#!/bin/sh

[ -e ../managed\_eni.zip ] && rm ../managed\_eni.zip

[ -e ../version.txt ] && rm ../version.txt

echo `date` > ../version.txt

cp custom\_resource\_base.zip ../managed\_eni.zip

cd ..

zip -g managed\_eni.zip managedeni.py utils.py version.txt

rm version.txt

chmod 777 managed\_eni.zip

#!/bin/sh

[ -e ../node\_manager.zip ] && rm ../node\_manager.zip

[ -e ../version.txt ] && rm ../version.txt

echo `date` > ../version.txt

cd ..

zip node\_manager.zip nodemanager.py utils.py version.txt

rm version.txt

chmod 777 node\_manager.zip

# Copyright 2002-2018 MarkLogic Corporation. All Rights Reserved.

import boto3

import logging

import hashlib

import json

from botocore.exceptions import ClientError

import cfn\_resource

import time

from utils import get\_network\_interface\_by\_id

from utils import cfn\_success\_response

from utils import cfn\_failure\_response

log = logging.getLogger()

log.setLevel(logging.INFO)

# global variables

handler = cfn\_resource.Resource()

ec2\_client = boto3.client('ec2')

ec2\_resource = boto3.resource('ec2')

def eni\_wait\_for\_creation(eni\_id):

max\_retry = 10

retries = 0

sleep\_interval = 10

while True and retries < max\_retry:

# resource is not ready yet, not visible to 'ec2\_resource.NetworkInterface(id=eni\_id)'

eni\_info = get\_network\_interface\_by\_id(eni\_id)

if eni\_info:

status = eni\_info["Status"]

if status == "available":

return eni\_info

else:

log.warning("Network interface %s in unexpected status: %s" % (eni\_id, status))

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Network interface %s not found" % eni\_id)

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Waiting for network interface %s creation timed out" % eni\_id)

def eni\_wait\_for\_detachment(eni\_id):

max\_retry = 10

retries = 0

sleep\_interval = 10

while True and retries < max\_retry:

eni\_info = ec2\_resource.NetworkInterface(id=eni\_id)

if eni\_info:

if not eni\_info.attachment:

break

status = eni\_info.attachment["Status"]

if status == "detached":

break

elif status == "attached" or status == "detaching":

time.sleep(sleep\_interval)

retries += 1

continue

else:

log.warning("Network interface %s in unexpected attachment status: %s" % (eni\_id, status))

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Network interface %s not found" % eni\_id)

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Waiting for network interface %s detachment timed out" % eni\_id)

def eni\_exist(subnet\_id, security\_group\_id, tag):

response = ec2\_client.describe\_network\_interfaces(

Filters=[

{

"Name": "tag:cluster-eni-id",

"Values": [tag]

},

{

"Name": "subnet-id",

"Values": [subnet\_id]

},

{

"Name": "group-id",

"Values": [security\_group\_id]

}

]

)

if len(response["NetworkInterfaces"]) > 0:

return response["NetworkInterfaces"][0]

else:

return None

def create\_eni(subnet\_id, security\_group\_id, tag):

"""

Create ENI and populate the description with the tag content.

This is a temporarily work around because of AWS SDK bug.

:param subnet\_id:

:param tag:

:return:

"""

eni\_info = eni\_exist(subnet\_id, security\_group\_id, tag)

if eni\_info:

return eni\_info

try:

eni = ec2\_client.create\_network\_interface(

Groups=[security\_group\_id],

SubnetId=subnet\_id,

)

eni\_id = eni['NetworkInterface']['NetworkInterfaceId']

log.info("Creating network interface %s in subnet %s with tag %s" % (

eni\_id,

subnet\_id,

tag

))

except ClientError as e:

reason = "network interface %s in subnet %s with tag %s" % (

eni\_id,

subnet\_id,

tag

)

log.exception(reason)

time.sleep(5)

log.info("Waiting for network interface %s creation finish" % eni\_id)

return eni\_wait\_for\_creation(eni\_id)

def detach\_eni(eni\_id, attachment\_id):

try:

response = ec2\_client.detach\_network\_interface(

AttachmentId=attachment\_id,

Force=True

)

log.info("Detaching network interface %s" % eni\_id)

except ClientError as e:

reason = "Failed to detach %s" % attachment\_id

log.exception(reason)

time.sleep(5)

return False

log.info("Waiting for network interface %s detachment finish" % eni\_id)

eni\_wait\_for\_detachment(eni\_id)

return True

def eni\_assign\_tag(eni\_id, tag):

max\_retry = 10

retries = 0

sleep\_interval = 5

while True:

try:

eni = ec2\_resource.NetworkInterface(id=eni\_id)

tag = eni.create\_tags(

Tags=[

{

'Key': 'cluster-eni-id',

'Value': tag

}

]

