Script to create a cluster

ONTAP Select

David Peterson November 21, 2019

This PDF was generated from https://docs.netapp.com/us-en/ontap-select/reference_api_script_cc.html on April 21, 2020. Always check docs.netapp.com for the latest.



Table of Contents

Script to create a cluster				
----------------------------	--	--	--	--

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
4 # File: cluster.py
6 # (C) Copyright 2019 NetApp, Inc.
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):
       """ Add credentials for the vcenter if present in the config """
29
30
       log debug trace()
31
32
       vcenter = config.get('vcenter', None)
       if vcenter and not deploy.resource_exists('/security/credentials',
33
                                                  'hostname', vcenter['hostname']):
34
           log_info("Registering vcenter {} credentials".format(vcenter['hostname']))
35
           data = {k: vcenter[k] for k in ['hostname', 'username', 'password']}
36
           data['type'] = "vcenter"
37
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.
           Does nothing if the host credential already exists on the Deploy.
43
44
       log_debug_trace()
45
46
47
       hosts = config.get('hosts', [])
       for host in hosts:
48
           # The presense of the 'password' will be used only for standalone hosts.
49
           # If this host is managed by a vcenter, it should not have a host 'password'
50
   in the json.
           if 'password' in host and not deploy.resource exists('/security/credentials',
51
                                                                  'hostname', host[
52
   'name']):
53
               log info("Registering host {} credentials".format(host['name']))
               data = {'hostname': host['name'], 'type': 'host',
54
                       'username': host['username'], 'password': host['password']}
55
               deploy.post('/security/credentials', data)
56
57
58
59 def register_unkown_hosts(deploy, config):
       ''' Registers all hosts with the deploy server.
60
           The host details are read from the cluster config json file.
61
62
           This method will skip any hosts that are already registered.
63
64
           This method will exit the script if no hosts are found in the config.
65
       log_debug_trace()
66
67
       data = {"hosts": []}
68
       if 'hosts' not in config or not config['hosts']:
69
           log_and_exit("The cluster config requires at least 1 entry in the 'hosts'
70
   list got {}".format(config))
71
72
       missing_host_cnt = 0
       for host in config['hosts']:
73
           if not deploy.resource_exists('/hosts', 'name', host['name']):
74
75
               missing_host_cnt += 1
               host_config = {"name": host['name'], "hypervisor_type": host['type']}
76
               if 'mgmt_server' in host:
77
78
                   host config["management server"] = host['mgmt server']
79
                   log_info(
                      "Registering from vcenter {mgmt_server}".format(**host))
80
81
82
               if 'password' in host and 'user' in host:
83
                   host_config['credential'] = {
                       "password": host['password'], "username": host['user']}
84
85
               log_info("Registering {type} host {name}".format(**host))
86
```

