Common support module

ONTAP Select

David Peterson
November 21, 2019

NetApp®

Table of Contents

Common support module

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

```python
#!/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,
```

```python
                                      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)
```

```python
 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, 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']
```

```python
            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}
```