)

except ClientError as e:

retries += 1

if (retries >= max\_retry):

raise e

log.exception(e)

time.sleep(sleep\_interval)

continue

break

def on\_create\_handler(event, context):

log.info("Handle resource create event %s" % json.dumps(event, indent=2))

# get parameters passed in

props = event["ResourceProperties"]

nodes\_per\_zone = int(props["NodesPerZone"])

zone\_count = int(props["NumberOfZones"])

parent\_stack\_name = props["ParentStackName"]

parent\_stack\_id = props["ParentStackId"]

subnets = props["Subnets"]

security\_group\_id = props["SecurityGroup"]

# prepare ENI meta information

id\_hash = hashlib.md5(parent\_stack\_id.encode()).hexdigest()

eni\_tag\_prefix = parent\_stack\_name + "-" + id\_hash + "\_"

addresses = []

# craete ENIs

for i in range(0,zone\_count):

for j in range(0,nodes\_per\_zone):

eni\_idx = i \* nodes\_per\_zone + j

tag = eni\_tag\_prefix + str(eni\_idx)

eni\_info = create\_eni(subnets[i], security\_group\_id, tag)

if not eni\_info:

reason = "Failed to create network interface with tag %s" % tag

log.warning(reason)

continue

eni\_id = eni\_info["NetworkInterfaceId"]

if "PrivateDnsName" in eni\_info:

eni\_address = eni\_info["PrivateDnsName"]

else:

eni\_address = eni\_info["PrivateIpAddress"]

eni\_assign\_tag(eni\_id=eni\_id, tag=tag)

addresses.append(eni\_address)

return cfn\_success\_response(event,data={

"Addresses": ",".join(addresses)

})

@handler.create

def on\_create(event, context):

try:

return on\_create\_handler(event, context)

except Exception as e:

log.exception(e)

return cfn\_failure\_response(event, str(e))

@handler.update

def on\_update(event, context):

log.info("Handle resource update event %s" % json.dumps(event, indent=2))

# get parameters passed in

props = event["ResourceProperties"]

nodes\_per\_zone = int(props["NodesPerZone"])

zone\_count = int(props["NumberOfZones"])

parent\_stack\_name = props["ParentStackName"]

parent\_stack\_id = props["ParentStackId"]

subnets = props["Subnets"]

security\_group\_id = props["SecurityGroup"]

# old properties

old\_props = event["OldResourceProperties"]

old\_nodes\_per\_zone = int(old\_props["NodesPerZone"])

old\_zone\_count = int(old\_props["NumberOfZones"])

# validate diff and handle special case

if old\_zone\_count != zone\_count:

reason = "Updating number of zones is not supported"

log.error(reason)

return cfn\_failure\_response(event,reason)

if old\_nodes\_per\_zone > nodes\_per\_zone and nodes\_per\_zone != 0:

reason = "Scaling down the number of nodes per zone by updating the stack is not recommended. " \

"Please manually remove unused network interface."

log.warning(reason)

return cfn\_failure\_response(event, reason)

if nodes\_per\_zone == 0:

log.info("Hibernating the cluster, retain network interfaces")

nodes\_per\_zone = old\_nodes\_per\_zone

# prepare ENI meta information

id\_hash = hashlib.md5(parent\_stack\_id.encode()).hexdigest()

eni\_tag\_prefix = parent\_stack\_name + "-" + id\_hash + "\_"

dns = []

eni\_idx = 0

for i in range(zone\_count):

for j in range(nodes\_per\_zone):

tag = eni\_tag\_prefix + str(eni\_idx)

eni\_idx += 1

eni\_info = create\_eni(subnets[i], security\_group\_id, tag)

if not eni\_info:

reason = "Failed to create network interface with tag %s" % tag

log.warning(reason)

continue

eni\_id = eni\_info["NetworkInterfaceId"]

eni\_dns = eni\_info["PrivateDnsName"]

eni\_assign\_tag(eni\_id=eni\_id, tag=tag)

dns.append(eni\_dns)

return cfn\_success\_response(event,reuse\_physical\_id=True,data={

"Addresses": ",".join(dns)