```
87
                data["hosts"].append(host_config)
 88
 89
        # only post /hosts if some missing hosts were found
        if missing_host_cnt:
 90
            deploy.post('/hosts', data, wait_for_job=True)
 91
 92
 93
 94 def add cluster attributes(deploy, config):
        ''' POST a new cluster with all needed attribute values.
 95
            Returns the cluster_id of the new config
 96
 97
 98
        log debug trace()
99
100
        cluster config = config['cluster']
        cluster_id = deploy.find_resource('/clusters', 'name', cluster_config['name'])
101
102
103
        if not cluster id:
            log info("Creating cluster config named {name}".format(**cluster_config))
104
105
106
            # Filter to only the valid attributes, ignores anything else in the json
107
            data = {k: cluster_config[k] for k in [
                'name', 'ip', 'gateway', 'netmask', 'ontap_image_version', 'dns_info',
108
    'ntp servers']}
109
            num_nodes = len(config['nodes'])
110
111
            log_info("Cluster properties: {}".format(data))
112
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):
        ''' Get the the ids of the nodes in a cluster. Returns a list of node_ids.'''
121
        log_debug_trace()
122
123
124
        response = deploy.get('/clusters/{}/nodes'.format(cluster_id))
        node ids = [node['id'] for node in response.json().get('records')]
125
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 '''
        log debug trace()
131
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}
        # Optional: Set a serial_number
137
        if 'license' in node:
138
139
            data['license'] = {'id': node['license']}
140
141
        # Assign the host
        host_id = deploy.find_resource('/hosts', 'name', node['host_name'])
142
        if not host id:
143
            log and exit("Host names must match in the 'hosts' array, and the
144
    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
        data['passthrough_disks'] = not is_hw_raid
150
151
152
        # Optionally set a custom node name
        if 'name' in node:
153
            data['name'] = node['name']
154
155
156
        log_info("Node properties: {}".format(data))
        deploy.patch('/clusters/{}/nodes/{}'.format(cluster_id, node_id), data)
157
158
159
160 def add_node_networks(deploy, cluster_id, node_id, node):
        ''' Set the network information for a node '''
161
162
        log_debug_trace()
163
        log_info("Adding node '{}' network properties".format(node_id))
164
165
        num_nodes = deploy.get_num_records('/clusters/{}/nodes'.format(cluster_id))
166
167
        for network in node['networks']:
168
169
            # single node clusters do not use the 'internal' network
170
            if num nodes == 1 and network['purpose'] == 'internal':
171
                continue
172
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),
                                               'purpose', network['purpose'])
176
177
            data = {"name": network['name']}
            if 'vlan' in network and network['vlan']:
178
```

```
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
        log info("Adding node '{}' storage properties".format(node id))
188
        log info("Node storage: {}".format(node['storage']['pools']))
189
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']:
            data = {'disks': node['storage']['disks']}
196
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)
        node_configs = config['nodes']
208
209
210
        for node_id, node_config in zip(node_ids, node_configs):
211
            add_node_attributes(deploy, cluster_id, node_id, node_config)
            add node networks(deploy, cluster id, node id, node config)
212
            add_node_storage(deploy, cluster_id, node_id, node_config)
213
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
        data = {'ontap credential': {'password': config['cluster'][
223
    '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():
        stack = traceback.extract_stack()
229
230
        parent function = stack[-2][2]
        logging.getLogger('deploy').debug('Calling %s()' % parent_function)
231
232
233
234 def log_info(msg):
        logging.getLogger('deploy').info(msg)
235
236
237
238 def log and exit(msg):
239
        logging.getLogger('deploy').error(msg)
240
        exit(1)
241
242
243 def configure_logging(verbose):
        FORMAT = '%(asctime)-15s:%(levelname)s:%(name)s: %(message)s'
244
245
        if verbose:
246
            logging.basicConfig(level=logging.DEBUG, format=FORMAT)
247
        else:
            logging.basicConfig(level=logging.INFO, format=FORMAT)
248
249
            logging.getLogger('requests.packages.urllib3.connectionpool').setLevel(
250
                logging.WARNING)
251
252
253 def main(args):
254
        configure logging(args.verbose)
        deploy = DeployRequests(args.deploy, args.password)
255
256
257
        with open(args.config_file) as json_data:
258
            config = json.load(json_data)
259
            add_vcenter_credentials(deploy, config)
260
261
262
            add_standalone_host_credentials(deploy, config)
263
            register unkown hosts(deploy, config)
264
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.')
         parser.add_argument('-d', '--deploy', help='Hostname or IP address of Deploy
273
server')
         parser.add_argument('-p', '--password', help='Admin password of Deploy server')
274
         parser.add_argument('-c', '--config_file', help='Filename of the cluster config')
parser.add_argument('-v', '--verbose', help='Display extra debugging messages for
275
276
seeing exact API calls and responses',
                                action='store_true', default=False)
277
278
         return parser.parse_args()
279
280 if __name__ == '__main__':
         args = parseArgs()
281
282
         main(args)
```

Copyright Information

Copyright © 2019–2020 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 systemwithout 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.