ExtraHop Rest API Guide

Version 5.0

Introduction to ExtraHop REST API

The ExtraHop REST application programming interface (API) enables you to automate administration and configuration tasks on your ExtraHop Discover and Command appliances. You can send requests to the ExtraHop API through a Representational State Transfer (REST) interface, which is accessed through resource URIs and standard HTTP methods.

You can automate administration tasks, such as configuring LDAP authentication, saving customizations, applying SSL decrypt keys, and managing support packs. And, you can automate configuration tasks, such as creating alerts, or writing triggers.

When a REST API request is sent over HTTPS to an ExtraHop appliance, that request is authenticated and then authorized through an API key. After authentication, the request is submitted to the ExtraHop system and the operation completes.

In addition, you can access the built in ExtraHop API Explorer tool through the Discover and Command appliances, which enables you to view all of the available system resources, methods, properties, and parameters. In addition, you can test out API calls directly on your ExtraHop appliance.

Note: This guide is intended for an audience that has a basic familiarity with software development and the ExtraHop system.

Get Started

If you have a user account for your ExtraHop appliance, you can connect to the ExtraHop API Explorer and begin browsing through the available resources.

1. From the ExtraHop Web UI, click the User icon, and then select **API Access**.
2. On the API Access page, click **REST API Explorer**.

Before you can begin coding against the ExtraHop REST API or performing operations through the ExtraHop API Explorer, you must meet the following requirements:

- Your ExtraHop appliance must be configured to allow API key generation for the type of user you are (remote or local).
- You must have a user account with appropriate privileges set for the type of tasks you want to perform.
- You must have access to the ExtraHop appliance.

Access and authenticate to the ExtraHop REST API
Learn about the ExtraHop REST API Explorer
Learn about the ExtraHop REST API
View Code Samples

Access and authenticate to the ExtraHop REST API

Administrators, or users with full system privileges, control whether users can generate API keys. For example, you can prevent remote users from generating keys or you can disable API key generation entirely. When this functionality is enabled, API keys are generated by users and can be viewed only by the user who generated the key.

Note: Administrators set up user accounts, and then users generate their own API key. Users can delete API keys for their own account, and users with full system privileges can delete API keys for any user. For more information, see the Users section in the ExtraHop Admin UI Guide.

The permission level that is set for a user dictates what that user can do through the REST API.

Permission level	Functionality
Full System Privileges	• Users can enable or disable API key generation for the ExtraHop appliance • Users can delete API keys for any user • Users can perform any operation available through the REST API • Users can view the last four digits and description for any API key on the system
Full Write Privileges	• Users can generate an API key • Users can view or delete their own API key • Users can perform any configuration task through the REST API that is available for the ExtraHop Web UI • Users cannot perform any administration task through the REST API that is available for the ExtraHop Admin UI
Limited Write Privileges	• Users can generate an API key • Users can view or delete their own API key • Users can modify personal customizations (such as a personal dashboard) through the REST API that is available for the ExtraHop Web UI • Users cannot perform any configuration task that might affect other users in the system • Users cannot perform any administration task through the REST API that is available for the ExtraHop Admin UI
Read-Only Privileges	• Users can generate an API key • Users can view or delete their own API key • Users can perform limited GET operations through the REST API for the ExtraHop Web UI

After you generate an API key, you must append the key to your request headers. The following example shows a request that would retrieve all of the alerts set on the ExtraHop system:

```
curl -i -X GET --header "Authorization: ExtraHop apikey=39284639207" \
--header "Accept: application/json" \
"https://<hostname-or-IP-of-your-ExtraHop-appliance>/api/v1/alerts"
```

Manage API Key Access

You can manage which users are able to generate API keys on the ExtraHop appliance.

1. Log in to the ExtraHop Admin UI through the following URL: https://<hostname-or-IP-of-your-ExtraHop-appliance>/admin

2. In the Access Settings section, click **API Access**.
3. In the Manage Access section, select one of the following options:
   - **Allow All User Generated API Keys**: Local and remote users can generate API keys.
   - **Local Users Only**: Only local users can generate API keys.
   - **No API Keys Allowed**: No API keys can be generated by any user.
4. Click **Save Settings**.

Generate an API Key

After you log into the ExtraHop appliance, if API key generation is enabled, you can generate an API key.

1. In the Access Settings section, click **API Access**.
2. In the API Keys section, type a description for the key, and then click **Generate**.
3. Copy the API key and paste the key into the REST API Explorer or append the key to a request header.

Delete an API Key

You can delete an API key from the ExtraHop appliance.

1. In the Access Settings section, click **API Access**.
2. In the Keys section, click the **X** next to the API key you want to delete.
3. Click **OK**.

Learn about the ExtraHop REST API Explorer

The ExtraHop API Explorer is a web-based tool that enables you to view detailed information about the ExtraHop REST API resources, methods, parameters, properties, and error codes. Code samples are available in Python, cURL, and Ruby for each resource. You also can perform operations directly through the tool, which are performed on your ExtraHop and return information about your network.