})

@handler.delete

def on\_delete(event, context):

log.info("Handle resource delete event %s " % json.dumps(event, indent=2))

# get parameters passed in

props = event["ResourceProperties"]

nodes\_per\_zone = int(props["NodesPerZone"])

zone\_count = int(props["NumberOfZones"])

parent\_stack\_name = props["ParentStackName"]

parent\_stack\_id = props["ParentStackId"]

# prepare ENI meta information

id\_hash = hashlib.md5(parent\_stack\_id.encode()).hexdigest()

eni\_tag\_prefix = parent\_stack\_name + "-" + id\_hash + "\_"

# delete ENIs

eni\_idx = 0

while True:

tag = eni\_tag\_prefix + str(eni\_idx)

log.info("Querying EC2 for ENI with tag %s" % tag)

# query

response = None

try:

response = ec2\_client.describe\_network\_interfaces(

Filters=[{

"Name": "tag:cluster-eni-id",

"Values": [tag]

}]

)

except ClientError as e:

reason = "Failed to describe network interface with tag %s" % tag

log.exception(reason)

time.sleep(5)

eni\_idx += 1

continue

if not response["NetworkInterfaces"]:

break

for eni\_info in response["NetworkInterfaces"]:

eni\_id = eni\_info["NetworkInterfaceId"]

log.info("Found network interface %s " % eni\_id)

# detach

if "Attachment" in eni\_info and (

eni\_info["Attachment"]["Status"] == "attached" or

eni\_info["Attachment"]["Status"] == "attaching"

):

attachment\_id = eni\_info["Attachment"]["AttachmentId"]

if not detach\_eni(eni\_id, attachment\_id):

reason = "Failed to detach network interface %s" % eni\_id

log.error(reason)

try:

ec2\_client.delete\_network\_interface(

NetworkInterfaceId=eni\_id

)

log.info("Deleting network interface %s" % eni\_id)

except ClientError as e:

reason = "Failed to delete network interface %s" % eni\_id

log.exception(reason)

return cfn\_failure\_response(event, reason)

eni\_idx += 1

return cfn\_success\_response(event)

# Copyright 2002-2018 MarkLogic Corporation. All Rights Reserved.

import boto3

import logging

import hashlib

import json

from botocore.exceptions import ClientError

import cfn\_resource

import time

from utils import get\_network\_interface\_by\_id

from utils import cfn\_success\_response

from utils import cfn\_failure\_response

log = logging.getLogger()

log.setLevel(logging.INFO)

# global variables

handler = cfn\_resource.Resource()

ec2\_client = boto3.client('ec2')

ec2\_resource = boto3.resource('ec2')

def eni\_wait\_for\_creation(eni\_id):

max\_retry = 10

retries = 0

sleep\_interval = 10

while True and retries < max\_retry:

# resource is not ready yet, not visible to 'ec2\_resource.NetworkInterface(id=eni\_id)'

eni\_info = get\_network\_interface\_by\_id(eni\_id)

if eni\_info:

status = eni\_info["Status"]

if status == "available":

return eni\_info

else:

log.warning("Network interface %s in unexpected status: %s" % (eni\_id, status))

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Network interface %s not found" % eni\_id)

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Waiting for network interface %s creation timed out" % eni\_id)

def eni\_wait\_for\_detachment(eni\_id):

max\_retry = 10

retries = 0

sleep\_interval = 10

while True and retries < max\_retry:

eni\_info = ec2\_resource.NetworkInterface(id=eni\_id)

if eni\_info:

if not eni\_info.attachment:

break

status = eni\_info.attachment["Status"]

if status == "detached":

break

elif status == "attached" or status == "detaching":

time.sleep(sleep\_interval)

retries += 1

continue

else:

log.warning("Network interface %s in unexpected attachment status: %s" % (eni\_id, status))

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Network interface %s not found" % eni\_id)

time.sleep(sleep\_interval)

retries += 1

else:

log.warning("Waiting for network interface %s detachment timed out" % eni\_id)

def eni\_exist(subnet\_id, security\_group\_id, tag):

response = ec2\_client.describe\_network\_interfaces(

Filters=[

{

"Name": "tag:cluster-eni-id",

"Values": [tag]

},

{

"Name": "subnet-id",

"Values": [subnet\_id]

},

{

"Name": "group-id",

"Values": [security\_group\_id]

}

]

)

if len(response["NetworkInterfaces"]) > 0:

return response["NetworkInterfaces"][0]

else:

return None

def create\_eni(subnet\_id, security\_group\_id, tag):

"""

Create ENI and populate the description with the tag content.

