



Python code samples

ONTAP Select

NetApp
August 14, 2025

This PDF was generated from https://docs.netapp.com/us-en/ontap-select/reference_api_script_cc.html on August 14, 2025. Always check docs.netapp.com for the latest.

Table of Contents

Python code samples	1
Script to create an ONTAP Select cluster	1
JSON for script to create an ONTAP Select cluster	8
Single-node cluster on ESXi	8
Single-node cluster on ESXi using vCenter	10
Single-node cluster on KVM	11
Script to add an ONTAP Select node license	13
Script to delete an ONTAP Select cluster	16
Common support Python module for ONTAP Select	18
Script to resize ONTAP Select cluster nodes	23

Python code samples

Script to create an ONTAP Select 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
# 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 traceback
21 import argparse
22 import json
23 import logging
24
25 from deploy_requests import DeployRequests
26
27
28 def add_vcenter_credentials(deploy, config):
29     """ Add credentials for the vcenter if present in the config """
30     log_debug_trace()
31
32     vcenter = config.get('vcenter', None)
33     if vcenter and not deploy.resource_exists('/security/credentials',
# 'hostname', vcenter
34                                         ['hostname']):
```

```

35     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
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

35 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 Does nothing if the host credential already exists on the
44 Deploy.
45 """
46 log_debug_trace()
47
48 hosts = config.get('hosts', [])
49 for host in hosts:
50 # The presence of the 'password' will be used only for
51 # standalone hosts.
52 # If this host is managed by a vcenter, it should not have a
53 # host 'password' in the json.
54 if 'password' in host and not deploy.resource_exists
55 ('/security/credentials',
56 'hostname', host['name']):
57 log_info("Registering host {} credentials".format(host
58 ['name']))
59 data = {'hostname': host['name'], 'type': 'host',
60 'username': host['username'], 'password': host
61 ['password']}
62 deploy.post('/security/credentials', data)
63
64
65
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")

```

the 'hosts' list got {}".format(config))

71     missing_host_cnt = 0
72     for host in config['hosts']:
73         if not deploy.resource_exists('/hosts', 'name', host['name']):
74             missing_host_cnt += 1
75             host_config = {"name": host['name'], "hypervisor_type":
host['type']}
76             if 'mgmt_server' in host:
77                 host_config["management_server"] = host['mgmt_server']
78                 log_info(
79                     "Registering from vcenter {mgmt_server}".format(
80                         **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(
105            **cluster_config))
106
107        # Filter to only the valid attributes, ignores anything else
in the json
108        data = {k: cluster_config[k] for k in [
109            '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(
115         num_nodes), data)
116     cluster_id = resp.headers.get('Location').split('/')[-1]
117
118     return cluster_id
119
120 def get_node_ids(deploy, cluster_id):
121     ''' Get the the ids of the nodes in a cluster. Returns a list of
122     node_ids.'''
123     log_debug_trace()
124
125     response = deploy.get('/clusters/{}/nodes'.format(cluster_id))
126     node_ids = [node['id'] for node in response.json().get('records')]
127
128     return node_ids
129
130 def add_node_attributes(deploy, cluster_id, node_id, node):
131     ''' Set all the needed properties on a node '''
132     log_debug_trace()
133
134     log_info("Adding node '{}' properties".format(node_id))
135
136     data = {k: node[k] for k in ['ip', 'serial_number',
137         'instance_type',
138             'is_storage_efficiency_enabled']] if k
139     in node}
140
141     # Optional: Set a serial_number
142     if 'license' in node:
143         data['license'] = {'id': node['license']}
144
145     # Assign the host
146     host_id = deploy.find_resource('/hosts', 'name', node[
147         'host_name'])
148     if not host_id:
149         log_and_exit("Host names must match in the 'hosts' array, and
150         the nodes.host_name property")
151
152     data['host'] = {'id': host_id}
153
154     # 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),
158                  data)
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(
167                                         cluster_id))
168
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(
176             '/clusters/{}/nodes/{}/networks'.format(cluster_id, node_id),
177                                         'purpose', network[
178                                         'purpose'])
179
180         data = {"name": network['name']}
181         if 'vlan' in network and network['vlan']:
182             data['vlan_id'] = network['vlan']
183
184         deploy.patch('/clusters/{}/nodes/{}/networks/{}'.format(
185             cluster_id, node_id, network_id), data)
186
187
188 def add_node_storage(deploy, cluster_id, node_id, node):
189     ''' Set all the storage information on a node '''
190     log_debug_trace()
191
192     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
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']
224         }['ontap_admin_password']]}
225     deploy.post('/clusters/{}/deploy?inhibit_rollback=true'.format
226         (cluster_id),
227             data, wait_for_job=True)
228
229 def log_debug_trace():

