```
import sys,os
import logging
import json,getpass
import distutils.spawn
import datetime,random
import subprocess
import time
import requests,shlex
from requests.auth import HTTPDigestAuth
from datetime import datetime, timedelta

from azure.common.credentials import ServicePrincipalCredentials
from azure.mgmt.resource import ResourceManagementClient
from azure.mgmt.compute import ComputeManagementClient
from azure.mgmt.network import NetworkManagementClient
from azure.mgmt.storage import StorageManagementClient

from azure.mgmt.compute.models import DiskCreateOption
from azure.storage.blob import BlockBlobService, PublicAccess
from azure.cosmosdb.table.tableservice import TableService

from azure.mgmt.authorization import AuthorizationManagementClient
script_dir = os.path.dirname(os.path.realpath(__file__))


#TODO: move all the params to .json files and fetch them


class Utils():
    name=""
    credentials=""
    resource_client=""
    compute_client=""
    network_client=""
    storage_client = ""

    def __init__(self,name):
        self.name=name




    #Create a logger for debugging purposes
    def createLogger(self):

        logFolder = script_dir + "/../logs/"

        logFile = logFolder + self.name + '.log'
        logging.basicConfig(filename=logFile, level=logging.INFO)
        logger = logging.getLogger(self.name)
        handler = logging.StreamHandler()
        formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
        handler.setFormatter(formatter)
        logger.addHandler(handler)
        logger.info("Please find the logs of this job at: "+script_dir+"/"+logFile)

        return logger


    #creates and returns a credentials object .
    # This object will be used later for login and creating resources in Azure
    def getCredentials(self,data):
        self.credentials = ServicePrincipalCredentials(
            client_id=data['AZURE_CLIENT_ID'],
            secret=data['AZURE_SECRET'],
            tenant=data['AZURE_TENANT']
        )



    #Function to read a given json input
    def read_json_file(self, logger, file_name=None):
        data = ""
        with open(file_name) as data_file:
            try:
                data = json.load(data_file)
            except Exception as e:
                logger.error(" Exception occured while reading json file :" + str(e))
        return data

    def read_file(self, logger, file_name=None):
        data = ""
        with open(file_name) as data_file:
            try:
                data = data_file.read()
            except Exception as e:
                logger.error(" Exception occured while reading file :" + str(e))
        return data


    #Creates all the resources required before creating a VM
    #eg: For now we only create  Vnet, subnet, NIC but might have to add more resources accordingly
    def prepareandcreateVM(self,VM_NAME, data,resource_group,logger,releaseinfo=None):
        logger.info("*****************VM Creation Started ***************")

        logger.info("For VM creation, we need NIC ID. Hence creating it")
        nic=self.create_network_interface(resource_group,data, VM_NAME,logger)

        logger.info("Completed Creation of Network Interface. NIC ID:"+nic.id+"\n")
        if releaseinfo:
            vm_parameters = self.create_vm_parameters(resource_group,VM_NAME, data,nic.id,logger,releaseinfo)
        else:
            vm_parameters = self.create_vm_parameters(resource_group,VM_NAME, data,nic.id,logger)

        logger.info("\t VM Parameters:")
        logger.info(vm_parameters)
        vm_poller = self.createVM(resource_group,data, VM_NAME, vm_parameters,logger)
        vm_poller.wait()
        logger.info("\t Done creating VM .. Retrieving and sending the publiciP")
        return self.getpublic_ip(resource_group,VM_NAME)


    def create_network_interface(self,resource_group,
data,VM_NAME,logger,public_ip_allocation_method=None,private_ip_allocation_method=None):
        if public_ip_allocation_method is None:
            public_ip_allocation_method="Static"
        if private_ip_allocation_method is None:
            private_ip_allocation_method="dynamic"

        logger.info("\t Generating NIC id. For that we need Subnet id and Public Ip Address.Hence creating them first")
        subnet = self.create_virtual_network(resource_group,data,VM_NAME,logger)

        logger.info("\t \t Virtual Network creation completed...Subnet : ")
        logger.info(subnet)
        self.create_public_ip(resource_group,data,VM_NAME,public_ip_allocation_method,logger)
        logger.info("\t \t Public IP creation completed...")
