-p', '--password', help='Admin password of
277     Deploy server')
278     parser.add_argument('-c', '--config_file', help='Filename of the
279     cluster config')
280     parser.add_argument('-v', '--verbose', help='Display extra
281     debugging messages for seeing exact API calls and responses',
282     action='store_true', default=False)
283
284     return parser.parse_args()
285
286
287 if __name__ == '__main__':
288     args = parseArgs()
289     main(args)

```

JSON for script to create a cluster

When creating or deleting an ONTAP Select cluster using the Python code samples, you must provide a JSON file as input to the script. You can copy and modify the appropriate JSON sample based on your deployment plans.

Single-node cluster on ESXi

```

1 {
2     "hosts": [
3         {
4             "password": "mypassword1",
5             "name": "host-1234",
6             "type": "ESX",
7             "username": "admin"
8         }
9     ],
10
11     "cluster": {
12         "dns_info": {

```

```

13     "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
14         "lab3.company-demo.com", "lab4.company-demo.com"
15     ],
16
17     "dns_ips": ["10.206.80.135", "10.206.80.136"]
18 },
19     "ontap_image_version": "9.7",
20     "gateway": "10.206.80.1",
21     "ip": "10.206.80.115",
22     "name": "mycluster",
23     "ntp_servers": ["10.206.80.183", "10.206.80.142"],
24     "ontap_admin_password": "mypassword2",
25     "netmask": "255.255.254.0"
26 },
27
28 "nodes": [
29     {
30         "serial_number": "3200000nn",
31         "ip": "10.206.80.114",
32         "name": "node-1",
33         "networks": [
34             {
35                 "name": "ontap-external",
36                 "purpose": "mgmt",
37                 "vlan": 1234
38             },
39             {
40                 "name": "ontap-external",
41                 "purpose": "data",
42                 "vlan": null
43             },
44             {
45                 "name": "ontap-internal",
46                 "purpose": "internal",
47                 "vlan": null
48             }
49         ],
50         "host_name": "host-1234",
51         "is_storage_efficiency_enabled": false,
52         "instance_type": "small",
53         "storage": {
54             "disk": [],
55             "pools": [
56                 {
57                     "name": "storage-pool-1",
58                     "capacity": 4802666790125

```

```

59         }
60     ]
61   }
62 ]
63 ]
64 }

```

Single-node cluster on ESXi using vCenter

```

{
  "hosts": [
    {
      "name": "host-1234",
      "type": "ESX",
      "mgmt_server": "vcenter-1234"
    }
  ],
  "cluster": {
    "dns_info": {"domains": ["lab1.company-demo.com", "lab2.company-
demo.com",
      "lab3.company-demo.com", "lab4.company-demo.com"
    ],
    "dns_ips": ["10.206.80.135", "10.206.80.136"]
  },
  "ontap_image_version": "9.7",
  "gateway": "10.206.80.1",
  "ip": "10.206.80.115",
  "name": "mycluster",
  "ntp_servers": ["10.206.80.183", "10.206.80.142"],
  "ontap_admin_password": "mypassword2",
  "netmask": "255.255.254.0"
},
  "vcenter": {
    "password": "mypassword2",
    "hostname": "vcenter-1234",
    "username": "selectadmin"
},
  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip": "10.206.80.114",

```

```

  "name": "node-1",
  "networks": [
    {
      "name": "ONTAP-Management",
      "purpose": "mgmt",
      "vlan": null
    },
    {
      "name": "ONTAP-External",
      "purpose": "data",
      "vlan": null
    },
    {
      "name": "ONTAP-Internal",
      "purpose": "internal",
      "vlan": null
    }
  ],
  "host_name": "host-1234",
  "is_storage_efficiency_enabled": false,
  "instance_type": "small",
  "storage": {
    "disk": [],
    "pools": [
      {
        "name": "storage-pool-1",
        "capacity": 5685190380748
      }
    ]
  }
}
]
}

```

Single-node cluster on KVM

```

{
  "hosts": [
    {
      "password": "mypassword1",
      "name": "host-1234",
      "type": "KVM",
      "username": "root"
    }
  ]
}

```

```

] ,


"cluster": {
  "dns_info": {
    "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
      "lab3.company-demo.com", "lab4.company-demo.com"
    ] ,
    "dns_ips": ["10.206.80.135", "10.206.80.136"]
  } ,