> **Note:** Be cautious when clicking the **Try it out!** button, because the operation is performed on your ExtraHop appliance.

Learn about the ExtraHop REST API

View Resource Information

Click on any resource group in the ExtraHop REST API Explorer to view information about the available methods and the expected URL syntax for the resource.

The following options enable you to manage the information displayed on the main page.

Show/Hide: Expands and collapses information about the resource.

List Operations: Expands information about the resource operations.

Expand Operations: Expands information about all of the resource operations. Clicking the method or path of the expanded operation will collapse the additional information.

View Operation Information

Click on any operation to view additional configuration information for the resource. The following table provides information about the sections available for resources in the ExtraHop API Explorer. Section availability varies by HTTP method; not all methods have all of the sections listed in the table.

Section	Description

Response Class	Provides the response code and type for successful requests.
Parameters	Provides information about the available query parameters.
Response Messages	Provides additional information about the possible HTTP status codes for the resource.
Model	Provides the JSON body objects and descriptions.
Model Schema	Provides the JSON body schema. Red text indicates user-defined text values. Green text indicates Boolean and number values.

GET Requests

GET requests retrieve information about the objects in the associated resource. You can request information about all of the objects in a resource or you can specify an object ID to retrieve detailed information about only that object.

Examples

Retrieve a list of devices on the ExtraHop system through the ExtraHop API Explorer.

1. Click **Device**, and then click **GET /devices**.
2. In the Parameters section, modify the fields to build your query. For example, the default limit parameter restricts the number of devices returned to 100.
3. Click the **Try it out!** button. A successful GET request returns a response body with the number of devices specified in the limit parameter. A failed GET request returns the response body with an error.

Retrieve information only about a specific device. You must have a device ID, which you can retrieve from the previous example.

1. Click **Device**, and then click **GET /devices/{id}**.
2. In the Parameters section, in the **id** field, type the device ID.
3. Click the **Try it out!** button. A successful GET request returns a response body with information about the specified device. A failed GET request returns the response body with an error.

POST Requests

POST requests create objects and queries for the associated resource.

Examples

Create a custom dynamic FTP device group on the ExtraHop system through the ExtraHop API Explorer.

1. Click **Device Group**, and then click **POST /devicegroups**.
2. Click on **Model** to see descriptions for optional and required fields.
3. Click on **Model Schema**, and then click in the box to automatically add the schema to the body parameter.
4. Edit the JSON fields. In the following example, the **dynamic** parameter is set to "true", the **field** parameter is set to "type" and the **value** parameter is set to "/^extrahop.device.ftp_servers$/".

```
{
 "description": "FTP Example",
 "dynamic": true,
```

```
 "field": "type",
 "include_custom_devices": true,
 "name": "",
 "value": "/^extrahop.device.ftp_server$/"
}
```

5. Click **Try it out!**

Note: To see available parameter options, perform a GET request on the resource. The response body for the GET request displays possible values for the fields.

A 201 status is returned upon success, there is no response body, and the response headers display the location URL (/api/v1/devicegroups/151) and ID (151) for the device group as follows:

```
{
 "date": "Mon, 30 Nov 2015 19:00:28 GMT",
 "via": "1.1 localhost",
 "server": "Apache",
 "content-type": "text/plain; charset=utf-8",
 "location": "/api/v1/devicegroups/151",
 "cache-control": "private, max-age=0",
 "connection": "Keep-Alive",
 "keep-alive": "timeout=45, max=99",
 "content-length": "0"
}
```

PATCH Requests

PATCH requests update existing objects with modified or missing information.

Examples

1. Click **Device Group** and then click **PATCH /devicegroups/{id}**.
2. In the **id** parameter, type the ID for the device group. From the POST example above, the device group ID is 151.
3. Add and modify only the fields that you want to change to the body parameter. For example, to add a name for the device group that was created in the example above, type:

```
{
 "name": "FTP Servers"
}
```

4. Click **Try it out!**

A status code of 204 is returned upon success.

DELETE Requests

DELETE requests remove objects from the system. You must have an object ID to perform a DELETE operation.

Examples

1. Click **Device Group**, and then click **DELETE /devicegroups/{id}**.
2. In the **id** parameter, type the ID of the device group you want to delete. In the POST example above, the device group ID is 151.
3. Click **Try it out!** The device group is deleted from your ExtraHop appliance.

A status code of 204 is returned upon success.

PUT Requests

For limited operations, you can erase and replace the content in a resource with a PUT request.

Examples

1. Click **Licenses** and then click **PUT /licenses/productkey**.
2. Click in the **Model Schema** box to add the schema to the product_key parameter.
3. Replace the string with your product key.
4. Click **Try it out!**

A status code of 204 is returned upon success.

Learn about the ExtraHop REST API

The ExtraHop REST API enables you to automate tasks for the ExtraHop Web UI and Admin UI. In addition, you can view and try all of the available resources through the ExtraHop REST API Explorer and perform operations directly on your ExtraHop appliance.