        public_ip = self.getpublic_ip(resource_group,VM_NAME)

        params_create = {
            'location': data['REGION'],
            'ip_configurations': [{
                'name': VM_NAME+"-ip",
                'private_ip_allocation_method': private_ip_allocation_method,
                'subnet': subnet,
                'public_ip_address': {
                    'id': public_ip.id
                }
            }]
        }
        nic_poller = self.network_client.network_interfaces.create_or_update(
            resource_group,
            VM_NAME+"-nic",
            params_create,
        )
        return nic_poller.result()
    def create_virtual_network(self,resource_group,data,VnetName,logger):
        logger.info("\t \t ONE:Creating Virtual Network")
        params_create = {
            'location': data['REGION'],
            'address_space': {
                'address_prefixes': ['10.0.0.0/16'],
            },
            'subnets': [{
                'name': VnetName+"-subnet",
                'address_prefix': '10.0.0.0/24',
            }]
        }
        vnet_poller = self.network_client.virtual_networks.create_or_update(
            resource_group,
            VnetName+"-Vnet",
            params_create,
        )
        vnet_poller.wait()

        return self.network_client.subnets.get(
            resource_group,
            VnetName + "-Vnet",
            VnetName + "-subnet",
        )


    def create_resourceGroup(self,name,data,logger):
        logger.info("Creating Resource Group:"+name)
        resource_group_params = {
            'location': data['REGION']
        }
        self.resource_client.resource_groups.create_or_update(name, resource_group_params)


    def delete_resourceGroup(self,name,logger):
        self.resource_client.resource_groups.delete(name)




    def create_public_ip(self,resource_group,data,VM_NAME,public_ip_allocation_method,logger):
        logger.info("\t \t  TWO:Creating publicIp")

        params_create = {
            'location': data['REGION'],
            'public_ip_allocation_method': public_ip_allocation_method,
        }
        pip_poller = self.network_client.public_ip_addresses.create_or_update(
            resource_group,
            VM_NAME+"-publicip",
            params_create,
        )
        return pip_poller.result()


    def getpublic_ip(self,resource_group,VM_NAME):

        publicip = self.network_client.public_ip_addresses.get(
            resource_group,
            VM_NAME + "-publicip")

        return publicip




    #create the VM using the Compute Management Client
    def createVM(self,resource_group,data, VM_NAME, vm_parameters,logger):
        try:

            vm_poller=self.compute_client.virtual_machines.create_or_update(
                resource_group,
                VM_NAME,
                vm_parameters,
            )
            return vm_poller

        except Exception as e:
            logger.info("Exception occured during virtual_machines.create_or_update:" + str(e))




    #Creates and returns the Parameter string required for VM creation

    def create_vm_parameters(self,resource_group,VM_NAME, data,nic_id,logger=None,releaseinfo=None):
        logger.info("Entered createVMParams")
        VHD_url=""
        storage_profile = ""

        if releaseinfo is None :
                imageid = "/subscriptions/" + data['AZURE_SUB'] + "/resourceGroups/" + resource_group +
"/providers/Microsoft.Compute/images/" + data['BASE_IMAGE']
                osDiskdetails = self.generateosDiskdetails(VM_NAME)
                logger.info("osDiskdetails:")
                logger.info(osDiskdetails)
                storage_profile = self.getstoragedetails(imageid, "use",logger)

        else:
            if releaseinfo['CREATE_VHD_URL']=="true":
                logger.info("Creating new VHD Url")
                imageid = "/subscriptions/" + data['AZURE_SUB'] + "/resourceGroups/" + resource_group +
"/providers/Microsoft.Compute/images/" + data['BASE_IMAGE']
                logger.info("Using Imageid:"+imageid)

                storage_url = "https://" + data[
                    'StorageAccount'] + ".blob.core.windows.net/vhds/sample.vhd"
                storage_url="https://mlqavhds.blob.core.windows.net/vhds/sortoutstuff.vhd"
                logger.info("osDisk_VHD_uri:" + storage_url)

                osDiskdetails = self.generateosDiskdetails(VM_NAME,storage_url,"create")
                storage_profile = self.getstoragedetails(imageid, "use",logger)

            else:
                imageid=releaseinfo['USE_VHD_URL']
                osDiskdetails = self.generateosDiskdetails(VM_NAME,releaseinfo['USE_VHD_URL'],"use")
                storage_profile = self.getstoragedetails(imageid, "no",logger)
                logger.info("logger:"+storage_profile)

        logger.info("Using the image:"+imageid)
        """Create the VM parameters structure.