  "ontap_image_version": "9.7",
  "gateway": "10.206.80.1",
  "ip": "10.206.80.115",
  "name": "CBF4ED97",
  "ntp_servers": ["10.206.80.183", "10.206.80.142"],
  "ontap_admin_password": "mypassword2",
  "netmask": "255.255.254.0"
} ,


"nodes": [
  {
    "serial_number": "3200000nn",
    "ip": "10.206.80.115",
    "name": "node-1",
    "networks": [
      {
        "name": "ontap-external",
        "purpose": "mgmt",
        "vlan": 1234
      } ,
      {
        "name": "ontap-external",
        "purpose": "data",
        "vlan": null
      } ,
      {
        "name": "ontap-internal",
        "purpose": "internal",
        "vlan": null
      }
    ] ,
    "host_name": "host-1234",
    "is_storage_efficiency_enabled": false,
    "instance_type": "small",
    "storage": {

```

```

"disk": [],
"pools": [
    {
        "name": "storage-pool-1",
        "capacity": 4802666790125
    }
]
}
]
}
}

```

Script to add a node license

You can use the following script to add a license for an ONTAP Select node.

```

1#!/usr/bin/env python
2#####
3#
4# File: add_license.py
5#
6# (C) Copyright 2019 NetApp, Inc.
7#
8# This sample code is provided AS IS, with no support or warranties of
9# any kind, including but not limited for warranties of
# merchantability
10# or fitness of any kind, expressed or implied. Permission to use,
11# reproduce, modify and create derivatives of the sample code is
# granted
12# solely for the purpose of researching, designing, developing and
13# testing a software application product for use with NetApp products,
14# provided that the above copyright notice appears in all copies and
15# that the software application product is distributed pursuant to
# terms
16# no less restrictive than those set forth herein.
17#
18#####
19
20import argparse
21import logging
22import json
23
24from deploy_requests import DeployRequests
25
26

```

```

27 def post_new_license(deploy, license_filename):
28     log_info('Posting a new license: {}'.format(license_filename))
29
30     # Stream the file as multipart/form-data
31     deploy.post('/licensing/licenses', data={}, 
32                 files={'license_file': open(license_filename, 'rb')})
33
34     # Alternative if the NLF license data is converted to a string.
35     # with open(license_filename, 'rb') as f:
36     #     nlf_data = f.read()
37     #     r = deploy.post('/licensing/licenses', data={}, 
38     #                     files={'license_file': (license_filename,
39     # nlf_data)})
40
41
42 def put_license(deploy, serial_number, data, files):
43     log_info('Adding license for serial number: {}'.format(
44         serial_number))
45
46
47 def put_used_license(deploy, serial_number, license_filename,
48     ontap_username, ontap_password):
49     ''' If the license is used by an 'online' cluster, a
50     username/password must be given. '''
51
52     data = {'ontap_username': ontap_username, 'ontap_password':
53             ontap_password}
54     files = {'license_file': open(license_filename, 'rb')}
55
56     put_license(deploy, serial_number, data, files)
57
58
59 def put_free_license(deploy, serial_number, license_filename):
60     data = {}
61     files = {'license_file': open(license_filename, 'rb')}
62
63     put_license(deploy, serial_number, data, files)
64
65
66 def get_serial_number_from_license(license_filename):
67     ''' Read the NLF file to extract the serial number '''
68     with open(license_filename) as f:
69         data = json.load(f)

```

```

67
68     statusResp = data.get('statusResp', {})
69     serialNumber = statusResp.get('serialNumber')
70
71     if not serialNumber:
72         log_and_exit("The license file seems to be missing the
73         serialNumber")
74
75
76 def log_info(msg):
77     logging.getLogger('deploy').info(msg)
78
79
80 def log_and_exit(msg):
81     logging.getLogger('deploy').error(msg)
82     exit(1)
83
84
85 def configure_logging():
86     FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
87     logging.basicConfig(level=logging.INFO, format=FORMAT)
88     logging.getLogger('requests.packages.urllib3.connectionpool'
89     ).setLevel(logging.WARNING)
90
91 def main(args):
92     configure_logging()
93     serial_number = get_serial_number_from_license(args.license)
94
95     deploy = DeployRequests(args.deploy, args.password)
96
97     # First check if there is already a license resource for this
98     # serial-number
99     if deploy.find_resource('/licensing/licenses', 'id',
100         serial_number):
101
102         # If the license already exists in the Deploy server,
103         # determine if its used
104         if deploy.find_resource('/clusters', 'nodes.serial_number',
105             serial_number):
106
107             # In this case, requires ONTAP creds to push the license
108             # to the node
109             if args.ontap_username and args.ontap_password:
110                 put_used_license(deploy, serial_number, args.license,

```

