



# Python code samples

## ONTAP Select

NetApp  
October 23, 2024

# Table of Contents

- Python code samples . . . . . 1
  - Script to create a cluster . . . . . 1
  - JSON for script to create a cluster . . . . . 8
  - Script to add a node license . . . . . 13
  - Script to delete a cluster . . . . . 16
  - Common support module . . . . . 18
  - Script to resize cluster nodes . . . . . 23

# Python code samples

## Script to create a cluster

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

```
1 #!/usr/bin/env python
2 ##-----
3 #
4 # File: cluster.py
5 #
6 # (C) Copyright 2019 NetApp, Inc.
7 #
8 # This sample code is provided AS IS, with no support or warranties of
9 # any kind, including but not limited for warranties of
10 # merchantability
11 # or fitness of any kind, expressed or implied. Permission to use,
12 # reproduce, modify and create derivatives of the sample code is
13 # granted
14 # solely for the purpose of researching, designing, developing and
15 # testing a software application product for use with NetApp products,
16 # provided that the above copyright notice appears in all copies and
17 # that the software application product is distributed pursuant to
18 # terms
19 # no less restrictive than those set forth herein.
20 ##-----
21
22 import traceback
23 import argparse
24 import json
25 import logging
26
27 from deploy_requests import DeployRequests
28
29 def add_vcenter_credentials(deploy, config):
30     """ Add credentials for the vcenter if present in the config """
31     log_debug_trace()
32
33     vcenter = config.get('vcenter', None)
34     if vcenter and not deploy.resource_exists('/security/credentials',
35                                             'hostname', vcenter
36                                             ['hostname']):
```

```

35     log_info("Registering vcenter {} credentials".format(vcenter
    ['hostname']))
36     data = {k: vcenter[k] for k in ['hostname', 'username',
    'password']}
37     data['type'] = "vcenter"
38     deploy.post('/security/credentials', data)
39
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']}}
224     deploy.post('/clusters/{}/deploy?inhibit_rollback=true'.format
           (cluster_id),
225                 data, wait_for_job=True)
226
227
228 def log_debug_trace():

```



```

229     stack = traceback.extract_stack()
230     parent_function = stack[-2][2]
231     logging.getLogger('deploy').debug('Calling %s()' %
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'
).setLevel(
250         logging.WARNING)
251
252
253 def main(args):
254     configure_logging(args.verbose)
255     deploy = DeployRequests(args.deploy, args.password)
256
257     with open(args.config_file) as json_data:
258         config = json.load(json_data)
259
260         add_vcenter_credentials(deploy, config)
261
262         add_standalone_host_credentials(deploy, config)
263
264         register_unkown_hosts(deploy, config)
265
266         cluster_id = create_cluster_config(deploy, config)
267
268         deploy_cluster(deploy, cluster_id, config)
269
270
271 def parseArgs():
272     parser = argparse.ArgumentParser(description='Uses the ONTAP

```

```

    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',
277                          action='store_true', default=False)
278     return parser.parse_args()
279
280 if __name__ == '__main__':
281     args = parseArgs()
282     main(args)

```

## JSON for script to create a cluster

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

### Single-node cluster on ESXi

```

1 {
2   "hosts": [
3     {
4       "password": "mypassword1",
5       "name": "host-1234",
6       "type": "ESX",
7       "username": "admin"
8     }
9   ],
10
11  "cluster": {
12    "dns_info": {
13      "domains": ["lab1.company-demo.com", "lab2.company-demo.com",
14                 "lab3.company-demo.com", "lab4.company-demo.com"]
15    },
16
17    "dns_ips": ["10.206.80.135", "10.206.80.136"]
18  },
19  "ontap_image_version": "9.7",
20  "gateway": "10.206.80.1",
21  "ip": "10.206.80.115",

```

```
22     "name": "mycluster",
23     "ntp_servers": ["10.206.80.183", "10.206.80.142"],
24     "ontap_admin_password": "mypassword2",
25     "netmask": "255.255.254.0"
26 },
27
28 "nodes": [
29   {
30     "serial_number": "3200000nn",
31     "ip": "10.206.80.114",
32     "name": "node-1",
33     "networks": [
34       {
35         "name": "ontap-external",
36         "purpose": "mgmt",
37         "vlan": 1234
38       },
39       {
40         "name": "ontap-external",
41         "purpose": "data",
42         "vlan": null
43       },
44       {
45         "name": "ontap-internal",
46         "purpose": "internal",
47         "vlan": null
48       }
49     ],
50     "host_name": "host-1234",
51     "is_storage_efficiency_enabled": false,
52     "instance_type": "small",
53     "storage": {
54       "disk": [],
55       "pools": [
56         {
57           "name": "storage-pool-1",
58           "capacity": 4802666790125
59         }
60       ]
61     }
62   }
63 ]
64 }
```