This is a temporarily work around because of AWS SDK bug.

:param subnet\_id:

:param tag:

:return:

"""

eni\_info = eni\_exist(subnet\_id, security\_group\_id, tag)

if eni\_info:

return eni\_info

try:

eni = ec2\_client.create\_network\_interface(

Groups=[security\_group\_id],

SubnetId=subnet\_id,

)

eni\_id = eni['NetworkInterface']['NetworkInterfaceId']

log.info("Creating network interface %s in subnet %s with tag %s" % (

eni\_id,

subnet\_id,

tag

))

except ClientError as e:

reason = "network interface %s in subnet %s with tag %s" % (

eni\_id,

subnet\_id,

tag

)

log.exception(reason)

time.sleep(5)

log.info("Waiting for network interface %s creation finish" % eni\_id)

return eni\_wait\_for\_creation(eni\_id)

def detach\_eni(eni\_id, attachment\_id):

try:

response = ec2\_client.detach\_network\_interface(

AttachmentId=attachment\_id,

Force=True

)

log.info("Detaching network interface %s" % eni\_id)

except ClientError as e:

reason = "Failed to detach %s" % attachment\_id

log.exception(reason)

time.sleep(5)

return False

log.info("Waiting for network interface %s detachment finish" % eni\_id)

eni\_wait\_for\_detachment(eni\_id)

return True

def eni\_assign\_tag(eni\_id, tag):

max\_retry = 10

retries = 0

sleep\_interval = 5

while True:

try:

eni = ec2\_resource.NetworkInterface(id=eni\_id)

tag = eni.create\_tags(

Tags=[

{

'Key': 'cluster-eni-id',

'Value': tag

}

]

)

except ClientError as e:

retries += 1

if (retries >= max\_retry):

raise e

log.exception(e)

time.sleep(sleep\_interval)

continue

break

def on\_create\_handler(event, context):

log.info("Handle resource create event %s" % json.dumps(event, indent=2))

# get parameters passed in

props = event["ResourceProperties"]

nodes\_per\_zone = int(props["NodesPerZone"])

zone\_count = int(props["NumberOfZones"])

parent\_stack\_name = props["ParentStackName"]

parent\_stack\_id = props["ParentStackId"]

subnets = props["Subnets"]

security\_group\_id = props["SecurityGroup"]

# prepare ENI meta information

id\_hash = hashlib.md5(parent\_stack\_id.encode()).hexdigest()

eni\_tag\_prefix = parent\_stack\_name + "-" + id\_hash + "\_"

addresses = []

# craete ENIs

for i in range(0,zone\_count):

for j in range(0,nodes\_per\_zone):

eni\_idx = i \* nodes\_per\_zone + j

tag = eni\_tag\_prefix + str(eni\_idx)

eni\_info = create\_eni(subnets[i], security\_group\_id, tag)

if not eni\_info:

reason = "Failed to create network interface with tag %s" % tag

log.warning(reason)

continue

eni\_id = eni\_info["NetworkInterfaceId"]

if "PrivateDnsName" in eni\_info:

eni\_address = eni\_info["PrivateDnsName"]

else:

eni\_address = eni\_info["PrivateIpAddress"]

eni\_assign\_tag(eni\_id=eni\_id, tag=tag)

addresses.append(eni\_address)

return cfn\_success\_response(event,data={

"Addresses": ",".join(addresses)

})

@handler.create

def on\_create(event, context):

try:

return on\_create\_handler(event, context)

except Exception as e:

log.exception(e)

return cfn\_failure\_response(event, str(e))

@handler.update

def on\_update(event, context):

log.info("Handle resource update event %s" % json.dumps(event, indent=2))

