Add device cloud instance properties through the REST API

Published: 2020-11-05

Device cloud properties enable you to view information about your cloud environment in the ExtraHop system. You can identify the cloud instance name, type, and ID of a device along with the cloud account that owns the device and the ID of the Virtual Private Cloud that the device is in.

This guide provides instructions for adding an observation through both the ExtraHop API Explorer and example Python scripts for Amazon AWS and Microsoft Azure. If you update cloud properties with a REST API script, you can continuously retrieve information from your cloud provider to make sure that your cloud property information is always up to date.

Before you begin

- You must log in to the ExtraHop system with an account that has full write privileges to generate an API key.
- You must have a valid API key to make changes through the REST API and complete the procedures below. (See Generate an API key.)
- Familiarize yourself with the ExtraHop REST API Guide to learn how to navigate the ExtraHop API Explorer.

Add cloud instance properties through the ExtraHop API Explorer

1. In a browser, navigate to the ExtraHop API Explorer.

   The URL is the hostname or IP address of your ExtraHop system, followed by `/api/v1/explore/`. For example, if your hostname is seattle-eda, the URL is `https://seattle-eda/api/v1/explore/`.
2. Click **Enter API Key** and then paste or type your API key into the **API Key** field.
3. Click **Authorize** and then click **Close**.
4. Find the ID of the device on the ExtraHop system by searching for the device MAC address.
   a) Click **Device** and then click **POST /devices/search**.
   b) Click **Try it out**.
   c) In the body field, specify the following JSON, replacing MACADDRESS with the MAC address of your cloud device:

   ```
   {
     "filter": {
       "field": "macaddr",
       "operand": "MACADDRESS",
       "operator": "="
     }
   }
   ```

   d) Click **Send Request**.
   e) In the Response body section, view and record the value of the `id` field for each device that is returned.
5. Add the cloud device metadata to the ExtraHop system.
   a) Click **PATCH /devices/{id}**.
   b) Click **Try it out**.
   c) In the `id` field, specify an ID.

d) In the body field, specify the following JSON, replacing the `string` values with properties from your cloud environment:

```
{
  "cloud_account": "string",
  "cloud_instance_id": "string",
  "cloud_instance_name": "string",
  "cloud_instance_type": "string",
  "vpc_id": "string"
}
```

e) Click **Send Request**.

Lambda Python Script Example for AWS

The following script imports AWS EC2 instance properties into the ExtraHop system. The script maps network interfaces of EC2 instances to devices discovered on the ExtraHop system by MAC address.

The script is designed to run as a Lambda function within AWS. Here are some important considerations for running the script in AWS:

- The script is designed to run on a set time interval. Each time the script is run, it scans each instance on the VPC and updates the corresponding devices in the ExtraHop system. For information about configuring a Lambda function to run periodically, see the AWS tutorial here.
- The Lambda function must be able to access resources on your VPC. For more information, see the AWS tutorial here.
- The Lambda function must have list and read access to the DescribeInstances action for the EC2 service. For more information, see the AWS tutorial here.
- The script imports the requests python module, which is not available to Lambda functions by default. For information about importing third-party libraries into Lambda, see the AWS tutorial here.

The script includes the following configuration variables that you must replace with information from your ExtraHop system:

- **HOSTNAME:** The private IP address or hostname of the EC2 instance of the ExtraHop system.
- **APIKEY:** The ExtraHop API key.

```
import boto3
import json
import requests
import os
import sys
from botocore.config import Config

APIKEY = '123456789abcdefghijklmnop'
HOST = 'https://extrahop.example.com'

# Method that adds the ExtraHop device IDs
# to the map object
def addEHids(mac_addresses, mapping):
    url = HOST + '/api/v1/devices/search'
    headers = {'Authorization': 'ExtraHop apikey=%s' % APIKEY}
    rules = []
    for macaddr in mac_addresses:
        rules.append({
            "field": "macaddr",
            "operand": macaddr,
            "operator": "="
        })
    search = {
```

```
        "filter": {
            "operator": "or",
            "rules": rules
        }
    }
    r = requests.post(url, headers=headers, verify=False,
 data=json.dumps(search))
    if r.status_code != 200:
        print('Error! Unable to retrieve devices from ExtraHop.')
        print(r.text)
        sys.exit()
    mac_id_map = {}
    for device in r.json():
        macaddr = device['macaddr'].lower()
        if device['is_l3'] == True:
            continue
        if macaddr in mac_id_map:
            mac_id_map[macaddr].append(device['id'])
        else:
            mac_id_map[macaddr] = [device['id']]
    to_do = []
    not_found = []
    for instance in mapping:
        aws_mac = instance['macaddr']
        if aws_mac in mac_id_map:
            instance['id'] = mac_id_map[aws_mac]
            instance.pop('macaddr')
            to_do.append(instance)
        else:
            not_found.append(aws_mac)
    return to_do, not_found

def updateMeta(device, dev_id):
    url = HOST + '/api/v1/devices/' + str(dev_id)
    headers = {'Authorization': 'ExtraHop apikey=%s' % APIKEY}
    r = requests.patch(url, headers=headers, data=json.dumps(device))
    if r.status_code != 204:
        return False
    else:
        return True

def lambda_handler(event, context):
    aws_map = []
    print('start')
    ec2 = boto3.client('ec2', config=Config(connect_timeout=10,
 read_timeout=10, retries={'total_max_attempts': 50}))
    response = ec2.describe_instances()
    reservations = response['Reservations']
    for reservation in reservations:
        for instance in reservation['Instances']:
            instance_id = instance['InstanceId']
            instance_type = instance['InstanceType']
            instance_name = ''
            for tag in instance['Tags']:
                if tag['Key'] == 'Name':
                    instance_name = tag['Value']
            for interface in instance['NetworkInterfaces']:
                instance_owner = interface['OwnerId']
                vpc_id = interface['VpcId']
                macaddr = interface['MacAddress']
                aws_map.append({
                    'macaddr': macaddr,
                    'cloud_instance_id': instance_id,
                    'cloud_instance_type': instance_type,
```

```
                    'cloud_instance_name': instance_name,
                    'cloud_account': instance_owner,
                    'vpc_id': vpc_id
                })
    mac_addresses = [x['macaddr'] for x in aws_map]
    aws_map, not_found = addEHids(mac_addresses, aws_map)
    updates = []
    failed = []
    for device in aws_map:
        ids = device.pop('id')
        for i in ids:
            success = updateMeta(device, i)
            if success:
                updates.append(str(i))
            else:
                failed.append(str(i))
    results = {
        'updated_device_ids': updates,
        'update_failed_device_ids': failed,
        'macaddr_not_found_on_eh': not_found
    }
    return {
        'statusCode': 200,
        'body': json.dumps(results)
    }
```

Note: If the script returns an error message that the SSL certificate verification failed, make sure that a trusted certificate has been added to your ExtraHop system. Alternatively, you can add the `verify=False` option to bypass certificate verification. However, this method is not secure and is not recommended. The following code sends an HTTP GET request without certificate verification:

```
requests.get(url, headers=headers, verify=False)
```

Python Script Example for Azure

The following script imports Azure device properties into the ExtraHop system. The following script assigns cloud device properties to every device discovered by the ExtraHOp system with a MAC address that belongs to an Azure VM network interface. The script is designed to be run on a set time interval. Each time the script is run, it scans each VM and updates the corresponding devices in ExtraHop.

The script requires the following modules from the Azure Python SDK:

- azure.mgmt.compute
- azure.mgmt.network
- azure.common.credentials

The script also requires you to have configured Azure authentication credentials in the following environment variables on the machine that runs the script:

- AZURE_SUBSCRIPTION_ID
- AZURE_CLIENT_ID
- AZURE_CLIENT_SECRET
- AZURE_TENANT_ID

For information about generating these credentials, see the Azure documentation.

The script includes the following configuration variables that you must replace with information from your ExtraHop system:

• **HOSTNAME:** The IP address or hostname of the ExtraHop system.