## Single-node cluster on ESXi using vCenter

```
{
  "hosts": [
    {
      "name": "host-1234",
      "type": "ESX",
      "mgmt_server": "vcenter-1234"
    }
  ],
  "cluster": {
    "dns_info": { "domains": ["lab1.company-demo.com", "lab2.company-
demo.com",
      "lab3.company-demo.com", "lab4.company-demo.com"
    ],
    "dns_ips": ["10.206.80.135", "10.206.80.136"]
  },
  "ontap_image_version": "9.7",
  "gateway": "10.206.80.1",
  "ip": "10.206.80.115",
  "name": "mycluster",
  "ntp_servers": ["10.206.80.183", "10.206.80.142"],
  "ontap_admin_password": "mypassword2",
  "netmask": "255.255.254.0"
},
  "vcenter": {
    "password": "mypassword2",
    "hostname": "vcenter-1234",
    "username": "selectadmin"
  },
  "nodes": [
    {
      "serial_number": "3200000nn",
      "ip": "10.206.80.114",
      "name": "node-1",
      "networks": [
        {
          "name": "ONTAP-Management",
          "purpose": "mgmt",
          "vlan": null
        },
        {
```

```

        "name": "ONTAP-External",
        "purpose": "data",
        "vlan": null
    },
    {
        "name": "ONTAP-Internal",
        "purpose": "internal",
        "vlan": null
    }
],

"host_name": "host-1234",
"is_storage_efficiency_enabled": false,
"instance_type": "small",
"storage": {
    "disk": [],
    "pools": [
        {
            "name": "storage-pool-1",
            "capacity": 5685190380748
        }
    ]
}
}
]
}

```

## Single-node cluster on KVM



- Beginning with ONTAP Select 9.10.1, you can no longer deploy a new cluster on the KVM hypervisor.
- Beginning with ONTAP Select 9.11.1, all manageability functionality is no longer available for existing KVM clusters and hosts, except for the take offline and delete functions.

```

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

  "cluster": {

```

```

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

"ontap_image_version": "9.7",
"gateway": "10.206.80.1",
"ip": "10.206.80.115",
"name": "CBF4ED97",
"ntp_servers": ["10.206.80.183", "10.206.80.142"],
"ontap_admin_password": "mypassword2",
"netmask": "255.255.254.0"
},
"nodes": [
  {
    "serial_number": "3200000nn",
    "ip": "10.206.80.115",
    "name": "node-1",
    "networks": [
      {
        "name": "ontap-external",
        "purpose": "mgmt",
        "vlan": 1234
      },
      {
        "name": "ontap-external",
        "purpose": "data",
        "vlan": null
      },
      {
        "name": "ontap-internal",
        "purpose": "internal",
        "vlan": null
      }
    ]
  },
  {
    "host_name": "host-1234",
    "is_storage_efficiency_enabled": false,
    "instance_type": "small",
    "storage": {
      "disk": [],
      "pools": [
        {

```

```

        "name": "storage-pool-1",
        "capacity": 4802666790125
    }
]
}
]
}

```

## Script to add a node license

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

```

1 #!/usr/bin/env python
2 ##-----
3 #
4 # File: add_license.py
5 #
6 # (C) Copyright 2019 NetApp, Inc.
7 #
8 # This sample code is provided AS IS, with no support or warranties of
9 # any kind, including but not limited for warranties of
10 # merchantability
11 # or fitness of any kind, expressed or implied. Permission to use,
12 # reproduce, modify and create derivatives of the sample code is
13 # granted
14 # solely for the purpose of researching, designing, developing and
15 # testing a software application product for use with NetApp products,
16 # provided that the above copyright notice appears in all copies and
17 # that the software application product is distributed pursuant to
18 # terms
19 # no less restrictive than those set forth herein.
20 #
21 ##-----
22
23
24 import argparse
25 import logging
26 import json
27
28 from deploy_requests import DeployRequests
29
30 def post_new_license(deploy, license_filename):
31     log_info('Posting a new license: {}'.format(license_filename))
32
33

