import time

import copy

import re

from common.constants import UNDERCLOUD, HOSTS\_CONFIG\_BEFORE\_CHANGES, \

HOSTS\_CONFIG\_TMP, HOSTS\_CONFIG, HOSTS\_CONFIG\_THE\_LAST\_CREATED, CINDER\_CONF,\

HOSTS\_CONFIG\_FOR\_DEBUG

from conf import config

from common.commons import CbisException

from common.constants import StackStates

from flows.base\_deployment import CbisBaseDeployment

from cbis\_common.hosts\_config\_utility import hosts\_config\_utility

SLEEP\_TIME\_SECONDS = 5

WAIT\_INTERVAL\_AFTER\_OSD\_OUT\_SECONDS = 2

MAX\_TIMES = 20

LOOP\_TIMEOUT = 10

class RemoveNode(CbisBaseDeployment):

nodes\_to\_remove = []

ignore\_ceph\_errors = False

hosts\_utility = None

is\_multiple\_pools = False

def \_\_init\_\_(self, logger):

super(RemoveNode, self).\_\_init\_\_(logger)

def set\_request(self, request, page\_json, hardware=None):

super(RemoveNode, self).set\_request(request, page\_json)

self.nodes\_to\_remove = \

self.request['remove\_node\_main']['remove\_node\_params']['node\_names']

self.ignore\_ceph\_errors = \

self.request['remove\_node\_main']['remove\_node\_params'][

'ignore\_ceph\_errors']

def deploy(self):

self.log.info("\*\*Remove node started for hosts: {}\*\*".format(

self.nodes\_to\_remove))

self.log.info("\*\*Collecting information about the system\*\*")

self.check\_if\_tls\_enabled()

self.check\_ssh\_access(self.nodes\_to\_remove)

osds\_to\_delete = self.fetch\_osd\_ids(self.nodes\_to\_remove)

self.log.info("Found Ceph nodes to delete: {}".format(

osds\_to\_delete.keys()))

self.log.info("Found OSDs to delete: {}".format(osds\_to\_delete))

nova\_ids = self.fetch\_nova\_ids\_for\_nodes(self.nodes\_to\_remove)

self.log.info("Found nova ids to remove: {}".format(nova\_ids))

ironic\_ids = self.fetch\_ironic\_ids(nova\_ids)

ipmis = self.get\_ipmi\_of\_removed\_node(nova\_ids)

self.cbis\_helper.cmds\_run\_sync(['scp -o StrictHostKeyChecking=no '

'stack@uc:/home/stack/hosts\_config.yaml'

' {} '

.format(HOSTS\_CONFIG\_BEFORE\_CHANGES)])

self.hosts\_utility = hosts\_config\_utility(HOSTS\_CONFIG\_BEFORE\_CHANGES)

self.is\_multiple\_pools = self.hosts\_utility.is\_multiple\_pools\_enabled()

self.log.info("Found ironic ids to remove: {}".format(ironic\_ids))

self.log.info("\*\*Executing pre-checks\*\*")

self.check\_if\_vms\_are\_running(self.nodes\_to\_remove)

# remove Ceph from all nodes with Ceph installed as step 1

if osds\_to\_delete:

self.log.info("\*\*Executing Ceph checks\*\*")

self.do\_ceph\_checks(osds\_to\_delete, self.ignore\_ceph\_errors, ipmis)

for node in osds\_to\_delete:

self.log.info("\*\*Marking Ceph OSDs out for {} ({})\*\*".format(

node, osds\_to\_delete[node]))

self.mark\_osds\_out(osds\_to\_delete[node])

self.wait\_for\_ceph\_health(self.ignore\_ceph\_errors)

for node in osds\_to\_delete:

self.log.info("\*\*Stopping OSD services for {}\*\*".format(node))

self.stop\_osd\_services(node)

self.log.info("\*\*Removing OSDs for {}\*\*".format(node))

self.completely\_remove\_osds(osds\_to\_delete[node], node, ipmis)

self.log.info("\*\*Waiting for Ceph health after stopping Ceph "

"services and removing OSDs\*\*")

self.wait\_for\_ceph\_health(self.ignore\_ceph\_errors)

