



Common support module

ONTAP Select

David Peterson
November 21, 2019

This PDF was generated from https://docs.netapp.com/us-en/ontap-select/reference_api_script_common.html on December 09, 2020. Always check docs.netapp.com for the latest.

Table of Contents

Common support module 1

Common support module

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

```
1 #!/usr/bin/env python
2 ##-----
3 #
4 # File: deploy_requests.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
20 import json
21 import logging
22 import requests
23
24 requests.packages.urllib3.disable_warnings()
25
26 class DeployRequests(object):
27     '''
28     Wrapper class for requests that simplifies the ONTAP Select Deploy
29     path creation and header manipulations for simpler code.
30     '''
31
32     def __init__(self, ip, admin_password):
33         self.base_url = 'https://{}/api'.format(ip)
34         self.auth = ('admin', admin_password)
35         self.headers = {'Accept': 'application/json'}
36         self.logger = logging.getLogger('deploy')
37
38     def post(self, path, data, files=None, wait_for_job=False):
39         if files:
40             self.logger.debug('POST FILES:')
41             response = requests.post(self.base_url + path,
```

```

43             files=files)
44     else:
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.filter_headers(response),
52 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
66         self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers(response),
67 response.text)
68         self.exit_on_errors(response)
69
70         if wait_for_job and response.status_code == 202:
71             self.wait_for_job(response.json())
72         return response
73
74     def put(self, path, data, files=None, wait_for_job=False):
75         if files:
76             print('PUT FILES: {}'.format(data))
77             response = requests.put(self.base_url + path,
78                                   auth=self.auth, verify=False,
79                                   data=data,
80                                   files=files)
81
82         else:
83             self.logger.debug('PUT DATA:')
84             response = requests.put(self.base_url + path,
85                                   auth=self.auth, verify=False,
86                                   json=data,
87                                   headers=self.headers)
88
89             self.logger.debug('HEADERS: %s\nBODY: %s', self.filter_headers(response),
90 response.text)
91             self.exit_on_errors(response)
92

```

```

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']

```

```

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, last_modified))
140
141         job_body = response.json().get('record', {})
142
143         # Show interesting message updates
144         message = job_body.get('message', '')
145         self.logger.info('Event: ' + message)
146
147         # Refresh the last modified time for the poll loop
148         last_modified = job_body.get('last_modified')
149
150         # Look for the final states
151         state = job_body.get('state', 'unknown')
152         if state in ['success', 'failure']:
153             if state == 'failure':
154                 self.logger.error('FAILED background job.\nJOB: %s', job_body)
155                 exit(1) # End the script if a failure occurs
156             break
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:
161 %s',
162                             response.request.url,
163                             self.filter_headers(response),
164                             response.text)
165             response.raise_for_status() # Displays the response error, and exits the
script
166
167     @staticmethod
168     def filter_headers(response):
169         ''' Returns a filtered set of the response headers '''
170         return {key: response.headers[key] for key in ['Location', 'request-id'] if
key in response.headers}

```

Copyright Information

Copyright © 2020 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system-without prior written permission of the copyright owner.

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

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

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

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

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

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.