```

106                                     args.ontap_username, args
107                                     .ontap_password)
108                                     else:
109                                         print("ERROR: The serial number for this license is in
110                                         use. Please provide ONTAP credentials.")
111                                         else:
112                                             # License exists, but its not used
113                                             put_free_license(deploy, serial_number, args.license)
114                                         else:
115                                             # No license exists, so register a new one as an available
116                                             license for later use
117                                             post_new_license(deploy, args.license)
118
119
120
121
122
123
124
125
126
127
128 if __name__ == '__main__':
129     args = parseArgs()
130     main(args)

```

Script to delete a cluster

You can use the following CLI script to delete an existing cluster.

```

1 #!/usr/bin/env python
2 #-----
3 #
4 # File: delete_cluster.py
5 #
6 # (C) Copyright 2019 NetApp, Inc.

```

```

7 #
8 # This sample code is provided AS IS, with no support or warranties of
9 # any kind, including but not limited for warranties of merchantability
10 # or fitness of any kind, expressed or implied. Permission to use,
11 # reproduce, modify and create derivatives of the sample code is
12 # granted
13 # solely for the purpose of researching, designing, developing and
14 # testing a software application product for use with NetApp products,
15 # provided that the above copyright notice appears in all copies and
16 # that the software application product is distributed pursuant to
17 # terms
18 # no less restrictive than those set forth herein.
19 #
20 #####-----#
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47

```

```

48
49 def configure_logging():
50     FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
51     logging.basicConfig(level=logging.INFO, format=FORMAT)
52     logging.getLogger('requests.packages.urllib3.connectionpool')
53         .setLevel(logging.WARNING)
54
55 def main(args):
56     configure_logging()
57     deploy = DeployRequests(args.deploy, args.password)
58
59     with open(args.config_file) as json_data:
60         config = json.load(json_data)
61
62         cluster_id = find_cluster(deploy, config['cluster']['name'])
63
64         log_info("Found the cluster {} with id: {}".format(config[
65             'cluster']['name'], cluster_id))
66
67         offline_cluster(deploy, cluster_id)
68
69         delete_cluster(deploy, cluster_id)
70
71 def parseArgs():
72     parser = argparse.ArgumentParser(description='Uses the ONTAP Select
73     Deploy API to delete a cluster')
74     parser.add_argument('-d', '--deploy', required=True, type=str,
75         help='Hostname or IP address of Deploy server')
76     parser.add_argument('-p', '--password', required=True, type=str,
77         help='Admin password of Deploy server')
78     parser.add_argument('-c', '--config_file', required=True, type=str,
79         help='Filename of the cluster json config')
80
81     return parser.parse_args()
82
83 if __name__ == '__main__':
84     args = parseArgs()
85     main(args)

```

Common support module

All of the Python scripts use a common Python class in a single module.

```
1 #!/usr/bin/env python
```



```

45         self.logger.debug('POST DATA: %s', data)
46         response = requests.post(self.base_url + path,
47                                     auth=self.auth, verify=False,
48                                     json=data,
49                                     headers=self.headers)
50
51         self.logger.debug('HEADERS: %s\nBODY: %s', self.
52             filter_headers(response), response.text)
53         self.exit_on_errors(response)
54
55     if wait_for_job and response.status_code == 202:
56         self.wait_for_job(response.json())
57     return response
58
59     def patch(self, path, data, wait_for_job=False):
60         self.logger.debug('PATCH DATA: %s', data)
61         response = requests.patch(self.base_url + path,
62                                     auth=self.auth, verify=False,
63                                     json=data,
64                                     headers=self.headers)
65         self.logger.debug('HEADERS: %s\nBODY: %s', self.
66             filter_headers(response), response.text)
67         self.exit_on_errors(response)
68
69     if wait_for_job and response.status_code == 202:
70         self.wait_for_job(response.json())
71     return response
72
73     def put(self, path, data, files=None, wait_for_job=False):
74         if files:
75             print('PUT FILES: {}'.format(data))
76             response = requests.put(self.base_url + path,
77                                     auth=self.auth, verify=False,
78                                     data=data,
79                                     files=files)
80         else:
81             self.logger.debug('PUT DATA:')
82             response = requests.put(self.base_url + path,
83                                     auth=self.auth, verify=False,
84                                     json=data,
85                                     headers=self.headers)
86
87             self.logger.debug('HEADERS: %s\nBODY: %s', self.
88             filter_headers(response), response.text)
89             self.exit_on_errors(response)

```