```

```

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

```

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
    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 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(
    cluster_id))
33     cluster_data = response.json()['record']
34     if cluster_data['state'] == 'powered_on':
35         log_info("Found the cluster to be online, modifying it to be
    powered_off.")
36         deploy.patch('/clusters/{}'.format(cluster_id), {
    'availability': 'powered_off'}, True)
37
38
39 def delete_cluster(deploy, cluster_id):
40     log_info("Deleting the cluster({}).".format(cluster_id))
41     deploy.delete('/clusters/{}'.format(cluster_id), True)
42     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'
53 )).setLevel(logging.WARNING)
54
55 def main(args):
56     configure_logging()
57     deploy = DeployRequests(args.deploy, args.password)
58
59     with open(args.config_file) as json_data:
60         config = json.load(json_data)
61
62         cluster_id = find_cluster(deploy, config['cluster']['name'])
63
64         log_info("Found the cluster {} with id: {}".format(config
65 ['cluster']['name'], cluster_id))
66
67         offline_cluster(deploy, cluster_id)
68
69         delete_cluster(deploy, cluster_id)
70
71 def parseArgs():
72     parser = argparse.ArgumentParser(description='Uses the ONTAP Select
73 Deploy API to delete a cluster')
74     parser.add_argument('-d', '--deploy', required=True, type=str,
75 help='Hostname or IP address of Deploy server')
76     parser.add_argument('-p', '--password', required=True, type=str,
77 help='Admin password of Deploy server')
78     parser.add_argument('-c', '--config_file', required=True, type=str,
79 help='Filename of the cluster json config')
80     return parser.parse_args()
81
82 if __name__ == '__main__':
83     args = parseArgs()
84     main(args)

```

## 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,
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     return response
57
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         return response
70
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         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=>={}'
139                                 .format(
140                                     job_id, poll_timeout,
141                                     last_modified))
142
143             job_body = response.json().get('record', {})
144
145             # Show interesting message updates
146             message = job_body.get('message', '')
147             self.logger.info('Event: ' + message)
148
149             # Refresh the last modified time for the poll loop
150             last_modified = job_body.get('last_modified')
151
152             # Look for the final states
153             state = job_body.get('state', 'unknown')
154             if state in ['success', 'failure']:
155                 if state == 'failure':
156                     self.logger.error('FAILED background job.\nJOB:
157                                     %s', job_body)
158                     exit(1) # End the script if a failure occurs
159                     break
160
161             def exit_on_errors(self, response):
162                 if response.status_code >= 400:
163                     self.logger.error('FAILED request to URL: %s\nHEADERS:
164                                     %s\nRESPONSE BODY: %s',
165                                     response.request.url,
166                                     self.filter_headers(response),
167                                     response.text)
168                     response.raise_for_status() # Displays the response error,
169                     and exits the script
170
171             @staticmethod
172             def filter_headers(response):
173                 ''' Returns a filtered set of the response headers '''
174                 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.

```
1 #!/usr/bin/env python
2 ##-----
3 #
4 # File: resize_nodes.py
5 #
6 # (C) Copyright 2019 NetApp, Inc.
7 #
8 # This sample code is provided AS IS, with no support or warranties of
9 # any kind, including but not limited for warranties of
10 # merchantability
11 # or fitness of any kind, expressed or implied. Permission to use,
12 # reproduce, modify and create derivatives of the sample code is
13 # granted
14 # solely for the purpose of researching, designing, developing and
15 # testing a software application product for use with NetApp products,
16 # provided that the above copyright notice appears in all copies and
17 # that the software application product is distributed pursuant to
18 # terms
19 # no less restrictive than those set forth herein.
20 #
21 ##-----
22
23
24 import argparse
25 import logging
26 import sys
27
28 from deploy_requests import DeployRequests
29
30
31 def _parse_args():
32     """ Parses the arguments provided on the command line when
33     executing this
34     script and returns the resulting namespace. If all required
35     arguments
36     are not provided, an error message indicating the mismatch is
37     printed and
38     the script will exit.
39     """
40
```

```

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
    in the cluster
84     nodes = [node for node in cluster['nodes']]
85     if parsed_args.nodes:
86         nodes = [node for node in nodes if node['name'] in
    parsed_args.nodes]
87
88     changes['nodes'] = [
89         {'instance_type': parsed_args.instance_type, 'id': node['id']}
    for node in nodes]
90
91     return changes
92
93
94 def main():
95     """ Set up the resize operation by gathering the necessary data
    and then send
96         the request to the ONTAP Select Deploy server.
97     """
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())
```

## Copyright information

Copyright © 2024 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.

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.