```

```

229     stack = traceback.extract_stack()
230     parent_function = stack[-2][2]
231     logging.getLogger('deploy').debug('Calling %s()' % 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')
250             .setLevel(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

```

```

    Select Deploy API to construct and deploy a cluster.')
273     parser.add_argument('-d', '--deploy', help='Hostname or IP address
of Deploy server')
274     parser.add_argument('-p', '--password', help='Admin password of
Deploy server')
275     parser.add_argument('-c', '--config_file', help='Filename of the
cluster config')
276     parser.add_argument('-v', '--verbose', help='Display extra
debugging messages for seeing exact API calls and responses',
action='store_true', default=False)
277
278     return parser.parse_args()
279
280 if __name__ == '__main__':
281     args = parseArgs()
282     main(args)

```

JSON for script to create an ONTAP Select 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": "Datastore",  
          "purpose": "storage",  
          "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 an ONTAP Select 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
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
24import argparse
25import logging
26import json
27
28from deploy_requests import DeployRequests
29
30
31def post_new_license(deploy, license_filename):
32    log_info('Posting a new license: {}'.format(license_filename))
33
34    # Stream the file as multipart/form-data
35    deploy.post('/licensing/licenses', data={}, files={'license_file': open(license_filename, 'rb')})
36
37    # 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 def put_license(deploy, serial_number, data, files):
42     log_info('Adding license for serial number: {}'.format(
43         serial_number))
44
45     deploy.put('/licensing/licenses/{}'.format(serial_number), data
46     =data, files=files)
47
48 def put_used_license(deploy, serial_number, license_filename,
49     ontap_username, ontap_password):
50     ''' If the license is used by an 'online' cluster, a
51     username/password must be given. '''
52
53     data = {'ontap_username': ontap_username, 'ontap_password':
54         ontap_password}
55     files = {'license_file': open(license_filename, 'rb')}
56
57     put_license(deploy, serial_number, data, files)
58
59
60 def put_free_license(deploy, serial_number, license_filename):
61     data = {}
62     files = {'license_file': open(license_filename, 'rb')}
63
64     put_license(deploy, serial_number, data, files)
65
66
67 def get_serial_number_from_license(license_filename):
68     ''' Read the NLF file to extract the serial number '''
69     with open(license_filename) as f:
70         data = json.load(f)
71
72         statusResp = data.get('statusResp', {})
73         serialNumber = statusResp.get('serialNumber')
74
75         if not serialNumber:
76             log_and_exit("The license file seems to be missing the
77             serialNumber")
78
79         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')
89         .setLevel(logging.WARNING)
90
91
92 def main(args):
93     configure_logging()
94     serial_number = get_serial_number_from_license(args.license)
95
96     deploy = DeployRequests(args.deploy, args.password)
97
98     # First check if there is already a license resource for this
99     # serial-number
100    if deploy.find_resource('/licensing/licenses', 'id',
101        serial_number):
102
103        # If the license already exists in the Deploy server,
104        # determine if its used
105        if deploy.find_resource('/clusters', 'nodes.serial_number',
106            serial_number):
107
108            # In this case, requires ONTAP creds to push the license
109            # to the node
110            if args.ontap_username and args.ontap_password:
111                put_used_license(deploy, serial_number, args.license,
112                                args.ontap_username, args
113                                .ontap_password)
114            else:
115                print("ERROR: The serial number for this license is in
116                use. Please provide ONTAP credentials.")
117            else:
118                # License exists, but its not used
119                put_free_license(deploy, serial_number, args.license)

```

```

112     else:
113         # No license exists, so register a new one as an available
114         license for later use
115         post_new_license(deploy, args.license)
116
117 def parseArgs():
118     parser = argparse.ArgumentParser(description='Uses the ONTAP
119     Select Deploy API to add or update a new or used NLF license file.')
120     parser.add_argument('-d', '--deploy', required=True, type=str,
121         help='Hostname or IP address of ONTAP Select Deploy')
122     parser.add_argument('-p', '--password', required=True, type=str,
123         help='Admin password of Deploy server')
124     parser.add_argument('-l', '--license', required=True, type=str,
125         help='Filename of the NLF license data')
126     parser.add_argument('-u', '--ontap_username', type=str,
127         help='ONTAP Select username with privilege to
128         add the license. Only provide if the license is used by a Node.')
129     parser.add_argument('-o', '--ontap_password', type=str,
130         help='ONTAP Select password for the
131         ontap_username. Required only if ontap_username is given.')
132
133     return parser.parse_args()
134
135
136 if __name__ == '__main__':
137     args = parseArgs()
138     main(args)

```

Script to delete an ONTAP Select 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,