# as step 2, remove all nodes from nova and ironic

# at this point, all nodes don't have Ceph

self.log.info("\*\*Removing hosts from host aggregates\*\*")

self.remove\_hosts\_from\_aggregates(self.nodes\_to\_remove)

self.log.info("\*\*Disabling compute services\*\*")

self.disable\_compute\_services(self.nodes\_to\_remove)

self.log.info("\*\*Stack update - removing nodes: {}\*\*".format(nova\_ids))

self.update\_stack(nova\_ids)

self.remove\_from\_hosts\_config(ipmis)

self.remove\_ceph\_user\_from\_nodes(ipmis)

self.cbis\_helper.cmds\_run\_sync([

'scp -o StrictHostKeyChecking=no {} stack@uc:{}'

.format(HOSTS\_CONFIG\_TMP, HOSTS\_CONFIG\_THE\_LAST\_CREATED)])

self.cbis\_helper.ssh\_cmds(UNDERCLOUD, ['sudo mv {} /usr/share/cbis'

.format(HOSTS\_CONFIG\_THE\_LAST\_CREATED)])

self.log.info("\*\*Removing compute services\*\*")

self.remove\_nova\_compute\_services(self.nodes\_to\_remove)

self.log.info("\*\*Removing neutron agents\*\*")

self.remove\_neutron\_agents(self.nodes\_to\_remove)

self.log.info("\*\*Removing ironic instances: {}\*\*".format(ironic\_ids))

self.remove\_from\_ironic(ironic\_ids)

self.log.info("\*\*Executing post-checks\*\*")

self.validate\_ironic(ironic\_ids)

self.validate\_nova\_nodes(self.nodes\_to\_remove)

self.log.info("\*\*Reconfiguring salt roster\*\*")

self.reconfigure\_salt\_roster()

self.log.info("\*\*Regenerating templates\*\*")

self.generate\_templates()

self.log.info("\*\*Regenerating /etc/hosts on undercloud\*\*")

self.regenerate\_etc\_hosts()

self.call\_ceph\_playbook()

self.log.info("\*\*Removing nodes from Zabbix\*\*")

self.remove\_from\_zabbix(self.nodes\_to\_remove)

self.log.info("\*\*Remove node finished successfully\*\*")

self.log.info("\*\*Please backup Undercloud\*\*")

def call\_ceph\_playbook(self):

self.log.info("\*\*Calling Ceph fast-pool playbook\*\*")

self.cbis\_helper.ssh\_cmds(UNDERCLOUD,

["ansible-playbook /usr/share/cbis/cbis-ansible/post-install/ceph-configure-fast-pool.yml"])

self.log.info("\*\*Finished running Ceph fast-pool playbook\*\*")

def do\_ceph\_checks(self, osds\_to\_delete, ignore\_errors, ipmis):

self.log.info("\*\*Checking ceph health\*\*")

health\_ok = self.check\_ceph\_is\_healthy()

if not health\_ok and not ignore\_errors:

raise CbisException("Failed in health\_ok check")

enough\_disk\_space = self.check\_ceph\_disk\_space(osds\_to\_delete)

if not enough\_disk\_space and not ignore\_errors:

raise CbisException("Failed in enough\_disk\_space check")

enough\_replicas = self.check\_ceph\_replicas(osds\_to\_delete, ipmis)

if not enough\_replicas and not ignore\_errors:

raise CbisException("Failed in enough\_replicas check")

pgs\_less\_than\_max\_allowed = self.check\_pgs\_less\_than\_max\_allowed(

osds\_to\_delete)

if not pgs\_less\_than\_max\_allowed and not ignore\_errors:

raise CbisException("Failed in pgs\_less\_than\_max\_allowed check")

if not health\_ok or not enough\_disk\_space or not enough\_replicas or \

not pgs\_less\_than\_max\_allowed:

# if we are here => ignore\_errors == True

self.log.warn(

"Encountered problems with Ceph (health\_ok={}, "

"enough\_disk\_space={}, enough\_replicas={} "

"pgs\_less\_than\_max\_allowed={})".format(health\_ok,

enough\_disk\_space, enough\_replicas,

pgs\_less\_than\_max\_allowed))

msg = "User selected to ignore Ceph problems, continuing..."

self.log.warn(msg)

else:

self.log.debug("Ceph check passed successfully, continuing....")