        """
        SSH_PUB = script_dir + data['SSH_PUB']
        return {
            'location': data['REGION'],
            'os_profile': {
                'computer_name': VM_NAME,
                'admin_username': data['VM_Username'],
                "linuxConfiguration": {
                    "ssh": {
                        "publicKeys": [
                            {
                                "path": "/home/builder-temp/.ssh/authorized_keys",
                                "keyData": data['SSH_PUB']
                            }
                        ]
                    },
                }
            },
            'hardware_profile': {
                'vm_size': data['VM_SIZE']
            },
            "storageProfile": {'osDisk':osDiskdetails ,
                               'imageReference':{'id':'/subscriptions/b143bcf7-e5a4-4ec1-a06e-
0e8361700ccb/resourceGroups/QA/providers/Microsoft.Compute/images/vmimage-qa-centos74-b9_0-07-02-2019'}
```

```
                    },
            "network_profile": {
                "networkInterfaces": [
                    {
                        "id": nic_id

                    }
                ]
            }
        }


    def generateosDiskdetails(self,VM_NAME,VHD_url=None,type=None):
        if VHD_url is None:
            return {
            "osType": "Linux",
            "name": VM_NAME + "osDisk",
            "createOption": "FromImage",
            "caching": "ReadWrite"
            }
        else:
            if type == "create":
                return {
                    "osType": "Linux",
                    "name": "sortoutstuff",
                    "createOption": "FromImage",
                    "caching": "ReadWrite",
                    "vhd": {
                        "uri": VHD_url
                    }
                }
            elif type == "use":
                return {
                    "osType": "Linux",
                    "name": VM_NAME + "osDisk",
                    "createOption": "Attach",
                    "caching": "ReadWrite",
                    "image": {
                        "uri": VHD_url
                    }
                }
    def getstoragedetails(self,id,type,logger):
        if type=="use":
            storageInfo= '\'id\':\''+id+'\''
            logger.info("storageInfo: "+storageInfo)
            return storageInfo

        else:
            return ""
    #checks if all executables required for this script are installed on the host
    def checkExecutables(self,logger):
        logger.info(distutils.spawn.find_executable("az"))
        if (not (distutils.spawn.find_executable("az"))):
            logger.error("AZ is not installed. Please install it on the current Host and rerun")
            #sys.exit(0)
        else:
            logger.info("AZ is installed")


    #validates the given json
    def validatejson(self,json_file,useForce,filetype,logger):
        logger.info("Checking for json file:"+json_file)
        if (not(os.path.exists(json_file))):
            logger.info("The "+filetype+" Json File does not exist")
            if(useForce=="false"):
                logger.info("Use Force is set to false. Hence using the standard "+filetype+"  Json")
                if(filetype=="config"):
                    json_file=script_dir+"/../resources/initialize/config.json"
                    return json_file
                elif (filetype=="releaseconfig"):
                    json_file = script_dir + "/../resources/initialize/releaseconfig.json"
                    return json_file
                else:
                    json_file = script_dir + "/../resources/initialize/testing_params.json"
                    return json_file

            else:
                logger.error("Use Force is set to true. But the given config file does not exist. Hence exiting ")
                sys.exit(0)
        else:
            return json_file
            logger.info("Given Config File looks good. Working with it:"+json_file)


    def resource_exists(self,azure_resource_id):
        try:
            return self.resource_client.resources.check_existence_by_id(azure_resource_id,"2017-12-01")
        except Exception as e:
            if e.status_code == 405:  # HEAD not supported
                try:
                    self.resource_client.resources.get_by_id(azure_resource_id,"2017-12-01")
                    return True
                except Exception:
                    return False  # Likely 404, might want to test it explicitly
            raise  # If not 405, not expected
```

```
#check and returns the image to be created
def getImageName(self,data,logger):
    date = datetime.datetime.today().strftime('%d-%m-%Y')
    image_path = "/subscriptions/b143bcf7-e5a4-4ec1-a06e-0e8361700ccb/resourceGroups/QA/providers/Microsoft.Compute/images/"
    BASE_IMAGE_NEW = "vmimage-qa-" + data['IMG_LBL'] + "-" + data['BRANCH'] + "-" + date
    logger.info("Checking if the base image "+BASE_IMAGE_NEW+" already exists")
    azure_resource_id = image_path + BASE_IMAGE_NEW

    if (self.resource_exists(azure_res[illegible]e_id)):
        logger.error("Base Image already exists. Hence exiting")
        return ""
    else:
        logger.info("Creating new Base Image " + BASE_IMAGE_NEW + " in region " + data['REGION'])
        return BASE_IMAGE_NEW



def run_command(self, command, logger=None, get_output=False):
    if get_output:
        if logger is None:
            return subprocess.Popen(command, shell=True, stdout=subprocess.PIPE).stdout.read()
        else:
            logger.info("------Executing command " + command + "------")
            process = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
            output, error = process.communicate()
            logger.info(output)
            if error:
                logger.error(error)
                return error
            return output
    else:
        if logger is None:
            return subprocess.call(command, shell=True)
        else:
            logger.info("------Executing command " + command + "------")
            process = subprocess.Popen(command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)
            while True:
                output = process.stdout.readline()
                error = process.stderr.readline()
                if output == '' and error == '' and process.poll() is not None:
                    break
                if output:
                    out = output.strip()
                    logger.info(out)
                if error:
                    err = error.strip()
```

```
                        logger.error(err)
                rc = process.poll()
                return rc




    def vmrestart(self,resource_group,VM_NAME,logger):
        try:
            logger.info("Restarting:"+VM_NAME)
            async_vm_restart = self.compute_client.virtual_machines.restart(resource_group, VM_NAME)
            async_vm_restart.wait()
        except Exception as e:
            logger.info("Error while restarting the VM: "+str(e))


    def vmstop(self,resource_group,VM_NAME,logger):
        try:
            async_vm_stop = self.compute_client.virtual_machines.power_off(resource_group, keyword)
            async_vm_stop.wait()
        except Exception as e:
            logger.info("Unable to poweroff the VM. ")



    def create_image_VHD(self,resource_group,data,image_name,VM_NAME,logger,create_image):

        # Deallocate
        try:
            logger.info("Starting with VM deallocation")
            async_vm_deallocate = self.compute_client.virtual_machines.deallocate(resource_group, VM_NAME)
            async_vm_deallocate.wait()
            logger.info("Done with VM deallocation")

            logger.info("Starting with VM Generalize")
            async_vm_generalize = self.compute_client.virtual_machines.generalize(resource_group, VM_NAME)
            logger.info("Done with VM Generalize")


        except Exception as e:
            logger.info("Error during VM deallocation and generalization"+ str(e))
            sys.exit(0)




        #create image
        if(create_image=="y"):
            try:
                parameters = {
                    "location": data['REGION'],
                    "properties": {
                                    "sourceVirtualMachine": {
                                        "id": "/subscriptions/" + data['AZURE_SUB'] + "/resourceGroups/" + resource_group +
"/providers/Microsoft.Compute/virtualMachines/"+VM_NAME
                                    }
                    }
                }

                logger.info("Starting with VM Image Creation")
                async_vm_image = self.compute_client.images.create_or_update(data['RGROUP'], image_name, parameters)
                logger.info("Done with VM Image Creation")
                return async_vm_image
            except Exception as e:
                logger.info("Error during VM Image Creation:"+ str(e))



    def updateJson(self,data,json_file,logger):
        try:
            with open(json_file, 'w') as f:
                f.write(json.dumps(data, indent=4))
        except Exception as e:
            logger.info("Error during Config json update"+ str(e))


    def deleteImage(self,data,resource_group,image_name,logger):

        try:
            while(self.compute_client.images.get(resource_group, image_name).name==image_name):
                self.compute_client.images.delete(resource_group, image_name)
                time.sleep(5)
        except Exception as e:
            logger.info("Exception while deleting Images:"+str(e.message))




    def createSnapshot(self,resource_group,diskname,snapshot,logger):

        try:
            #resource_id = "/subscriptions/" + data['AZURE_SUB'] + "/resourceGroups/QA/providers/Microsoft.Compute/snapshots/" +
source
            #create_param = "\"create_option\": DiskCreateOption.empty,\n source_resource_id:" + resource_id

            managed_disk = self.compute_client.disks.get(resource_group, diskname)
            async_snapshot_creation = self.compute_client.snapshots.create_or_update(
                "QA-regr_snapshots",
                snapshot,
                {
                    'location': 'westus',
                    'creation_data': {
                        'create_option': 'Copy',
                        'source_uri': managed_disk.id
                    }
                }
            )
            return async_snapshot_creation.result()
        except Exception as e:
            logger.info("Exception while d[illegible]ing snapshot:"+str(e.message))


    def deletesnapshot(self,resource_group,snapshot,logger):
        try:
            while(self.compute_client.snapshots.get(resource_group, snapshot).name==snapshot):
                self.compute_client.snapshots.delete(resource_group, snapshot)
                time.sleep(5)
        except Exception as e:
            logger.info("Exception while deleting snapshot:"+str(e.message))



    def searchSnapshot(self,resource_group,snapshot,logger):
        """
            :param snapshotName: Snapshot name to be searched
            :param resourceGroup: Resource Group under which the snapshot should exist
            :return: return snapshotName if it exists.
                     else the existing snapshot in the Resource Group.
                     Null if no snapshot exists in the  Resource Group.
            """


        try:
            logger.info("Searching for snapshot:"+snapshot)
            try:
                info = self.compute_client.snapshots.get(resource_group, snapshot)
                if(info is not None):
                    snapshotid = info.id
                    logger.info("Found snapshot: "+snapshotid)
                    return True,snapshotid
            except Exception as e:
                try:
                    logger.info("snapshot not found. Will try to return any existing snapshot")
                    snapshot_list = self.compute_client.snapshots.list_by_resource_group(resource_group)
                    snapshotid=""
                    if(snapshot_list is not None):
                        for existing_snapshot in snapshot_list:
                            snapshotid=existing_snapshot.id

                        return True,snapshotid
                    else:
                        return False,None
                except Exception as e:
                    logger.info("Exception getting list of snapshots: "+str(e))

        except Exception as e:
            logger.info(e)
            return False,None



    def deleteotherSnapshots(self,resource_group,snapshot, logger):
        try:
            snapshot_list = self.compute_client.snapshots.list_by_resource_group(resource_group)
            logger.info("Printing snapshot list in the given resource group")
            logger.info(snapshot_list)
            for existing_snapshot in snapshot_list:
                if (existing_snapshot.name!=snapshot):
                    logger.info("Deleting snapshot:")
                    logger.info(existing_snapshot.name)
                    self.compute_client.snapshots.delete(resource_group, existing_snapshot.name)
                    time.sleep(5)

        except Exception as e:
            logger.info("Exception while deleting snapshot:"+str(e.message))





    def deleteAllresources(self,data,resource_group,keyword,logger):

        try:
            async_vm_stop = self.compute_client.virtual_machines.power_off(resource_group, keyword)
            async_vm_stop.wait()
        except Exception as e:
            logger.info("Unable to poweroff the VM. Trying to delete it brute force")

        logger.info("Got access to resource client")
        logger.info("Checking for keyword:"+keyword)
        try:
            for item in self.resource_client.resources.list_by_resource_group(resource_group):
                if(keyword in item.name):
                    if(item.type=="Microsoft.Compute/virtualMachines"):
                        logger.info("Deleting virtualMachines:" + item.name)
                        vm_delete_poller = self.compute_client.virtual_machines.delete(resource_group,item.name)
                        vm_delete_poller.wait()

                    if (item.type == "Microsoft.Network/networkInterfaces"):
                        logger.info("Deleting networkInterfaces:"+item.name)
                        networkInterface_delete_poller = self.network_client.network_interfaces.delete(resource_group,item.name)
                        networkInterface_delete_poller.wait()

                    if (item.type == "Microsoft.Network/publicIPAddresses"):
                        logger.info("Deleting publicIPAddresses:"+item.name)
                        publicIP_delete_poller=self.network_client.public_ip_addresses.delete(resource_group,item.name)
                        publicIP_delete_poller.wait()

                    if (item.type == "Microsoft.Network/virtualNetworks"):
                        logger.info("Deleting virtualNetworks:"+item.name)
                        VirtualNetwork_delete_poller=self.network_client.virtual_networks.delete(resource_group,item.name)
                        VirtualNetwork_delete_poller.wait()

                    if (item.type == "Microsoft.Compute/disks"):
                        logger.info("Deleting disks:" + item.name)
                        #self.compute_client.disks.delete(data['RGROUP'],item.name)
        except Exception as e:
            logger.info("Exception while deleting resource:"+str(e)+" Please delete it manually")




    def createNetworkSecurityGroup(self,data,resource_group,VM_NAME,port,priority,logger):

        params_create = {
            'location': data['REGION'],
            'security_rules':[{
                'name':VM_NAME+port,
                'source_port_range':"*",
                'source_address_prefix':"*",
                'destination_port_range':port,
                'destination_address_prefix': "*",
                'priority':priority,
                'protocol':'*',
                'access':'allow',
                'direction':'inbound'
            }]
        }
        logger.info("Done with creation of params for NSG update")

        nsg_poller = self.network_client.network_security_groups.create_or_update(
            resource_group,
            VM_NAME + "-nsg",
            params_create
        )
        nsg_poller.wait()
        logger.info("Done with create NSG with the given port "+port+" to open")



    def addsecurityRule(self,data,resource_group,VM_NAME,port,priority,logger):
        params_securtiy_rule={
                'name':VM_NAME+port,
                'source_port_range':"*",
                'source_address_prefix':"*"[illegible]
                'destination_port_range':p[illegible]
                'destination_address_prefix': "*",
                'priority':priority,
                'protocol':'*',
                'access':'allow',
                'direction':'inbound'
            }
        nsr_poller = self.network_client.security_rules.create_or_update(resource_group,
                                                                          VM_NAME + "-nsg",
                                                                          VM_NAME + "-nsr",
                                                                          params_securtiy_rule
                                                                          )
        nsr_poller.wait()
        logger.info("Done with adding security rule")




    def updatesubnet(self,VM_NAME,data,resource_group,logger):
        params_subnetInfo=self.network_client.subnets.get(resource_group,
                                                          VM_NAME + "-Vnet",
                                                          VM_NAME + "-subnet")
        params_nsgInfo = self.network_client.network_security_groups.get(resource_group,VM_NAME + "-nsg")
        params_subnetUpdate = {
            'address_prefix':params_subnetInfo.address_prefix,
            'network_security_group': {
                'id':params_nsgInfo.id,
                'location':params_nsgInfo.location,
                'name': VM_NAME + "-nsg"
            }
        }
        AzureOperationPoller = self.network_client.subnets.create_or_update(resource_group,
                                                                            VM_NAME + "-Vnet",
                                                                            VM_NAME + "-subnet",
                                                                            params_subnetUpdate