```

```

14 # provided that the above copyright notice appears in all copies and
15 # that the software application product is distributed pursuant to
16 # terms
17 #
18 #-----
19
20 import argparse
21 import json
22 import logging
23
24 from deploy_requests import DeployRequests
25
26 def find_cluster(deploy, cluster_name):
27     return deploy.find_resource('/clusters', 'name', cluster_name)
28
29
30 def offline_cluster(deploy, cluster_id):
31     # Test that the cluster is online, otherwise do nothing
32     response = deploy.get('/clusters/{}/?fields=state'.format(
33         cluster_id))
34     cluster_data = response.json()['record']
35     if cluster_data['state'] == 'powered_on':
36         log_info("Found the cluster to be online, modifying it to be
37         powered_off.")
38         deploy.patch('/clusters/{}'.format(cluster_id), {
39             'availability': 'powered_off'}, True)
40
41
42 def delete_cluster(deploy, cluster_id):
43     log_info("Deleting the cluster({}).".format(cluster_id))
44     deploy.delete('/clusters/{}'.format(cluster_id), True)
45     pass
46
47
48
49 def log_info(msg):
50     logging.getLogger('deploy').info(msg)
51
52
53 def configure_logging():
54     FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
55     logging.basicConfig(level=logging.INFO, format=FORMAT)
56     logging.getLogger('requests.packages.urllib3.connectionpool')
57         .setLevel(logging.WARNING)

```

```

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     return parser.parse_args()
81
82 if __name__ == '__main__':
83     args = parseArgs()
84     main(args)

```

Common support Python module for ONTAP Select

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,
42                                     auth=self.auth, verify=False,
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), response.text)

```

```

52     self.exit_on_errors(response)
53
54     if wait_for_job and response.status_code == 202:
55         self.wait_for_job(response.json())
56
57     return response
58
58     def patch(self, path, data, wait_for_job=False):
59         self.logger.debug('PATCH DATA: %s', data)
60         response = requests.patch(self.base_url + path,
61                                   auth=self.auth, verify=False,
62                                   json=data,
63                                   headers=self.headers)
64         self.logger.debug('HEADERS: %s\nBODY: %s', self.
filter_headers(response), response.text)
65         self.exit_on_errors(response)
66
67     if wait_for_job and response.status_code == 202:
68         self.wait_for_job(response.json())
69
70     return response
71
71     def put(self, path, data, files=None, wait_for_job=False):
72         if files:
73             print('PUT FILES: {}'.format(data))
74             response = requests.put(self.base_url + path,
75                                     auth=self.auth, verify=False,
76                                     data=data,
77                                     files=files)
78         else:
79             self.logger.debug('PUT DATA:')
80             response = requests.put(self.base_url + path,
81                                     auth=self.auth, verify=False,
82                                     json=data,
83                                     headers=self.headers)
84
85             self.logger.debug('HEADERS: %s\nBODY: %s', self.
filter_headers(response), response.text)
86             self.exit_on_errors(response)
87
88         if wait_for_job and response.status_code == 202:
89             self.wait_for_job(response.json())
90
91         return response
92
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.
96     filter_headers(response), response.text)
97         self.exit_on_errors(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.
102             .auth, verify=False)
102         self.logger.debug('HEADERS: %s\nBODY: %s', self.
103     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
108     return response
109
110     def find_resource(self, path, name, value):
111         ''' Returns the 'id' of the resource if it exists, otherwise
112             None '''
113         resource = None
114         response = self.get('{path}?{field}={value}'.format(
115             path=path, field=name, value=value))
116         if response.status_code == 200 and response.json().get
117             ('num_records') >= 1:
118             resource = response.json().get('records')[0].get('id')
119
120     return resource
121
122     def get_num_records(self, path, query=None):
123         ''' Returns the number of records found in a container, or
124             None on error '''
125         resource = None
126         query_opt = '?{}'.format(query) if query else ''
127         response = self.get('{path}{query}'.format(path=path, query
128             =query_opt))
129         if response.status_code == 200 :
130             return response.json().get('num_records')
131
132     return None
133
134     def resource_exists(self, path, name, value):
135         return self.find_resource(path, name, value) is not None
136
137     def wait_for_job(self, response, poll_timeout=120):
138         last_modified = response['job']['last_modified']
139         job_id = response['job']['id']
140
141

```