def check\_ceph\_disk\_space(self, osds\_to\_delete):

osd\_df = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd df"],

as\_json=True)

self.is\_multiple\_pools = self.hosts\_utility.is\_multiple\_pools\_enabled()

if not self.is\_multiple\_pools:

space\_total = osd\_df["summary"]["total\_kb"]

space\_used = osd\_df["summary"]["total\_kb\_used"]

osds\_to\_delete\_list = [osd\_name for host in osds\_to\_delete for

osd\_name in osds\_to\_delete[host] ]

self.log.debug("osds\_to\_delete\_list {}".format(osds\_to\_delete\_list))

return self.calculate\_ceph\_disk\_space(osds\_to\_delete\_list, osd\_df,

space\_total, space\_used)

pool\_osd\_dictionary = self.get\_osd\_pool\_dictionary(osds\_to\_delete)

ceph\_osd\_df\_tree = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd df tree"], as\_json=True)

for pool\_name in pool\_osd\_dictionary:

host\_ids = [host\_id for node in ceph\_osd\_df\_tree["nodes"] if

node["name"] == pool\_name and node["type"] == "root"

for host\_id in node["children"] ]

osd\_ids = [osd\_id for node in ceph\_osd\_df\_tree["nodes"] if

node["id"] in host\_ids for osd\_id in node["children"]]

osds\_to\_delete\_list = [node['name'] for node in

ceph\_osd\_df\_tree["nodes"]

if node['id'] in osd\_ids]

self.log.info("host\_ids {} osds\_to\_delete\_list {}".format(

host\_ids, osds\_to\_delete\_list))

space\_total\_pool = sum([pool["kb"] for pool in

ceph\_osd\_df\_tree["nodes"]

if pool["name"] == pool\_name and

pool["type"] == "root"])

space\_used\_pool = sum([pool["kb\_used"] for pool in

ceph\_osd\_df\_tree["nodes"]

if pool["name"] == pool and

pool["type"] == "root"])

self.log.info("space total pool: {} and the space used pool: {}"

" for the pool: {}"

.format(space\_total\_pool, space\_used\_pool, pool\_name))

if not self.calculate\_ceph\_disk\_space(osds\_to\_delete\_list, osd\_df,

space\_total\_pool,

space\_used\_pool):

return False

return True

def check\_pgs\_less\_than\_max\_allowed(self, osds\_to\_delete):

check\_pgs\_if\_scale\_in = self.cbis\_helper.ssh\_cmds(

UNDERCLOUD,

["/usr/share/cbis/undercloud/tools/update\_ceph\_pgs.sh -c {}".format(

",".join(osds\_to\_delete))])

find\_allow\_sacle\_in = re.search(r'"allow\_scale\_in": (\w+)',

check\_pgs\_if\_scale\_in)

if find\_allow\_sacle\_in:

res = find\_allow\_sacle\_in.group(1)

return res.lower() == "true"

return False

def get\_osd\_pool\_dictionary(self, osds\_to\_delete):

pool\_osd\_dictionary = {}

for osds in osds\_to\_delete.values():

for osd in osds:

osd\_pool\_information = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd find {} ".format(osd)], as\_json=True)

pool\_name = osd\_pool\_information["crush\_location"]["root"]

pool\_osd\_dictionary.setdefault(pool\_name, []).append(osd)

self.log.info("The pool:osds dictionary is: {}".format(pool\_osd\_dictionary))

return pool\_osd\_dictionary

def calculate\_ceph\_disk\_space(self, osds\_to\_delete\_list, osd\_df,

space\_total, space\_used):

space\_to\_delete = sum([osd["kb"]

for osd in osd\_df["nodes"]

if osd["name"] in osds\_to\_delete\_list])

space\_used\_osds = sum([osd["kb\_used"]

for osd in osd\_df["nodes"]

if osd["name"] in osds\_to\_delete\_list])

self.log.debug("Calculated space to delete = {:0.2f} GB".format(

space\_to\_delete / (1024\*1024)))

self.log.debug("Total space after deletion = {:0.2f} GB".format(

(space\_total - space\_to\_delete) / (1024\*1024)))

self.log.debug("Maximum capacity after operation = {:0.2f} GB".format(

(space\_total - space\_to\_delete) \* 0.85 / (1024\*1024)))

self.log.debug("Space used = {} GB space\_used\_osds {} GB".format(

space\_used / (1024\*1024), space\_used\_osds / (1024\*1024)))

return (space\_total - space\_to\_delete) \* 0.85 >= (space\_used -

space\_used\_osds)

def check\_ceph\_replicas(self, osds\_to\_delete, ipmis=None):

osd\_dump = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd dump"], as\_json=True)

if not self.is\_multiple\_pools:

nodes\_to\_delete\_cnt = len(osds\_to\_delete.keys())

max\_pool\_size\_common = max(map(

lambda pool: pool["size"], [pool\_ for pool\_ in osd\_dump["pools"]