```

88         if wait_for_job and response.status_code == 202:
89             self.wait_for_job(response.json())
90     return response
91
92     def get(self, path):
93         """ Get a resource object from the specified path """
94         response = requests.get(self.base_url + path, auth=self.auth,
95         verify=False)
96         self.logger.debug('HEADERS: %s\nBODY: %s', self.
97         filter_headers(response), response.text)
98         self.exit_on_errors(response)
99     return response
100
101     def delete(self, path, wait_for_job=False):
102         """ Delete's a resource from the specified path """
103         response = requests.delete(self.base_url + path, auth=self.
104         .auth, verify=False)
105         self.logger.debug('HEADERS: %s\nBODY: %s', self.
106         filter_headers(response), response.text)
107         self.exit_on_errors(response)
108
109     def find_resource(self, path, name, value):
110         ''' Returns the 'id' of the resource if it exists, otherwise
111         None '''
112         resource = None
113         response = self.get('{path}?{field}={value}'.format(
114             path=path, field=name, value=value))
115         if response.status_code == 200 and response.json().get(
116             'num_records') >= 1:
117             resource = response.json().get('records')[0].get('id')
118     return resource
119
120     def get_num_records(self, path, query=None):
121         ''' Returns the number of records found in a container, or
122         None on error '''
123         resource = None
124         query_opt = '?{}'.format(query) if query else ''
125         response = self.get('{path}{query}'.format(path=path, query
126         =query_opt))
127         if response.status_code == 200 :
128             return response.json().get('num_records')
129     return None

```

```

126
127     def resource_exists(self, path, name, value):
128         return self.find_resource(path, name, value) is not None
129
130     def wait_for_job(self, response, poll_timeout=120):
131         last_modified = response['job']['last_modified']
132         job_id = response['job']['id']
133
134         self.logger.info('Event: ' + response['job']['message'])
135
136         while True:
137             response = self.get('/jobs/{}?fields=state,message&'
138                                 'poll_timeout={}&last_modified=>{}'.format(
139                                     job_id, poll_timeout,
140                                     last_modified))
141
142             job_body = response.json().get('record', {})
143
144             # Show interesting message updates
145             message = job_body.get('message', '')
146             self.logger.info('Event: ' + message)
147
148             # Refresh the last modified time for the poll loop
149             last_modified = job_body.get('last_modified')
150
151             # Look for the final states
152             state = job_body.get('state', 'unknown')
153             if state in ['success', 'failure']:
154                 if state == 'failure':
155                     self.logger.error('FAILED background job.\nJOB: %s', job_body)
156                     exit(1) # End the script if a failure occurs
157
158     def exit_on_errors(self, response):
159         if response.status_code >= 400:
160             self.logger.error('FAILED request to URL: %s\nHEADERS: %s\nRESPONSE BODY: %s',
161                               response.request.url,
162                               self.filter_headers(response),
163                               response.text)
164             response.raise_for_status() # Displays the response error,
165             and exits the script
166

```

```
167     def filter_headers(response):
168         ''' Returns a filtered set of the response headers '''
169         return {key: response.headers[key] for key in ['Location',
170 'request-id']} if key in response.headers}
```

Script to resize cluster nodes

You can use the following script to resize the nodes in an ONTAP Select cluster.