                                                                            )
        AzureOperationPoller.wait()
        logger.info("Done with updating the NSG on the subnet")
    def getmarketplace_Image(self,data,logger):
            logger.info("Retrieving MarketPlace image")
            marketplace_Image= self.compute_client.virtual_machine_images.get(data['REGION'],"marklogic","marklogic-9-
byol","ml9061_centos_byol","1.0.0")
            logger.info(marketplace_Image)
            return marketplace_Image

    def validateParam(self,template_file, resource_group,data,params,BASE_URL_PARAM,logger):
        try:
            logger.info("\n")
            logger.info("\t Validating parameters:")
            logger.info(params)

            default_params=script_dir+"/defaultTemplateParameters.json"
            logger.info("Default params:"+default_params)

            for key,value in params.items():
                logger.info("Validating Params:")
                param=key+"="+value
                logger.info(param)

                parameters = default_params+" "+param+ " "+BASE_URL_PARAM

                logger.info(parameters)
                params={
                    "template" :template_file,
                    "parameters" : parameters,
                    "mode":'Incremental'
                }

                result = self.resource_client.deployments.validate(resource_group,"validate",params)
                logger.info("Validation Result:")
                logger.info(result)
            logger.info("Done with Validation \n")
        except Exception as e:
            logger.info("Exception occured while validating params:" + str(e))


    def deployTemplate(self,solution_template,data,resource_group,params_toValidate,BASE_URL_PARAM,logger):

        try:
            logger.info("\n Starting deployment")
            default_params_file=script_dir+"/../resources/defaultTemplateParameters.json"
            logger.info("Default paramsFile:"+default_params_file)
            cmd  = "az group deployment validate --resource-group "+resource_group+" --template-file "+solution_template+" --
parameters "+default_params_file+" --parameters "+params_toValidate+" "+BASE_URL_PARAM
```

```
      validate_output = self.run_command(cmd, logger, "true")
      validate_json = json.loads(validate_output)
      if(validate_json['properties']['provisioningState']=="Succeeded"):
      #if ("Succeeded"):
        logger.info("Validation :"+validate_json['properties']['provisioningState']+".Hence deploying the template")
        cmd = "az group deployment create --resource-group " + resource_group + " --template-file " + solution_template + " --
parameters " + default_params_file + " --parameters " + params_toValidate + " " + BASE_URL_PARAM
        create_output = self.run_command(cmd, logger, "true")
        create_output=create_output.decode('utf-8')
        create_output = create_output.replace("\x1b[0m", "")
        contents = script_dir + "/output.txt"
        rm_cmd = "rm -rf " + contents
        rm_output = self.run_command(rm_cmd, logger, "true")

        with open(contents,"w") as file:
          file.write(create_output)
        create_output = self.read_json_file(logger, contents)
        logger.info(create_output['properties']['provisioningState'])


      if(create_output['properties']['provisioningState']=="Succeeded"):
          logger.info("Deployement "+create_output['properties']['provisioningState']+" for Params:"+params_toValidate)
          keyword = params_toValidate.split()[0].split("=")[1]
          logger.info("Deleting resourced related to :"+keyword)
          #self.deleteAllresources(data, keyword, logger)
      else :
          logger.info("Due to failure, for debugging, resources are not deleted. Please delete them manually from Azure
Portal ")
          sys.exit("Deployement Failed for Params:"+params_toValidate)
    except Exception as e:
      logger.info("Exception occured while deploy params:" + str(e))


  def  createResources(self,data,logger):
     self.getCredentials(data)
     self.resource_client = ResourceManagementClient(self.credentials, data['AZURE_SUB'])
     self.compute_client = ComputeManagementClient(self.credentials, data['AZURE_SUB'])
     self.network_client = NetworkManagementClient(self.credentials, data['AZURE_SUB'])
     self.storage_client = StorageManagementClient(self.credentials, data['AZURE_SUB'])

     self.printresources()


  def printresources(self):
    print(self.resource_client)
    print(self.compute_client)
    print(self.network_client)
    def createandAttachDisk(self,resource_group,resource_group_snapshot,source,attach_to_vm,data,logger):
        logger.info("Creating Disk for VM:"+attach_to_vm+ " and using source:"+source)
        lun =0
        try:
            if(source==""):
                params = {
                    "location": data['REGION'],
                    "os_type": "linux",
                    "disk_size_gb": "30",
                    "creation_data": {
                        'create_option': DiskCreateOption.empty
                    }
                }
                lun =3
                diskName= attach_to_vm + "qaDisk"
            else:
                resource_id =
  "/subscriptions/"+data['AZURE_SUB']+"/resourceGroups/"+resource_group_snapshot+"/providers/Microsoft.Compute/snapshots/"+source
                params = {
                    "location": data['REGION'],
                    "os_type": "linux",
                    "disk_size_gb": "100",
                    "creation_data": {
                        'create_option': DiskCreateOption.copy,
                        'source_resource_id':resource_id
                    }
                }
                diskName= attach_to_vm + "snapshotqaDisk"


            logger.info(params)

            data_disk = self.compute_client.disks.create_or_update(resource_group,diskName,params)
            data_disk.wait()

            logger.info(data_disk.result())
            logger.info(data_disk.status())

            VM_DETAILS = self.compute_client.virtual_machines.get(
                resource_group,
                attach_to_vm
                    )

