

# Add device cloud instance properties through the REST API

Published: 2023-10-02

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 sensor or console 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

**Important:** The REST API Explorer is not available on Reveal(x) 360.

1. In a browser, navigate to the ExtraHop API Explorer.  
The URL is the hostname or IP address of your sensor or console, 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 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.
  - 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**.


## Retrieve and install the example Lambda Python script for AWS

The ExtraHop GitHub repository contains an example Python script that 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](#).

 **Important:** The example python script authenticates to the sensor or console through an API key, which is not compatible with the Reveal(x) 360 REST API. To run this script with Reveal(x) 360, you must modify the script to authenticate with API tokens. See the [py\\_rx360\\_auth.py](#) script in the ExtraHop GitHub repository for an example of how to authenticate with API tokens.

 **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 sensor or console](#). Alternatively, you can add the `verify=False` option to bypass certificate verification. However, this method is not secure and not recommended. The following code sends an HTTP GET request without certificate verification:

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

1. Go to the ExtraHop [code-examples GitHub repository](#) and download the `add_cloud_props_lambda/add_cloud_props_lambda.py` file to your local machine.
2. In a text editor, open the `add_cloud_props_lambda.py` file and replace the following configuration variables with information from your environment:
  - **HOSTNAME:** The private IP address or hostname of the sensor or console EC2 instance.
  - **APIKEY:** The ExtraHop API key.
3. Add the `add_cloud_props_lambda.py` file to a zip file with the `requests` Python module. The script imports the `requests` Python module, which is not available to Lambda functions by default. For information about creating a zip file to import third-party libraries into Lambda, see the [AWS documentation](#).
4. In AWS, create a Lambda function. For more information about creating Lambda functions, see the [AWS documentation](#).

5. On the Lambda function page, click **Actions** and select **Upload a .zip** file.
6. Select the zip file you created.

## Retrieve and install the example Python script for Azure

The ExtraHop GitHub repository contains an example Python script that imports Azure device properties into the ExtraHop system. The 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](#).

 **Important:** The example python script authenticates to the sensor or console through an API key, which is not compatible with the Reveal(x) 360 REST API. To run this script with Reveal(x) 360, you must modify the script to authenticate with API tokens. See the [py\\_rx360\\_auth.py](#) script in the ExtraHop GitHub repository for an example of how to authenticate with API tokens.

1. Go to the [ExtraHop code-examples GitHub repository](#) and download the `add_cloud_props_azure/add_cloud_props_azure.py` file to your local machine.
2. In a text editor, open the `add_cloud_props_azure.py` file and replace the following configuration variables with information from your environment:
  - **HOSTNAME:** The IP address or hostname of the sensor or console.
  - **APIKEY:** The ExtraHop API key.
3. Run the following command:

```
python3 add_cloud_props_azure.py
```

 **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 sensor or console](#). Alternatively, you can add the `verify=False` option to bypass certificate verification. However, this method is not secure and not recommended. The following code sends an HTTP GET request without certificate verification:

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