```
1 #!/usr/bin/env python
2 #####-----#
3 #
4 # File: resize_nodes.py
5 #
6 # (C) Copyright 2019 NetApp, Inc.
7 #
8 # This sample code is provided AS IS, with no support or warranties of
9 # any kind, including but not limited for warranties of
10 # merchantability
11 # or fitness of any kind, expressed or implied. Permission to use,
12 # reproduce, modify and create derivatives of the sample code is
13 # granted
14 # solely for the purpose of researching, designing, developing and
15 # testing a software application product for use with NetApp products,
16 # provided that the above copyright notice appears in all copies and
17 # that the software application product is distributed pursuant to
18 # terms
19 # no less restrictive than those set forth herein.
20 #
21 #####-----#
22 #
23
24 import argparse
25 import logging
26 import sys
27
28 from deploy_requests import DeployRequests
29
30
31 def _parse_args():
32     """ Parses the arguments provided on the command line when
33     executing this
34         script and returns the resulting namespace. If all required
35     arguments
36         are not provided, an error message indicating the mismatch is
37     printed and
```

```

31         the script will exit.
32     """
33
34     parser = argparse.ArgumentParser(description=
35         'Uses the ONTAP Select Deploy API to resize the nodes in the
36         cluster.'
37         ' For example, you might have a small (4 CPU, 16GB RAM per
38         node) 2 node'
39         ' cluster and wish to resize the cluster to medium (8 CPU,
40         64GB RAM per'
41         ' node). This script will take in the cluster details and then
42         perform'
43         ' the operation and wait for it to complete.'
44     )
45     parser.add_argument('--deploy', required=True, help=(
46         'Hostname or IP of the ONTAP Select Deploy VM.'
47     ))
48     parser.add_argument('--deploy-password', required=True, help=(
49         'The password for the ONTAP Select Deploy admin user.'
50     ))
51     parser.add_argument('--cluster', required=True, help=(
52         'Hostname or IP of the cluster management interface.'
53     ))
54     parser.add_argument('--instance-type', required=True, help=(
55         'The desired instance size of the nodes after the operation is
56         complete.'
57     ))
58     parser.add_argument('--ontap-password', required=True, help=(
59         'The password for the ONTAP administrative user account.'
60     ))
61     parser.add_argument('--ontap-username', default='admin', help=(
62         'The username for the ONTAP administrative user account.
63         Default: admin.'
64     ))
65     parser.add_argument('--nodes', nargs='+', metavar='NODE_NAME',
66     help=(
67         'A space separated list of node names for which the resize
68         operation'
69         ' should be performed. The default is to apply the resize to
70         all nodes in'
71         ' the cluster. If a list of nodes is provided, it must be
72         provided in HA'
73         ' pairs. That is, in a 4 node cluster, nodes 1 and 2
74         (partners) must be'
75         ' resized in the same operation.'
76     ))

```

```
66     return parser.parse_args()
67
68
69 def _get_cluster(deploy, parsed_args):
70     """ Locate the cluster using the arguments provided """
71
72     cluster_id = deploy.find_resource('/clusters', 'ip', parsed_args
73     .cluster)
74     if not cluster_id:
75         return None
76     return deploy.get('/clusters/%s?fields=nodes' % cluster_id).  

77         json()['record']
78
79
80 def _get_request_body(parsed_args, cluster):
81     """ Build the request body """
82
83     changes = {'admin_password': parsed_args.ontap_password}
84
85     # if provided, use the list of nodes given, else use all the nodes
86     # in the cluster
87     nodes = [node for node in cluster['nodes']]
88     if parsed_args.nodes:
89         nodes = [node for node in nodes if node['name'] in
90         parsed_args.nodes]
91
92     changes['nodes'] = [
93         {'instance_type': parsed_args.instance_type, 'id': node['id']}
94         for node in nodes]
95
96     return changes
97
98
99 def main():
100     """ Set up the resize operation by gathering the necessary data
101     and then send
102         the request to the ONTAP Select Deploy server.
103     """
104
105     logging.basicConfig(
106         format='[% (asctime)s] [% (levelname)5s] % (message)s', level
107         =logging.INFO,
108
109         logging.getLogger('requests.packages.urllib3').setLevel(logging
110         .WARNING)
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
```

```
104     parsed_args = _parse_args()  
105     deploy = DeployRequests(parsed_args.deploy, parsed_args  
106     .deploy_password)  
107     cluster = _get_cluster(deploy, parsed_args)  
108     if not cluster:  
109         deploy.logger.error(  
110             'Unable to find a cluster with a management IP of %s' %  
111             parsed_args.cluster)  
112         return 1  
113     changes = _get_request_body(parsed_args, cluster)  
114     deploy.patch('/clusters/%s' % cluster['id'], changes,  
115     wait_for_job=True)  
116 if __name__ == '__main__':  
117     sys.exit(main())
```

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—with 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.