Automate with REST

ONTAP Select

NetApp
December 03, 2020

Table of Contents

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 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.

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.

Confirming 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

Filtering objects

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

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

In addition to an exact match, 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.

Sorting objects in the output set

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

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

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

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

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.

Interpreting 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.

Code	Meaning	Description
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.

Querying the Job object associated with an API request

The Job object returned in the HTTP response contains several properties. You can query the state property 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.

Accessing 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.

Understanding and executing 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.

Understanding 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.

Constructing 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: Creating 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:admin123 -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:admin123 -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:admin123 -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:admin123 -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:admin123 -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:admin123 -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:admin123 -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:admin123 -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:admin123 -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:admin123 -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

Understanding 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.

Creating 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. 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

Adding 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

Deleting 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.

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

import traceback
import argparse
import json
import logging

from deploy_requests import DeployRequests


def add_vcenter_credentials(deploy, config):
    """ Add credentials for the vcenter if present in the config """
    log_debug_trace()

    vcenter = config.get('vcenter', None)
    if vcenter and not deploy.resource_exists('/security/credentials',
                                              'hostname', vcenter['hostname']):
        log_info("Registering vcenter {} credentials".format(vcenter['hostname']))
        data = {k: vcenter[k] for k in ['hostname', 'username', 'password']}
        data['type'] = "vcenter"
        deploy.post('/security/credentials', data)

```

```
40
41 def add_standalone_host_credentials(deploy, config):
42     """ Add credentials for standalone hosts if present in the config.
43         Does nothing if the host credential already exists on the Deploy.
44     """
45     log_debug_trace()
46
47     hosts = config.get('hosts', [])
48     for host in hosts:
49         # The presense of the 'password' will be used only for standalone hosts.
50         # If this host is managed by a vcenter, it should not have a host 'password'
   in the json.
51         if 'password' in host and not deploy.resource_exists('/security/credentials',
52                                                              'hostname', host[
   'name']):
53             log_info("Registering host {} credentials".format(host['name']))
54             data = {'hostname': host['name'], 'type': 'host',
55                     'username': host['username'], 'password': host['password']}
56             deploy.post('/security/credentials', data)
57
58
59 def register_unkown_hosts(deploy, config):
60     ''' Registers all hosts with the deploy server.
61         The host details are read from the cluster config json file.
62
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 config.
65     '''
66     log_debug_trace()
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 the 'hosts'
   list got {}".format(config))
71
72     missing_host_cnt = 0
73     for host in config['hosts']:
74         if not deploy.resource_exists('/hosts', 'name', host['name']):
75             missing_host_cnt += 1
76             host_config = {"name": host['name'], "hypervisor_type": host['type']}
77             if 'mgmt_server' in host:
78                 host_config["management_server"] = host['mgmt_server']
79                 log_info(
80                    "Registering from vcenter {mgmt_server}".format(**host))
81
82             if 'password' in host and 'user' in host:
83                 host_config['credential'] = {
84                     "password": host['password'], "username": host['user']}
```

```
85
86              log_info("Registering {type} host {name}".format(**host))
87              data["hosts"].append(host_config)
88
89      # only post /hosts if some missing hosts were found
90      if missing_host_cnt:
91          deploy.post('/hosts', data, wait_for_job=True)
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', cluster_config['name'])
102
103     if not cluster_id:
104         log_info("Creating cluster config named {name}".format(**cluster_config))
105
106         # Filter to only the valid attributes, ignores anything else in the json
107         data = {k: cluster_config[k] for k in [
108             'name', 'ip', 'gateway', 'netmask', 'ontap_image_version', 'dns_info',
    'ntp_servers']}
109
110         num_nodes = len(config['nodes'])
111
112         log_info("Cluster properties: {}".format(data))
113
114         resp = deploy.post('/v3/clusters?node_count={}'.format(num_nodes), data)
115         cluster_id = resp.headers.get('Location').split('/')[-1]
116
117     return cluster_id
118
119
120 def get_node_ids(deploy, cluster_id):
121     ''' Get the the ids of the nodes in a cluster. Returns a list of node_ids.'''
122     log_debug_trace()
123
124     response = deploy.get('/clusters/{}/nodes'.format(cluster_id))
125     node_ids = [node['id'] for node in response.json().get('records')]
126     return node_ids
127
128
129 def add_node_attributes(deploy, cluster_id, node_id, node):
130     ''' Set all the needed properties on a node '''
131     log_debug_trace()
```