if pool\_["pool\_name"] != "volumes-fast"]))

ceph\_nodes = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph node ls osd"], as\_json=True)

all\_nodes\_cnt\_common = len([node\_name for node\_name in

ceph\_nodes.keys() if not

node\_name.startswith("fast-")])

replica\_not\_fail\_common, at\_least\_one\_osd\_common = \

self.calculate\_ceph\_replicas(

max\_pool\_size\_common, all\_nodes\_cnt\_common, nodes\_to\_delete\_cnt)

return replica\_not\_fail\_common and at\_least\_one\_osd\_common

self.log.info("Check ceph replica when multiple pools enabled")

counter\_multiple\_pools = {}

ceph\_osd\_df\_tree = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd df tree"], as\_json=True)

for ipmi in ipmis:

pool\_names = self.hosts\_utility.get\_pool\_names(ipmi)

for pool in pool\_names:

if pool in counter\_multiple\_pools:

counter\_multiple\_pools[pool] = \

counter\_multiple\_pools[pool] + 1

else:

counter\_multiple\_pools[pool] = 1

self.log.info("The deleted pools are: {}".format(

counter\_multiple\_pools.keys()))

at\_least\_one\_osd = False

for pool\_name in counter\_multiple\_pools.keys():

all\_nodes\_pool\_cnt = sum([len(pool["children"])

for pool in ceph\_osd\_df\_tree["nodes"]

if pool["name"] == pool\_name and

pool["type"] == "root"])

pool\_size = max([pool["size"]

for pool in osd\_dump["pools"]

if pool["pool\_name"] == pool\_name])

self.log.info("The number of nodes which contribute to "

"pool: {} is: {}".format(pool\_name, all\_nodes\_pool\_cnt

))

replica\_not\_fail, at\_least\_one\_osd\_for\_pool = \

self.calculate\_ceph\_replicas(pool\_size,all\_nodes\_pool\_cnt,

counter\_multiple\_pools[pool\_name])

self.log.debug("pool\_name {} replica\_not\_fail {} "

"at\_least\_one\_osd\_for\_pool {}".format(pool\_name,

replica\_not\_fail, at\_least\_one\_osd\_for\_pool))

if not replica\_not\_fail:

return False

if at\_least\_one\_osd\_for\_pool:

at\_least\_one\_osd = True

return at\_least\_one\_osd

def calculate\_ceph\_replicas(self, max\_pool\_size,

all\_nodes\_cnt, nodes\_to\_delete\_cnt):

self.log.debug(

"Max pool size: {}, number of nodes after deletion: {}".format(

max\_pool\_size, all\_nodes\_cnt - nodes\_to\_delete\_cnt))

at\_least\_one\_osd = all\_nodes\_cnt - nodes\_to\_delete\_cnt != 0

replica\_not\_fail = True

if at\_least\_one\_osd:

replica\_not\_fail = \

all\_nodes\_cnt - nodes\_to\_delete\_cnt >= max\_pool\_size

return replica\_not\_fail, at\_least\_one\_osd

def get\_ipmi\_of\_removed\_node(self, nova\_instance\_ids):

ipmis = []

ironic\_nodes = self.cbis\_helper.ssh\_cmds\_json(

UNDERCLOUD,

["openstack baremetal node list"]

)

for nova\_instance\_id in nova\_instance\_ids:

self.log.debug(

"Following ironic instances will be removed: {}".format(

nova\_instance\_ids))

server\_id = next(node["Name"] for node in ironic\_nodes

if node["Instance UUID"] == nova\_instance\_id)

cmd = "openstack baremetal node show " + server\_id

node\_info = self.cbis\_helper.ssh\_cmds\_json(UNDERCLOUD, [cmd])

ipmi = node\_info["driver\_info"]["ipmi\_address"]

ipmis.append(ipmi)

self.log.debug("The IPMIs of the removed nodes are: {}".format(ipmis))

return ipmis

def check\_if\_vms\_are\_running(self, nodes):

vms\_on\_nodes\_to\_be\_deleted = filter(

lambda vm: vm["Host"] in self.get\_hostnames\_for\_nova\_nodes(nodes),

self.fetch\_overcloud\_server\_list())

if vms\_on\_nodes\_to\_be\_deleted:

msg = "Found VMs running on nodes to be removed. Following VMs " \

"need to be evacuated/deleted before continuing: " \

"{}".format(vms\_on\_nodes\_to\_be\_deleted)

self.log.error(msg)