# get parameters passed in

props = event["ResourceProperties"]

nodes\_per\_zone = int(props["NodesPerZone"])

zone\_count = int(props["NumberOfZones"])

parent\_stack\_name = props["ParentStackName"]

parent\_stack\_id = props["ParentStackId"]

subnets = props["Subnets"]

security\_group\_id = props["SecurityGroup"]

# old properties

old\_props = event["OldResourceProperties"]

old\_nodes\_per\_zone = int(old\_props["NodesPerZone"])

old\_zone\_count = int(old\_props["NumberOfZones"])

# validate diff and handle special case

if old\_zone\_count != zone\_count:

reason = "Updating number of zones is not supported"

log.error(reason)

return cfn\_failure\_response(event,reason)

if old\_nodes\_per\_zone > nodes\_per\_zone and nodes\_per\_zone != 0:

reason = "Scaling down the number of nodes per zone by updating the stack is not recommended. " \

"Please manually remove unused network interface."

log.warning(reason)

return cfn\_failure\_response(event, reason)

if nodes\_per\_zone == 0:

log.info("Hibernating the cluster, retain network interfaces")

nodes\_per\_zone = old\_nodes\_per\_zone

# prepare ENI meta information

id\_hash = hashlib.md5(parent\_stack\_id.encode()).hexdigest()

eni\_tag\_prefix = parent\_stack\_name + "-" + id\_hash + "\_"

dns = []

eni\_idx = 0

for i in range(zone\_count):

for j in range(nodes\_per\_zone):

tag = eni\_tag\_prefix + str(eni\_idx)

eni\_idx += 1

eni\_info = create\_eni(subnets[i], security\_group\_id, tag)

if not eni\_info:

reason = "Failed to create network interface with tag %s" % tag

log.warning(reason)

continue

eni\_id = eni\_info["NetworkInterfaceId"]

eni\_dns = eni\_info["PrivateDnsName"]

eni\_assign\_tag(eni\_id=eni\_id, tag=tag)

dns.append(eni\_dns)

return cfn\_success\_response(event,reuse\_physical\_id=True,data={

"Addresses": ",".join(dns)

})

@handler.delete

def on\_delete(event, context):

log.info("Handle resource delete event %s " % json.dumps(event, indent=2))

# get parameters passed in

props = event["ResourceProperties"]

nodes\_per\_zone = int(props["NodesPerZone"])

zone\_count = int(props["NumberOfZones"])

parent\_stack\_name = props["ParentStackName"]

parent\_stack\_id = props["ParentStackId"]

# prepare ENI meta information

id\_hash = hashlib.md5(parent\_stack\_id.encode()).hexdigest()

eni\_tag\_prefix = parent\_stack\_name + "-" + id\_hash + "\_"

# delete ENIs

eni\_idx = 0

while True:

tag = eni\_tag\_prefix + str(eni\_idx)

log.info("Querying EC2 for ENI with tag %s" % tag)

# query

response = None

try:

response = ec2\_client.describe\_network\_interfaces(

Filters=[{

"Name": "tag:cluster-eni-id",

"Values": [tag]

}]

)

except ClientError as e:

reason = "Failed to describe network interface with tag %s" % tag

log.exception(reason)

time.sleep(5)

eni\_idx += 1

continue

if not response["NetworkInterfaces"]:

break

for eni\_info in response["NetworkInterfaces"]:

eni\_id = eni\_info["NetworkInterfaceId"]

log.info("Found network interface %s " % eni\_id)

# detach

if "Attachment" in eni\_info and (

eni\_info["Attachment"]["Status"] == "attached" or

eni\_info["Attachment"]["Status"] == "attaching"

):

attachment\_id = eni\_info["Attachment"]["AttachmentId"]

if not detach\_eni(eni\_id, attachment\_id):

reason = "Failed to detach network interface %s" % eni\_id

log.error(reason)

try:

ec2\_client.delete\_network\_interface(

NetworkInterfaceId=eni\_id

)

log.info("Deleting network interface %s" % eni\_id)

except ClientError as e:

reason = "Failed to delete network interface %s" % eni\_id

log.exception(reason)

return cfn\_failure\_response(event, reason)

eni\_idx += 1

return cfn\_success\_response(event)

# Copyright 2002-2018 MarkLogic Corporation. All Rights Reserved.

import boto3

import botocore

import logging

import hashlib

import json

import time

from botocore.exceptions import ClientError

log = logging.getLogger()

log.setLevel(logging.INFO)