```

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',
156                                   job_body)
157                 exit(1)      # End the script if a failure occurs
158             break
159
160     def exit_on_errors(self, response):
161         if response.status_code >= 400:
162             self.logger.error('FAILED request to URL: %s\nHEADERS: %s\nRESPONSE BODY: %s',
163                               response.request.url,
164                               self.filter_headers(response),
165                               response.text)
166             response.raise_for_status()    # Displays the response error,
167                                         and exits the script
168
169     @staticmethod
170     def filter_headers(response):
171         ''' Returns a filtered set of the response headers '''
172         return {key: response.headers[key] for key in ['Location',
173 'request-id'] if key in response.headers}

```

Script to resize ONTAP Select 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
# 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 argparse
21 import logging
22 import sys
23
24 from deploy_requests import DeployRequests
25
26
27 def _parse_args():
28     """ Parses the arguments provided on the command line when
executing this
script and returns the resulting namespace. If all required
arguments
30     are not provided, an error message indicating the mismatch is
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
cluster.'
```

```

36         ' For example, you might have a small (4 CPU, 16GB RAM per
node) 2 node'
37         ' cluster and wish to resize the cluster to medium (8 CPU,
64GB RAM per'
38         ' node). This script will take in the cluster details and then
perform'
39         ' the operation and wait for it to complete.'
40     ))
41     parser.add_argument('--deploy', required=True, help=(
42         'Hostname or IP of the ONTAP Select Deploy VM.'
43     ))
44     parser.add_argument('--deploy-password', required=True, help=(
45         'The password for the ONTAP Select Deploy admin user.'
46     ))
47     parser.add_argument('--cluster', required=True, help=(
48         'Hostname or IP of the cluster management interface.'
49     ))
50     parser.add_argument('--instance-type', required=True, help=(
51         'The desired instance size of the nodes after the operation is
complete.'
52     ))
53     parser.add_argument('--ontap-password', required=True, help=(
54         'The password for the ONTAP administrative user account.'
55     ))
56     parser.add_argument('--ontap-username', default='admin', help=(
57         'The username for the ONTAP administrative user account.
Default: admin.'
58     ))
59     parser.add_argument('--nodes', nargs='+', metavar='NODE_NAME',
help=(
60         'A space separated list of node names for which the resize
operation'
61         ' should be performed. The default is to apply the resize to
all nodes in'
62         ' the cluster. If a list of nodes is provided, it must be
provided in HA'
63         ' pairs. That is, in a 4 node cluster, nodes 1 and 2
(partners) must be'
64         ' resized in the same operation.'
65     ))
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
    .cluster)
73     if not cluster_id:
74         return None
75     return deploy.get('/clusters/%s?fields=nodes' % cluster_id).
    json()['record']
76
77
78 def _get_request_body(parsed_args, cluster):
79     """ Build the request body """
80
81     changes = {'admin_password': parsed_args.ontap_password}
82
83     # if provided, use the list of nodes given, else use all the nodes
84     # in the cluster
85     nodes = [node for node in cluster['nodes']]
86     if parsed_args.nodes:
87         nodes = [node for node in nodes if node['name'] in
88         parsed_args.nodes]
89
90     changes['nodes'] = [
91         {'instance_type': parsed_args.instance_type, 'id': node['id']}
92         for node in nodes]
93
94 def main():
95     """ Set up the resize operation by gathering the necessary data
96     and then send
97         the request to the ONTAP Select Deploy server.
98
99     logging.basicConfig(
100         format='[%(asctime)s] [%(levelname)5s] %(message)s', level
101         =logging.INFO,)
102
103     logging.getLogger('requests.packages.urllib3').setLevel(logging
104         .WARNING)
105
106     parsed_args = _parse_args()
107     deploy = DeployRequests(parsed_args.deploy, parsed_args
108         .deploy_password)
109
110     cluster = _get_cluster(deploy, parsed_args)
111     if not cluster:

```

```
109         deploy.logger.error(
110             'Unable to find a cluster with a management IP of %s' %
111             parsed_args.cluster)
112
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 © 2025 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.