- **APIKEY:** The ExtraHop API key.

```
from azure.mgmt.compute import ComputeManagementClient
from azure.mgmt.network import NetworkManagementClient
from azure.common.credentials import ServicePrincipalCredentials
import os
import json
import sys
import requests

HOST = 'https://extrahop.example.com'
APIKEY = '123456789abcdefghijklmnop'

subscription_id = os.environ.get('AZURE_SUBSCRIPTION_ID')
credentials = ServicePrincipalCredentials(
    client_id=os.environ['AZURE_CLIENT_ID'],
    secret=os.environ['AZURE_CLIENT_SECRET'],
    tenant=os.environ['AZURE_TENANT_ID']
)
compute_client = ComputeManagementClient(credentials, subscription_id)
network_client = NetworkManagementClient(credentials,subscription_id)
region = 'eastus'


result_list_pub = compute_client.virtual_machine_images.list_publishers(
    region,
)

def addEHids(mac_addresses, mapping):
    url = HOST + '/api/v1/devices/search'
    headers = {'Authorization': 'ExtraHop apikey=%s' % APIKEY}
    rules = []
    for macaddr in mac_addresses:
        rules.append({
            "field": "macaddr",
            "operand": macaddr,
            "operator": "="
        })
    search = {
        "filter": {
            "operator": "or",
            "rules": rules
        }
    }
    r = requests.post(url, headers=headers, verify=False,
 data=json.dumps(search))
    if r.status_code != 200:
        print('Error! Unable to retrieve devices from ExtraHop.')
        print(r.text)
        sys.exit()
    mac_id_map = {}
    for device in r.json():
        macaddr = device['macaddr']
        if device['is_l3'] == True:
            continue
        if macaddr in mac_id_map:
            mac_id_map[macaddr].append(device['id'])
        else:
            mac_id_map[macaddr] = [device['id']]
    return_list = []
    for instance in mapping:
        aws_mac = instance['macaddr']
        if aws_mac in mac_id_map:
```

```
            instance['id'] = mac_id_map[aws_mac]
            instance.pop('macaddr')
            return_list.append(instance)
        else:
            print('MAC address: ' + instance['macaddr'] + ' not found on the
 ExtraHop system')
    return return_list

def updateMeta(device, dev_id):
    url = HOST + '/api/v1/devices/' + str(dev_id)
    headers = {'Authorization': 'ExtraHop apikey=%s' % APIKEY}
    r = requests.patch(url, headers=headers, data=json.dumps(device))
    if r.status_code != 204:
        return False
    else:
        return True

az_map = []
for vm in compute_client.virtual_machines.list_all():
    vm_name = vm.name
    vm_id = vm.vm_id
    vm_size = vm.hardware_profile.vm_size
    subscription_id = vm.id.split('/')[2]
    for interface in vm.network_profile.network_interfaces:
        name = interface.id.split('/')[-1]
        subnet = interface.id.split('/')[4]
        nic_group = interface.id.split('/')[-5]
        nic = network_client.network_interfaces.get(nic_group, name)
        macaddr = nic.mac_address.replace('-',':')
        vnet_name = nic.ip_configurations[0].subnet.id.split('/')[-3]
        az_map.append({
            "macaddr": macaddr,
            "cloud_instance_name": vm_name,
            "cloud_instance_type": vm_size,
            "cloud_instance_id": vm_id,
            "cloud_account": subscription_id,
            "vpc_id": vnet_name
        })

mac_addresses = [x['macaddr'] for x in az_map]
az_map = addEHids(mac_addresses, az_map)

updates = []
failed = []
for device in az_map:
    ids = device.pop('id')
    for i in ids:
        success = updateMeta(device, i)
        if success:
            updates.append("Device ID: " + str(i))
        else:
            failed.append("Device ID: " + str(i))

if updates:
    print('Updated cloud properties for the following devices:')
    for u in updates:
        print('    ' + u)
if failed:
    print('Failed to update cloud properties for the following devices:')
    for f in failed:
        print('    ' + f)
```

Note: If the script returns an error message that the SSL certificate verification failed, make sure that a trusted certificate has been added to your ExtraHop system. Alternatively, you can add the `verify=False` option to bypass certificate verification. However, this method is not secure and is not recommended. The following code sends an HTTP GET request without certificate verification:

```
requests.get(url, headers=headers, verify=False)
```