raise CbisException(msg)

def mark\_osds\_out(self, osd\_ids):

ceph\_status = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph -s"], as\_json=True)

is\_norebalance = "norebalance flag(s) set" in str(ceph\_status)

if not is\_norebalance:

self.log.debug("Setting no rebalance and start getting out OSDs")

self.cbis\_helper.execute\_on\_active\_controller(

["ceph osd set norebalance "])

else:

self.log.debug("Start getting out OSDs. (norebalance is already set)")

for osd in osd\_ids:

self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd out " + osd])

time.sleep(WAIT\_INTERVAL\_AFTER\_OSD\_OUT\_SECONDS)

if not is\_norebalance:

self.cbis\_helper.execute\_on\_active\_controller(

["ceph osd unset norebalance "])

self.log.info("After marking OSDs out")

def fetch\_osd\_ids(self, nodes):

node\_ls\_osd = self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph node ls osd"], as\_json=True)

osd\_ids = {}

for node in nodes:

osds\_for\_node = []

for ceph\_host in node\_ls\_osd:

if ceph\_host.endswith(node):

osds\_for\_node.extend(

map(lambda osd\_num: "osd." + str(osd\_num),

node\_ls\_osd[ceph\_host]))

if osds\_for\_node:

osd\_ids[node] = osds\_for\_node

return osd\_ids

def stop\_osd\_services(self, node):

try:

# we need to go through all services separately because of some bug

# in ceph-osd.target

services\_to\_stop = self.cbis\_helper.execute\_on\_host(

node, ["systemctl | "

"grep ceph-osd@.\*.service | "

"grep -v ceph-osd.target | "

"awk '{print $1}' | "

"xargs echo"]).split()

for service in services\_to\_stop:

self.cbis\_helper.execute\_on\_host(

node, ["sudo systemctl stop {}".format(service)])

time\_out = time.time() + config.SYSTEMCTL\_TIMEOUT

is\_active = True

while is\_active and time.time() < time\_out:

is\_active = int(self.cbis\_helper.execute\_on\_host(

node,

["sudo docker ps | grep ceph-osd | wc -l"])) != 0

time.sleep(LOOP\_TIMEOUT)

if is\_active:

self.log.warn("Ceph services still running after timeout")

except Exception as e:

self.log.warn("Can't stop Ceph service on the node {}: {}".format(

node, e))

def remove\_from\_hosts\_config(self, ipmis):

self.log.debug("copy the hosts\_config.yaml from UC")

self.cbis\_helper.cmds\_run\_sync([

'scp -o StrictHostKeyChecking=no stack@uc:/'

'home/stack/hosts\_config.yaml {} '.format(HOSTS\_CONFIG\_TMP)])

hosts\_config\_dictionary = self.cbis\_helper.get\_dict\_from\_file(

HOSTS\_CONFIG\_TMP)

tmp\_json = copy.deepcopy(hosts\_config\_dictionary)

for host\_group in tmp\_json['host\_groups']:

hostgroup\_index = hosts\_config\_dictionary[

'host\_groups'].index(host\_group)

if 'pm\_addr' in host\_group and host\_group['pm\_addr'] is not None:

for ip in host\_group['pm\_addr']:

for ipmi in ipmis:

if ip == ipmi:

if len(hosts\_config\_dictionary['host\_groups'][

hostgroup\_index]['pm\_addr']) == 1:

self.log.debug(

"remove the hosts group section: {} "

"from hosts\_config".format(

host\_group['host\_group']))

del hosts\_config\_dictionary['host\_groups'][

hostgroup\_index]

else:

self.log.debug(

'remove the ip: {} from hosts\_config,'

' host\_group index is: {}'

.format(ipmi, hostgroup\_index))

hosts\_config\_dictionary['host\_groups'][

hostgroup\_index]['pm\_addr'].remove(ipmi)

self.log.debug("hosts\_config dict after changes: {}".format(

hosts\_config\_dictionary))

self.cbis\_helper.write\_dict\_to\_file(HOSTS\_CONFIG\_TMP,

hosts\_config\_dictionary)

self.cbis\_helper.cmds\_run\_sync([

'scp -o StrictHostKeyChecking=no {} stack@uc:{}'

.format(HOSTS\_CONFIG\_TMP, HOSTS\_CONFIG)])

def completely\_remove\_osds(self, osd\_ids, node, ipmis):

self.log.warn("If this operation fails you may need to "

"power off the node of which the"

" OSD was associated to by running:\n "