```
132
133     log_info("Adding node '{}' properties".format(node_id))
134
135     data = {k: node[k] for k in ['ip', 'serial_number', 'instance_type',
136                                  'is_storage_efficiency_enabled'] if k in node}
137     # Optional: Set a serial_number
138     if 'license' in node:
139         data['license'] = {'id': node['license']}
140
141     # Assign the host
142     host_id = deploy.find_resource('/hosts', 'name', node['host_name'])
143     if not host_id:
144         log_and_exit("Host names must match in the 'hosts' array, and the
    nodes.host_name property")
145
146     data['host'] = {'id': host_id}
147
148     # Set the correct raid_type
149     is_hw_raid = not node['storage'].get('disks')  # The presence of a list of disks
    indicates sw_raid
150     data['passthrough_disks'] = not is_hw_raid
151
152     # Optionally set a custom node name
153     if 'name' in node:
154         data['name'] = node['name']
155
156     log_info("Node properties: {}".format(data))
157     deploy.patch('/clusters/{}/nodes/{}'.format(cluster_id, node_id), data)
158
159
160 def add_node_networks(deploy, cluster_id, node_id, node):
161     ''' Set the network information for a node '''
162     log_debug_trace()
163
164     log_info("Adding node '{}' network properties".format(node_id))
165
166     num_nodes = deploy.get_num_records('/clusters/{}/nodes'.format(cluster_id))
167
168     for network in node['networks']:
169
170         # single node clusters do not use the 'internal' network
171         if num_nodes == 1 and network['purpose'] == 'internal':
172             continue
173
174         # Deduce the network id given the purpose for each entry
175         network_id = deploy.find_resource('/clusters/{}/nodes/{}/networks'.format
    (cluster_id, node_id),
176                                           'purpose', network['purpose'])
```

```
177         data = {"name": network['name']}
178         if 'vlan' in network and network['vlan']:
179             data['vlan_id'] = network['vlan']
180 
181         deploy.patch('/clusters/{}/nodes/{}/networks/{}'.format(cluster_id, node_id,
    network_id), data)
182 
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 json properties
192     deploy.post(
193         '/clusters/{}/nodes/{}/storage/pools'.format(cluster_id, node_id), data)
194 
195     if 'disks' in node['storage'] and node['storage']['disks']:
196         data = {'disks': node['storage']['disks']}
197         deploy.post(
198             '/clusters/{}/nodes/{}/storage/disks'.format(cluster_id, node_id), data)
199 
200 
201 def create_cluster_config(deploy, config):
202     ''' Construct a cluster config in the deploy server using the input json data '''
203     log_debug_trace()
204 
205     cluster_id = add_cluster_attributes(deploy, config)
206 
207     node_ids = get_node_ids(deploy, cluster_id)
208     node_configs = config['nodes']
209 
210     for node_id, node_config in zip(node_ids, node_configs):
211         add_node_attributes(deploy, cluster_id, node_id, node_config)
212         add_node_networks(deploy, cluster_id, node_id, node_config)
213         add_node_storage(deploy, cluster_id, node_id, node_config)
214 
215     return cluster_id
216 
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'][
```

```
    'ontap_admin_password']}}
    deploy.post('/clusters/{}/deploy?inhibit_rollback=true'.format(cluster_id),
                data, wait_for_job=True)


def log_debug_trace():
    stack = traceback.extract_stack()
    parent_function = stack[-2][2]
    logging.getLogger('deploy').debug('Calling %s()' % parent_function)


def log_info(msg):
    logging.getLogger('deploy').info(msg)


def log_and_exit(msg):
    logging.getLogger('deploy').error(msg)
    exit(1)


def configure_logging(verbose):
    FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
    if verbose:
        logging.basicConfig(level=logging.DEBUG, format=FORMAT)
    else:
        logging.basicConfig(level=logging.INFO, format=FORMAT)
        logging.getLogger('requests.packages.urllib3.connectionpool').setLevel(
            logging.WARNING)


def main(args):
    configure_logging(args.verbose)
    deploy = DeployRequests(args.deploy, args.password)

    with open(args.config_file) as json_data:
        config = json.load(json_data)

        add_vcenter_credentials(deploy, config)