            logger.info("Done retreiving VM_details")

            VM_DETAILS.storage_profile.data_disks.append({
                'lun': lun,
                'name': diskName,
                'create_option': DiskCreateOption.attach,
                'managed_disk': {
                    'id': data_disk.result().id
                }
            })
            logger.info("Done appending VM_details to storage profile")

            async_disk_attach = self.compute_client.virtual_machines.create_or_update(
                resource_group,
                VM_DETAILS.name,
                VM_DETAILS
            )
            logger.info("Done attaching managed disk VM_details ")

            async_disk_attach.wait()
        except Exception as e:
            logger.info("Exception while creating and attaching disk:"+str(e))



    def vmdiskdettach(self,resource_group,VM_NAME,keyword,logger):
        virtual_machine = self.compute_client.virtual_machines.get(
            resource_group,
            VM_NAME
        )

        data_disks = virtual_machine.storage_profile.data_disks
        data_disks[:] = [disk for disk in data_disks if disk.name != VM_NAME+keyword]
        async_vm_update = self.compute_client.virtual_machines.create_or_update(
            resource_group,
            VM_NAME,
            virtual_machine
        )

        async_vm_update.wait()







    def copyFilesandSetup(self,resource_group,VM_NAME,files_list,data,logger):
        logger.info("Copying files and setting up VM:"+VM_NAME)
        publicip = self.getpublic_ip(resource_group,VM_NAME)
        logger.info("PublicIP:")
        logger.info(publicip.ip_address)
        try:
            isVMaccessible=self.checkVMAccessibility("builder-temp",publicip.ip_address,data,logger)
            if(isVMaccessible=="true"):
                files = " "
                for x in files_list:
                    files = files+" "+x

                logger.info("Copying files:"+files)

                cmd = "scp -i " + script_dir+"/../resources/"+data['SSH_PRIV'] + " " + files + " builder-temp@" + publicip.ip_address
+ ":/tmp"
                output = self.run_command(cmd, logger, "true")
                logger.info(output.decode('utf-8'))
                cmd = "ssh -i " + script_dir+"/../resources/"+data['SSH_PRIV'] + " -oStrictHostKeyChecking=no builder-temp@" +
publicip.ip_address + " \" cd /tmp/; sudo /tmp/vm-setup.sh\" "
                output = self.run_command(cmd, logger, "true")
                logger.info(output.decode('utf-8'))
                isVMaccessible = self.checkVMAccessibility("builder", publicip.ip_address, data, logger)

            else:
                logger.info("VM is inaccessible.Hence quitting the run")
                sys.exit(0)

        except Exception as e:
            logger.info("Exception while setting up the VM")


    def checkVMAccessibility(self,username,ip_address,data,logger):
        try:
            logger.info("verifying if VM is accessible")
            privateKey =script_dir+"/../resources/"+data['SSH_PRIV']
            logger.info("Using Private Key:"+privateKey)
            cmd = "ssh -i " + privateKey + " -oStrictHostKeyChecking=no "+username+"@" + ip_address + " 'whoami'  "

            count = 0
            while (count < 3):
                output = self.run_command(cmd, logger, "true")
                logger.info(output.decode('utf-8'))
                output = output.decode('utf-8')
                if (username in output):
                    logger.info("VM is up and running and we can login")
                    return "true"
                else:
                    logger.info("Unable to  login into VM for time:")
                    logger.info(count)
                    count = count + 1
                    if (count > 3):
                        logger.info("Issue with VM. Hence exiting. Please check VM with IP:"+ip_address)
                    return "false"
        except Exception as e:
            logger.info("Exception occured while verifying VM" + str(e))
            sys.exit(0)

    def createResourceGroup(self,resource_group,data,logger):
        try:
            resource_group_params = {'location': data['REGION']}
            self.resource_client.resource_groups.create_or_update(resource_group,resource_group_params)
            return resource_group
        except Exception as e:
            logger.info("Error creating re[illegible]ce group: "+resource_group+". Hence switching to default Rgroup")
            logger.info("Exception:"+str(e))
            return "pbokka"


    def http_request_send(self,url, method, data, headers):
        r = None
        if method == "POST":
            r = requests.post(
                url,
                json=data,
                headers={'Content-type': 'application/json'} if not headers else headers
            )
        elif method == "PUT":
            r = requests.put(
                url,
                json=data,
                headers={'Content-type': 'application/json'} if not headers else headers
            )
        elif method == "DELETE":
            r = requests.delete(
                url,
                headers={'Accept': 'application/json'} if not headers else headers
            )
        elif method == "GET":
            r = requests.get(
                url,
                auth=HTTPDigestAuth('admin', 'admin')
            )
        return r


    def getstorageAccount(self,stackname,logger):
        logger.info("ResourceGp:"+stackname+"-rg")
        storage_account = ""
        for item in self.storage_client.storage_accounts.list_by_resource_group(stackname+"-rg"):
            return item.name
    def getstorageKey(self,stackname,storageAccount):
        logging.info("Getting storage Key for :"+storageAccount)
        storage_keys = self.storage_client.storage_accounts.list_keys(stackname+"-rg", storageAccount)
        storage_keys = {v.key_name: v.value for v in storage_keys.keys}
        return storage_keys['key1']



    def checkTableService(self,storageAccount,storageAccount_key,logger):
        logger.info("Checking for logs in the Table storage")
        table_service = TableService(account_name=storageAccount, account_key=storageAccount_key)