" baremetal node power off <server-name> \n"

"and baremetal node maintenance set <server-name")

for osd in osd\_ids:

self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd crush remove {}".format(osd),

"sudo ceph auth del {}".format(osd),

"sudo ceph osd rm {}".format(osd)]

)

# TODO: Noam Sitton - read the field as yaml and search for the value

fast\_pool\_in\_user\_config = self.cbis\_helper.ssh\_cmds(

UNDERCLOUD,

["grep enable\_fast\_pool /home/stack/templates/user\_config.yaml"

" | awk '{ print $2 }'"]).strip() == "true"

if self.is\_multiple\_pools:

for ipmi in ipmis:

self.log.info("remove multiple pool hostname from crush map")

pool\_list = self.hosts\_utility.get\_pool\_names(ipmi)

if pool\_list:

for pool in pool\_list:

cmd = '"sudo ceph osd tree | grep -w ' + "'host {}-{}'" \

" && sudo ceph osd crush remove {}-{} || echo " \

"'Skipping unexisting node" \

" entry'".format\

(pool, node.lower(), pool, node.lower()) + '"'

self.cbis\_helper.execute\_on\_active\_controller([cmd])

cmd = '"sudo ceph osd tree | grep -w ' + "'host {}'" \

" && sudo ceph osd crush remove {} || echo " \

"'Skipping unexisting node" \

" entry'".format(node.lower(), node.lower()) + '"'

self.cbis\_helper.execute\_on\_active\_controller([cmd])

delete\_pool\_from\_crush\_map = self.hosts\_utility.\

get\_deleted\_pools(ipmis)

for pool in delete\_pool\_from\_crush\_map:

self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd pool rm {0} {0} "

"--yes-i-really-really-mean-it".format(pool),

"sudo ceph osd crush rule rm"

" {}\_replicated\_ruleset".format(pool),

"sudo ceph osd crush remove {}".format(pool)])

elif fast\_pool\_in\_user\_config:

self.log.info("remove fast pool hostname from crush map")

self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd crush remove common-{}".format(node.lower()),

"sudo ceph osd crush remove fast-{}".format(node.lower())])

else:

self.log.info("remove hostname from crush map")

self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph osd crush remove {}".format(node.lower())])

def remove\_hosts\_from\_aggregates(self, nodes):

for aggregate in self.fetch\_overcloud\_aggregates():

aggregate\_details = self.cbis\_helper.execute\_on\_overcloud(

["openstack aggregate show {}".format(aggregate["ID"])],

as\_json=True)

hosts\_to\_remove\_from\_aggregate = filter(

lambda hostname: hostname in aggregate\_details["hosts"],

self.get\_hostnames\_for\_nova\_nodes(nodes))

for host in hosts\_to\_remove\_from\_aggregate:

self.cbis\_helper.execute\_on\_overcloud(

["openstack aggregate remove host {} {}".format(

aggregate["ID"], host)])

def disable\_compute\_services(self, nodes):

services\_to\_disable = filter(

lambda serv:

serv["Host"] in self.get\_hostnames\_for\_nova\_nodes(nodes),

self.fetch\_overcloud\_compute\_services())

for service in services\_to\_disable:

self.cbis\_helper.execute\_on\_overcloud(

["openstack compute service set --disable {} {}".format(

service["Host"], service["Binary"])])

def update\_stack(self, nova\_instance\_ids):

cmd = "openstack cbis node remove {}".format(str(" ".join(nova\_instance\_ids)))

self.cbis\_helper.ssh\_cmds(UNDERCLOUD, [cmd])

self.validate\_heat\_output([StackStates.UPDATE\_COMPLETE],

wait\_for\_progress=False)

non\_deleted\_nodes = nova\_instance\_ids

counter = 0

while non\_deleted\_nodes and counter < MAX\_TIMES:

self.log.info("Waiting for nova instances {} to be removed".format(

nova\_instance\_ids))

ironic\_nodes = self.fetch\_ironic\_nodes()

non\_deleted\_nodes = filter(

lambda node: node["Instance UUID"] in nova\_instance\_ids,

ironic\_nodes)

time.sleep(SLEEP\_TIME\_SECONDS)

counter += 1

if counter >= MAX\_TIMES:

msg = "Some of the nova instances are not removed ({}), aborting " \

"operation.".format(nova\_instance\_ids)

self.log.error(msg)

raise CbisException(msg)

def remove\_from\_ironic(self, ironic\_ids):

cmd = []

for id in ironic\_ids:

cmd.append("openstack baremetal node maintenance set " + id)

cmd.append("openstack baremetal node delete {}".format(" ".join(ironic\_ids)))

self.cbis\_helper.ssh\_cmds(

UNDERCLOUD,

cmd)

def remove\_nova\_compute\_services(self, nodes):

hostnames = self.get\_hostnames\_for\_nova\_nodes(nodes)