        add_standalone_host_credentials(deploy, config)

        register_unkown_hosts(deploy, config)

        cluster_id = create_cluster_config(deploy, config)

        deploy_cluster(deploy, cluster_id, config)


```

```
def parseArgs():
    parser = argparse.ArgumentParser(description='Uses the ONTAP Select Deploy API to construct and deploy a cluster.')
    parser.add_argument('-d', '--deploy', help='Hostname or IP address of Deploy server')
    parser.add_argument('-p', '--password', help='Admin password of Deploy server')
    parser.add_argument('-c', '--config_file', help='Filename of the cluster config')
    parser.add_argument('-v', '--verbose', help='Display extra debugging messages for seeing exact API calls and responses',
                        action='store_true', default=False)
    return parser.parse_args()

if __name__ == '__main__':
    args = parseArgs()
    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

```
{
  "hosts": [
    {
      "password": "mypassword1",
      "name": "host-1234",
      "type": "ESX",
      "username": "admin"
    }
  ],

  "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"
  },

  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip": "10.206.80.114",
      "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
          }
        ]
      }
    }
  ]
}
```

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.

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

import argparse
import logging
import json

from deploy_requests import DeployRequests


def post_new_license(deploy, license_filename):
    log_info('Posting a new license: {}'.format(license_filename))

    # Stream the file as multipart/form-data
    deploy.post('/licensing/licenses', data={},
                files={'license_file': open(license_filename, 'rb')})

    # Alternative if the NLF license data is converted to a string.
    # with open(license_filename, 'rb') as f:
    #    nlf_data = f.read()
    #    r = deploy.post('/licensing/licenses', data={},
    #                    files={'license_file': (license_filename, nlf_data)})


def put_license(deploy, serial_number, data, files):
    log_info('Adding license for serial number: {}'.format(serial_number))

    deploy.put('/licensing/licenses/{}'.format(serial_number), data=data, files=
files)


```

```
47 def put_used_license(deploy, serial_number, license_filename, ontap_username,
   ontap_password):
48     ''' If the license is used by an 'online' cluster, a username/password must be
   given. '''
49
50     data = {'ontap_username': ontap_username, 'ontap_password': ontap_password}
51     files = {'license_file': open(license_filename, 'rb')}
52
53     put_license(deploy, serial_number, data, files)
54
55
56 def put_free_license(deploy, serial_number, license_filename):
57     data = {}
58     files = {'license_file': open(license_filename, 'rb')}
59
60     put_license(deploy, serial_number, data, files)
61
62
63 def get_serial_number_from_license(license_filename):
64     ''' Read the NLF file to extract the serial number '''
65     with open(license_filename) as f:
66         data = json.load(f)
67
68         statusResp = data.get('statusResp', {})
69         serialNumber = statusResp.get('serialNumber')
70         if not serialNumber:
71             log_and_exit("The license file seems to be missing the serialNumber")
72
73         return 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').setLevel(logging
   .WARNING)
89
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 serial-number
98      if deploy.find_resource('/licensing/licenses', 'id', serial_number):
99
100         # If the license already exists in the Deploy server, determine if its used
101         if deploy.find_resource('/clusters', 'nodes.serial_number', serial_number):
102
103             # In this case, requires ONTAP creds to push the license to the node
104             if args.ontap_username and args.ontap_password:
105                 put_used_license(deploy, serial_number, args.license,
106                                  args.ontap_username, args.ontap_password)
107             else:
108                 print "ERROR: The serial number for this license is in use. Please
    provide ONTAP credentials."
109         else:
110             # License exists, but its not used
111             put_free_license(deploy, serial_number, args.license)
112     else:
113         # No license exists, so register a new one as an available license for later
    use
114         post_new_license(deploy, args.license)
115
116
117 def parseArgs():
118     parser = argparse.ArgumentParser(description='Uses the ONTAP Select Deploy API to
    add or update a new or used NLF license file.')
119     parser.add_argument('-d', '--deploy', required=True, type=str, help='Hostname or
    IP address of ONTAP Select Deploy')
120     parser.add_argument('-p', '--password', required=True, type=str, help='Admin
    password of Deploy server')
121     parser.add_argument('-l', '--license', required=True, type=str, help='Filename of
    the NLF license data')
122     parser.add_argument('-u', '--ontap_username', type=str,
123                         help='ONTAP Select username with privelege to add the
    license. Only provide if the license is used by a Node.')
124     parser.add_argument('-o', '--ontap_password', type=str,
125                         help='ONTAP Select password for the ontap_username. Required
    only if ontap_username is given.')
126     return parser.parse_args()
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.