        if(table_service.exists("MarkLogicLogsVer2v0")):
            logger.info("The MarkLogicLogsVer2v0 table exists. Hence test has passed")
        else:
            sys.exit("The MarkLogicLogsVer2v0 table does not exist. Hence test has failed")



    def checkBlobStorage(self,storageAccount,storageAccount_key,logger):
        logger.info("Checking for logs in the blob storage")
        blob_service = BlockBlobService(account_name=storageAccount,
                                        account_key=storageAccount_key)
        containers = blob_service.list_containers()
        container_found = "false"
        for c in containers:
            if (c.name == "azure-dhs"):
                count=0
                #logger.info("Contents of Container:")
                blobs = blob_service.list_blobs(c.name)
                for b in blobs:
                    #logger.info(b.name)
                    count=count+1
                if(count>0):
                    logger.info("Count:"+str(count))
                    container_found = "true"
                    logger.info("Container azure-dhs is found with contents. Hence test has passed")

        if (container_found != "true"):
            sys.exit("azure-dhs containter has no contents. This means the log forwarding function is not working")



    def copyMarkLogicBinaries(self, ml_rpm, ml_conv_rpm,logger):
        """
        :param ml_rpm:  Marklogic user input rpm . Will be changed to nightly once pushed in Jenkins
        :param ml_conv_rpm: Marklogic Converter user input rpm . Will be changed to nightly once pushed in Jenkins
        :return: None
        This definition will copy the converters into staging location
        """

        try:
            logger.info("Copying Marklogic rpm and converters rpm to staging location")
            cmd = "rm -rf " + script_dir + "/../../resources/ml-azure/regr_scripts/MarkLogic*;"
            cmd = "cp -rf " + ml_rpm + " " + script_dir + "/../../resources/ml-azure/regr_scripts/setupFiles/MarkLogic.rpm;"
            cmd = cmd + "cp -rf " + ml_conv_rpm + " " + script_dir + "/../../resources/ml-
azure/regr_scripts/setupFiles/MarkLogicConverters.rpm;"
            logger.info(cmd)
            output = self.run_command(cmd, logger, "true")
            logger.info(output.decode('utf-8'))
        except Exception as e:
            logger.info("Exception while copying rpm files. Hence exiting ")

    def createEmptyHost(self, isMaster,hostname,logger):
        """
        Creates  a VM using Terraform with datadisk as Empty
        If isMaster is true then sets up svn config in the host else skips it
        :return: Hostname
        """
        logger.info("Code here for empty and Master is: "+str(isMaster))
        tf_json_file = script_dir + "/../resources/ml-azure/terraform/NewSnapshot/createVM.auto.tfvars.json"
        tf_data = self.read_json_file(logger, tf_json_file)
        tf_data['vm_name'] = hostname
        tf_data['setupFiles_location'] = script_dir + "/../resources/ml-azure/regr_scripts/setupFiles/"
        tf_data['user'] = getpass.getuser()
        tf_data['mount_script'] = "mountNew_SVNdownload.sh"
        tf_data['create_option'] = "Empty"
        tf_data['isMaster'] = str(isMaster)
        self.updateJson(tf_data, tf_json_file, logger)
        try:
            cmd = "cd " + script_dir + "/../resources/ml-azure/terraform/NewSnapshot;rm -rf terraform.tfstate*; sh invoketerraform.sh;
cd " + script_dir
            logger.info(cmd)
            output = self.run_command(cmd, logger, "true")
            logger.info(output.decode('utf-8'))
        except Exception as e:
            logger.info("Exception while triggering terraform")



    def createsnapshotHost(self, snapshotName, data,hostname,logger):
        """
            Creates  a VM using Terraform with datadisk as managed disk from given snapshot
            If isMaster is true then sets up svn config in the host else skips it
            :return: Hostname
         """
        logger.info("Code her for snapshot and master")

        tf_json_file = script_dir + "/../resources/ml-azure/terraform/ExistingSnapshot/createVM.auto.tfvars.json"
        tf_data = self.read_json_file(logger, tf_json_file)
        tf_data['vm_name'] = hostname
        tf_data['setupFiles_location'] = script_dir + "/../resources/ml-azure/regr_scripts/setupFiles/"
        tf_data['user'] = getpass.getuser()
        tf_data['mount_script']="mount_existingsnapshot.sh"
        tf_data['create_option']="copy"
        tf_data['isMaster']="true"
        self.updateJson(tf_data, tf_json_file, logger)
        try:
            cmd = "cd " + script_dir + "/../resources/ml-azure/terraform/ExistingSnapshot; sh invoketerraform.sh; cd " + script_dir
            logger.info(cmd)
            output = self.run_command(cmd, logger, "true")
            logger.info(output.decode('utf-8'))
        except Exception as e:
            logger.info("Exception while triggering terraform")




    def createsnapshot(self, snapshot_new, VM_NAME, resource_group, data,logger):

        try:
            logger.info("Creating snapshots")
            publicip = self.getpublic_ip(resource_group, VM_NAME)
            try:
                cmd = "ssh -i " + script_dir + "/../resources/" + data[
                    'SSH_PRIV'] + " -oStrictHostKeyChecking=no builder@" + publicip.ip_address + " /tmp/vm1-cleanup.sh "
                output = utils.run_command(cmd, logger, "true")
                logger.info(output.decode('utf-8'))
            except Exception as e:
                logger.info("Exception while running cleanup scripts:" + str(e))

            # utils.vmdiskdettach(data, VM_NAME, "snapshotqaDisk", logger)
            diskname = VM_NAME + "snapshotqaDisk"
            logger.info("Creating snapshot from Disk:" + diskname)

            # below hardcoding "QA-regr_snapshots" is intentional. Don't remove it
            snapshot = utils.createSnapshot(resource_group, diskname, snapshot_new, logger)
            logger.info("Created snapshot:")
            logger.info(snapshot)
        except Exception as e:
            logger.info("Exception while updating snapshot")
```