# remove services for all nodes

services\_to\_remove = filter(

lambda service: service["Host"] in hostnames,

self.fetch\_overcloud\_compute\_services())

service\_ids = map(lambda service: service["ID"], services\_to\_remove)

# Making sure there are services on the removed node/s (for storage

# nodes there are no services)

if len(service\_ids) > 0:

self.cbis\_helper.execute\_on\_overcloud(

["openstack compute service delete {}".format(

" ".join(map(str, service\_ids)))])

# verifying services were removed

services\_not\_removed = filter(

lambda service: service["Host"] in hostnames,

self.fetch\_overcloud\_compute\_services())

if services\_not\_removed:

msg = "Following services were not removed from nova service " \

"list: {}".format(services\_not\_removed)

self.log.error(msg)

raise CbisException(msg)

def remove\_neutron\_agents(self, nodes):

hostnames = self.get\_hostnames\_for\_nova\_nodes(nodes)

# remove agents for all nodes

agents\_to\_remove = filter(lambda agent: agent["Host"] in hostnames,

self.fetch\_overcloud\_network\_agents())

agent\_ids = map(lambda agent: agent["ID"], agents\_to\_remove)

# Making sure there are network agents on the removed node/s (for

# storage nodes there are no agents)

if len(agent\_ids) > 0:

self.cbis\_helper.execute\_on\_overcloud(

["openstack network agent delete {}".format(" ".join(agent\_ids))])

# verifying agents were removed

agents\_not\_removed = filter(lambda agent: agent["Host"] in hostnames,

self.fetch\_overcloud\_network\_agents())

if agents\_not\_removed:

msg = "Following neutron agents were not removed: {}".format(

agents\_not\_removed)

self.log.error(msg)

raise CbisException(msg)

def reconfigure\_salt\_roster(self):

self.cbis\_helper.ssh\_cmds(

UNDERCLOUD,

["/bin/bash "

"/usr/share/cbis/undercloud/tools/configure\_salt\_build\_roster.sh"])

def validate\_ironic(self, ironic\_ids):

non\_deleted\_nodes = filter(lambda node: node["UUID"] in ironic\_ids,

self.fetch\_ironic\_nodes())

if non\_deleted\_nodes:

msg = "Some of ironic instances were not removed successfully: " \

"{}".format(non\_deleted\_nodes)

self.log.error(msg)

raise CbisException(msg)

def validate\_nova\_nodes(self, nodes):

non\_deleted\_instances = filter(

lambda instance: instance["Name"] in nodes,

self.fetch\_nova\_server\_list())

if non\_deleted\_instances:

msg = "Some nova instances were not removed successfully: " \

"{}".format(non\_deleted\_instances)

self.log.error(msg)

raise CbisException(msg)

def fetch\_nova\_ids\_for\_nodes(self, nodes):

return [nova\_instance["ID"]

for nova\_instance in self.fetch\_nova\_server\_list()

if nova\_instance["Name"] in nodes]

def fetch\_ironic\_ids(self, nova\_ids):

return [node["UUID"]

for node in self.fetch\_ironic\_nodes()

if node["Instance UUID"] in nova\_ids]

def fetch\_nova\_server\_list(self):

return self.cbis\_helper.ssh\_cmds\_json(UNDERCLOUD,

["openstack server list"])

def fetch\_ironic\_nodes(self):

return self.cbis\_helper.ssh\_cmds\_json(UNDERCLOUD,

["openstack baremetal node list"])

def fetch\_overcloud\_compute\_services(self):

return self.cbis\_helper.execute\_on\_overcloud(

["openstack compute service list"],

as\_json=True)

def fetch\_overcloud\_server\_list(self):

return self.cbis\_helper.execute\_on\_overcloud(

["openstack server list --all --long"],

as\_json=True)

def fetch\_overcloud\_aggregates(self):

return self.cbis\_helper.execute\_on\_overcloud(

["openstack aggregate list"],

as\_json=True)

def fetch\_overcloud\_network\_agents(self):

return self.cbis\_helper.execute\_on\_overcloud(

["openstack network agent list"],

as\_json=True)