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

import argparse
import json
import logging

from deploy_requests import DeployRequests

def find_cluster(deploy, cluster_name):
    return deploy.find_resource('/clusters', 'name', cluster_name)


def offline_cluster(deploy, cluster_id):
    # Test that the cluster is online, otherwise do nothing
    response = deploy.get('/clusters/{}?fields=state'.format(cluster_id))
    cluster_data = response.json()['record']
    if cluster_data['state'] == 'powered_on':
        log_info("Found the cluster to be online, modifying it to be powered_off.")
        deploy.patch('/clusters/{}'.format(cluster_id), {'availability':
'powered_off'}, True)


def delete_cluster(deploy, cluster_id):
    log_info("Deleting the cluster({}).".format(cluster_id))
    deploy.delete('/clusters/{}'.format(cluster_id), True)
    pass
```

```
43
44
45 def log_info(msg):
46     logging.getLogger('deploy').info(msg)
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').setLevel(logging
   .WARNING)
53
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['cluster']['name'],
   cluster_id))
65
66         offline_cluster(deploy, cluster_id)
67
68         delete_cluster(deploy, cluster_id)
69
70
71 def parseArgs():
72     parser = argparse.ArgumentParser(description='Uses the ONTAP Select Deploy API to
   delete a cluster')
73     parser.add_argument('-d', '--deploy', required=True, type=str, help='Hostname or
   IP address of Deploy server')
74     parser.add_argument('-p', '--password', required=True, type=str, help='Admin
   password of Deploy server')
75     parser.add_argument('-c', '--config_file', required=True, type=str, help='Filename
   of the cluster json config')
76     return parser.parse_args()
77
78 if __name__ == '__main__':
79     args = parseArgs()
80     main(args)
```

Common support module

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

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

import json
import logging
import requests

requests.packages.urllib3.disable_warnings()

class DeployRequests(object):
    '''
    Wrapper class for requests that simplifies the ONTAP Select Deploy
    path creation and header manipulations for simpler code.
    '''

    def __init__(self, ip, admin_password):
        self.base_url = 'https://{}/api'.format(ip)
        self.auth = ('admin', admin_password)
        self.headers = {'Accept': 'application/json'}
        self.logger = logging.getLogger('deploy')

    def post(self, path, data, files=None, wait_for_job=False):
        if files:
            self.logger.debug('POST FILES:')
            response = requests.post(self.base_url + path,
                                     auth=self.auth, verify=False,
                                     files=files)
```

```
        else:
            self.logger.debug('POST DATA: %s', data)
            response = requests.post(self.base_url + path,
                                     auth=self.auth, verify=False,
                                     json=data,
                                     headers=self.headers)

        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers(response),
response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def patch(self, path, data, wait_for_job=False):
        self.logger.debug('PATCH DATA: %s', data)
        response = requests.patch(self.base_url + path,
                                  auth=self.auth, verify=False,
                                  json=data,
                                  headers=self.headers)
        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers(response),
response.text)
        self.exit_on_errors(response)

        if wait_for_job and response.status_code == 202:
            self.wait_for_job(response.json())
        return response

    def put(self, path, data, files=None, wait_for_job=False):
        if files:
            print('PUT FILES: {}'.format(data))
            response = requests.put(self.base_url + path,
                                    auth=self.auth, verify=False,
                                    data=data,
                                    files=files)
        else:
            self.logger.debug('PUT DATA:')
            response = requests.put(self.base_url + path,
                                    auth=self.auth, verify=False,
                                    json=data,
                                    headers=self.headers)

        self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers(response),
response.text)
        self.exit_on_errors(response)

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

```
        self.logger.info('Event: ' + response['job']['message'])

        while True:
            response = self.get('/jobs/{}?fields=state,message&'
                                'poll_timeout={}&last_modified=>={}'.format(
                                    job_id, poll_timeout, last_modified))

            job_body = response.json().get('record', {})

            # Show interesting message updates
            message = job_body.get('message', '')
            self.logger.info('Event: ' + message)