# global variables

ec2\_client = boto3.client('ec2')

asg\_client = boto3.client('autoscaling')

ec2\_resource = boto3.resource('ec2')

def eni\_wait\_for\_attachment(eni\_id):

max\_rety = 10

retries = 0

sleep\_interval = 10

eni\_info = None

while True and retries < max\_rety:

try:

eni\_info = ec2\_resource.NetworkInterface(id=eni\_id)

except ClientError as e:

reason = "Failed to get network interface by id %s" % eni\_id

log.exception(reason)

time.sleep(sleep\_interval)

retries += 1

continue

if not eni\_info.attachment:

time.sleep(sleep\_interval)

retries += 1

continue

status = eni\_info.attachment["Status"]

if status == "attached":

break

elif status == "attaching":

time.sleep(sleep\_interval)

retries += 1

continue

else:

log.warning(

"Network interface %s in unexpected status: %s" % (eni\_id, status)

)

retries += 1

continue

else:

log.warning(

"Waiting for network interface %s attachment timed out" % eni\_id

)

def handler(event, context):

msg\_text = event["Records"][0]["Sns"]["Message"]

msg = json.loads(msg\_text)

if "LifecycleTransition" in msg and \

msg["LifecycleTransition"] == "autoscaling:EC2\_INSTANCE\_LAUNCHING":

log.info("Handle EC2\_INSTANCE\_LAUNCHING event %s" % (json.dumps(event, indent=2)))

on\_launch(msg)

# continue with the life cycle event

try:

asg\_client.complete\_lifecycle\_action(

LifecycleHookName=msg['LifecycleHookName'],

AutoScalingGroupName=msg['AutoScalingGroupName'],

LifecycleActionToken=msg['LifecycleActionToken'],

LifecycleActionResult='CONTINUE'

)

except botocore.exceptions.ClientError as e:

reason = "Error completing life cycle hook for instance"

log.exception(reason)

time.sleep(5) # sleep for 5 seconds to allow exception info being sent to CloudWatch

def on\_launch(msg):

instance\_id = msg["EC2InstanceId"]

log.info("Launch event of instance %s" % instance\_id)

try:

instance = ec2\_client.describe\_instances(InstanceIds=[instance\_id])

except botocore.exceptions.ClientError as e:

reason = "Failed to describe instance %s" % instance\_id

log.exception(reason)

time.sleep(5)

return False

# manage ENI

subnet\_id = instance['Reservations'][0]['Instances'][0]['SubnetId']

tags = instance['Reservations'][0]['Instances'][0]['Tags']

stack\_name = None

stack\_id = None

for tag in tags:

if tag["Key"] == "marklogic:stack:name":

stack\_name = tag["Value"]

if tag["Key"] == "marklogic:stack:id":

stack\_id = tag["Value"]

if stack\_name and stack\_id:

log.info("Subnet: %s, Stack Name: %s, Stack Id: %s" % (str(subnet\_id), stack\_name, stack\_id))

id\_hash = hashlib.md5(stack\_id.encode()).hexdigest()

eni\_tag\_prefix = stack\_name + "-" + id\_hash + "\_"

for i in range(0,200):

tag = eni\_tag\_prefix + str(i)

log.info("Querying unattached ENI with tag %s" % tag)

# query

response = ec2\_client.describe\_network\_interfaces(

Filters=[

{

"Name": "tag:cluster-eni-id",

"Values": [tag]

},

{

"Name": "status",

"Values": ["available"]

},

{

"Name": "subnet-id",

"Values": [subnet\_id]

}

]

)

if len(response["NetworkInterfaces"]) == 0:

log.info("No qualified ENI found")

continue

# attach the available ENI

for eni\_info in response["NetworkInterfaces"]:

eni\_id = eni\_info["NetworkInterfaceId"]

try:

attachment = ec2\_client.attach\_network\_interface(

NetworkInterfaceId=eni\_id,

InstanceId=instance\_id,

DeviceIndex=1

)

log.info("Attaching ENI %s to instance %s" % (eni\_id, instance\_id))

except botocore.exceptions.ClientError as e:

reason = "Error attaching network interface %s" % eni\_id

log.exception(reason)

time.sleep(5)

continue

eni\_wait\_for\_attachment(eni\_id)

break

else:

continue

break

else:

log.warning("Tags for stack name or stack id not found")