def remove\_ceph\_user\_from\_nodes(self, ipmis):

if self.is\_multiple\_pools:

self.log.info("Multiple pools enabled. Check and remove ceph user ")

delete\_ceph\_user\_pool = self.hosts\_utility.get\_deleted\_pools(ipmis)

if delete\_ceph\_user\_pool:

for pool in delete\_ceph\_user\_pool:

self.unset\_uuid\_secret\_active\_controllers(pool)

self.cbis\_helper.execute\_on\_active\_controller(

["sudo ceph auth del client.{}".format(pool)])

self.cbis\_helper.execute\_on\_overcloud(

["cinder type-delete tripleo-ceph-{}".format(pool)])

self.cbis\_helper.execute\_on\_all\_active\_controllers(

["sudo crudini --del {} "

"'tripleo-ceph-{}'".format(CINDER\_CONF, pool),

"sudo rm -f /etc/ceph/ceph.client.{}.keyring ".format

(pool), "sudo rm -f /etc/nova/secret\_{}.xml".format

(pool)])

self.log.info("Delete the ceph user of the: {} ".

format(pool))

enable\_backends\_list = self.cbis\_helper.execute\_on\_active\_controller(

["sudo crudini --get {} "

"'DEFAULT' 'enabled\_backends'".format(CINDER\_CONF)]).split(",")

enable\_backends\_list = map(str.strip, enable\_backends\_list)

hosts\_utility\_new = \

hosts\_config\_utility(src\_hosts=HOSTS\_CONFIG\_TMP)

list\_pool\_for\_enabled\_backend = hosts\_utility\_new. \

get\_all\_pool\_names()

prefix\_of\_enabled\_backend = "tripleo-ceph-"

str\_vm\_pool = " "

if list\_pool\_for\_enabled\_backend:

str\_vm\_pool = list\_pool\_for\_enabled\_backend[0]

for pool in delete\_ceph\_user\_pool:

if "tripleo-ceph-{}".format(pool) in enable\_backends\_list:

enable\_backends\_list.remove("tripleo-ceph-{}".format(pool))

str\_pool\_for\_enabled\_backend = enable\_backends\_list[0]

for item in enable\_backends\_list[1:]:

str\_pool\_for\_enabled\_backend = \

str\_pool\_for\_enabled\_backend + "," + item

self.log.info("Edit the cinder.conf file")

self.cbis\_helper.execute\_on\_all\_active\_controllers(

['sudo crudini --set {} "DEFAULT" '

'"enabled\_backends" "{}"'.format

(CINDER\_CONF, str\_pool\_for\_enabled\_backend),

'sudo crudini --set {0} "DEFAULT" "default\_volume\_type"'

' "{1}{2}"'.format(CINDER\_CONF, prefix\_of\_enabled\_backend,

str\_vm\_pool)])

self.log.info("Restart openstack cinder services")

self.cbis\_helper.execute\_on\_active\_controller(

['sudo pcs resource restart openstack-cinder-volume',

'sudo docker restart cinder\_api',

'sudo docker restart cinder\_scheduler'])

def unset\_uuid\_secret\_active\_controllers(self, pool):

self.log.info("Executing unset uuid command on all active controllers")

all\_active\_controllers = self.cbis\_helper.find\_all\_active\_controllers()

for controller in all\_active\_controllers:

uuid = self.cbis\_helper.execute\_on\_host(

controller, ['sudo cat /etc/nova/secret\_{}.xml '

'| grep "uuid" | cut -d ">" -f2 '

'| cut -d "<" -f1'.format(pool)])

self.cbis\_helper.execute\_on\_host(

controller, ['sudo virsh secret-undefine {}'.format(uuid)])

@staticmethod

def get\_hostnames\_for\_nova\_nodes(nodes):

return map(lambda node: node.lower() + ".localdomain", nodes)

def regenerate\_etc\_hosts(self):

self.cbis\_helper.ssh\_cmds(UNDERCLOUD, [

"sudo /usr/share/cbis/cbis-ansible/tools/setup-ansible.sh",

"ansible-playbook /usr/share/cbis/cbis-ansible/post-install/configure-etc-hosts.yml"

])

def remove\_from\_zabbix(self, nodes\_to\_remove):

for node in nodes\_to\_remove:

self.cbis\_helper.ssh\_cmds(UNDERCLOUD, [

"/usr/share/cbis/undercloud/tools/zabbix/zabbix\_delete\_host.py {}".format(node)])