            # Refresh the last modified time for the poll loop
            last_modified = job_body.get('last_modified')

            # Look for the final states
            state = job_body.get('state', 'unknown')
            if state in ['success', 'failure']:
                if state == 'failure':
                    self.logger.error('FAILED background job.\nJOB: %s', job_body)
                    exit(1)   # End the script if a failure occurs
                break

    def exit_on_errors(self, response):
        if response.status_code >= 400:
            self.logger.error('FAILED request to URL: %s\nHEADERS: %s\nRESPONSE BODY:
%s',
                              response.request.url,
                              self.filter_headers(response),
                              response.text)
        response.raise_for_status()   # Displays the response error, and exits the
script

    @staticmethod
    def filter_headers(response):
        ''' Returns a filtered set of the response headers '''
        return {key: response.headers[key] for key in ['Location', '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.

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

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

import argparse
import logging
import sys

from deploy_requests import DeployRequests


def _parse_args():
    """ Parses the arguments provided on the command line when executing this
        script and returns the resulting namespace. If all required arguments
        are not provided, an error message indicating the mismatch is printed and
        the script will exit.
    """

    parser = argparse.ArgumentParser(description=(
        'Uses the ONTAP Select Deploy API to resize the nodes in the cluster.'
        ' For example, you might have a small (4 CPU, 16GB RAM per node) 2 node'
        ' cluster and wish to resize the cluster to medium (8 CPU, 64GB RAM per'
        ' node). This script will take in the cluster details and then perform'
        ' the operation and wait for it to complete.'
    ))
    parser.add_argument('--deploy', required=True, help=(
        'Hostname or IP of the ONTAP Select Deploy VM.'
    ))
    parser.add_argument('--deploy-password', required=True, help=(
        'The password for the ONTAP Select Deploy admin user.'
    ))
    parser.add_argument('--cluster', required=True, help=(
        'Hostname or IP of the cluster management interface.'
    ))
    parser.add_argument('--instance-type', required=True, help=(
```

```
        'The desired instance size of the nodes after the operation is complete.'
    ))
    parser.add_argument('--ontap-password', required=True, help=(
        'The password for the ONTAP administrative user account.'
    ))
    parser.add_argument('--ontap-username', default='admin', help=(
        'The username for the ONTAP administrative user account. Default: admin.'
    ))
    parser.add_argument('--nodes', nargs='+', metavar='NODE_NAME', help=(
        'A space separated list of node names for which the resize operation'
        ' should be performed. The default is to apply the resize to all nodes in'
        ' the cluster. If a list of nodes is provided, it must be provided in HA'
        ' pairs. That is, in a 4 node cluster, nodes 1 and 2 (partners) must be'
        ' resized in the same operation.'
    ))
    return parser.parse_args()


def _get_cluster(deploy, parsed_args):
    """ Locate the cluster using the arguments provided """

    cluster_id = deploy.find_resource('/clusters', 'ip', parsed_args.cluster)
    if not cluster_id:
        return None
    return deploy.get('/clusters/%s?fields=nodes' % cluster_id).json()['record']


def _get_request_body(parsed_args, cluster):
    """ Build the request body """

    changes = {'admin_password': parsed_args.ontap_password}

    # if provided, use the list of nodes given, else use all the nodes in the cluster
    nodes = [node for node in cluster['nodes']]
    if parsed_args.nodes:
        nodes = [node for node in nodes if node['name'] in parsed_args.nodes]

    changes['nodes'] = [
        {'instance_type': parsed_args.instance_type, 'id': node['id']} for node in
nodes]

    return changes


def main():
    """ Set up the resize operation by gathering the necessary data and then send
        the request to the ONTAP Select Deploy server.
    """
```

```
 98
 99     logging.basicConfig(
100         format='[%(asctime)s] [%(levelname)5s] %(message)s', level=logging.INFO,)
101
102     logging.getLogger('requests.packages.urllib3').setLevel(logging.WARNING)
103
104     parsed_args = _parse_args()
105     deploy = DeployRequests(parsed_args.deploy, parsed_args.deploy_password)
106
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' % parsed_args
    .cluster)
111         return 1
112
113     changes = _get_request_body(parsed_args, cluster)
114     deploy.patch('/clusters/%s' % cluster['id'], changes, wait_for_job=True)
115
116 if __name__ == '__main__':
117     sys.exit(main())
```