Learn about the ExtraHop REST API Explorer
Identify Objects on the ExtraHop System
View Code Samples

ExtraHop API Resources

You can perform operations on the following resources through the ExtraHop REST API. You also can view more detailed information about these resources, such as available HTTP methods, query parameters, and object properties in the ExtraHop REST API Explorer.

Activity Group

Activity groups classify devices automatically based on their network traffic. You can retrieve IDs for all activity groups and then perform additional operations on a group that is associated with a single ID. For example, activity group IDs can be added to Metric queries to retrieve metrics simultaneously for a group of devices.

Alert

Alerts are system notifications that are generated upon an event. Default alerts are available in the system, or you can create a custom alert. Threshold alerts can be set to alert you if a metric crosses the value defined in the alert. Trend alerts cannot be configured through the REST API.

Application

Applications are user-defined groups that collect metrics identified through triggers across multiple types of traffic. The default All Activity application contains all collected metrics.

Audit Log

The audit log displays a record of all recorded system administration and configuration activity. The log displays the time of the activity, the user who performed the activity, the operation, operation details, and

system component.

Bundle

Bundles are JSON-formatted documents that contain information about selected system configuration, such as triggers, dashboards, applications, or alerts). You can create a bundle and then transfer those configurations to another ExtraHop appliance, or save the bundle as a backup. Bundles can also be downloaded from ExtraHop Solution Bundles and applied through the REST API.

Custom Device

You can create a custom device by defining a set of rules. For example, you can create a custom device that has an IP address on a specified VLAN. By default, all IP addresses outside of the locally-monitored broadcast domains are aggregated behind a router. To identify devices that are behind that router, you can create a custom device, and then collect metrics from the device.

Customization

Similar to Bundles, customizations enable you to save ExtraHop configurations for backup. Unlike with Bundles, however, you cannot select the information that is contained in a Customization. All of the major system changes are saved as a JSON-formatted document and can be uploaded to a restored or new ExtraHop appliance. Customizations are accessible only to users with Full System Permissions.

Dashboards

Dashboards are built-in or customized views of your ExtraHop metrics information.

Device

Devices are objects on your network that have been identified and classified by your ExtraHop appliance.

Device Group (or Custom Groups)

Device groups can be either static or dynamic. A static device group is user-defined; you create a custom group and then manually identify and assign each device to that group. A dynamic device group is defined and automatically managed by a set of configured rules. For example, you can create a custom group and then set a rule to classify all devices within a certain IP address range to be added to that group automatically.

Email Group

You can add individual or group email addresses to an email group and assign them to a system alert. When that alert is triggered, the system sends an email to all of the addresses in the email group.

Exclusion Intervals

An exclusion interval can be created to set a time period to suppress an alert. For example, if you do not want to be notified about alerts after hours or on the weekends, an exclusion interval can create a rule to suppress the alert during that time period.

ExtraHop

This resource provides metadata about the ExtraHop appliance, such as the firmware version or if the appliance is a Command appliance.

Flex Grid

A Flex Grid provides a table view of metrics information about devices.

Geomap

Geomaps display metrics across a global map, which indicates where metrics activity has occurred.

License

This resource enables you to retrieve and set product keys or to set a license. (License information cannot be retrieved.)

Metrics

Metrics information is collected about every object identified by the ExtraHop appliance. Note that metrics are retrieved through the POST method, which creates a query to collect the requested information through the API.

Network

Networks are correlated to the network interface card that receives input from all of the objects identified by the ExtraHop appliance. On an ExtraHop Command appliance, each node is identified as a network capture that is looking at the traffic for each ExtraHop Discover node that is connected to the Command cluster.

Node

A node is defined by its relationship to an ExtraHop Command appliance. The environment which contains Discover nodes and a Command appliance is called a Command cluster.

Packet Capture

Packet captures store packets from a network traffic flow. You must write a trigger to identify the information you want to generate. For example, you can write a trigger to collect all of the packets going to a particular device that is generating a high volume of errors. Then, you can download or delete that information.

Pages

Pages provide a template for creating a customized view of built-in metrics or metrics collected from triggers.

Running Config

The running config is a JSON document that contains core system configuration information for the ExtraHop appliance.

SSL Decrypt Key

This resource enables you to add a decryption key for your network traffic.

Support Pack

A support pack is a file that contains configuration adjustments provided by ExtraHop Support.

Tag

Device tags enable you to associate a device or group of devices by some characteristic. For example, you might tag all of your HTTP servers or tag all of the devices that are in a common subnet.

Trigger

Triggers are custom scripts that perform an action upon a pre-defined event. For example, you can write a trigger to record a custom metric every time an HTTP request occurs, or classify traffic for a particular server as an Application server. For more information, see the Trigger API Reference.

User

This resource enables you to manage the list of users who have access to the ExtraHop appliance and their permission levels.

Vlan

Virtual lans are logical groupings of traffic or devices on the network.

Whitelist

A whitelist prioritizes devices that are added after the device limit is reached. When the device limit is reached, any new devices that are added are placed in limited analysis mode. If you want to prioritize a device, you can add that device to the whitelist. Devices are removed from the ExtraHop device list based on seniority; the latest devices are removed first.

Identify Objects on the ExtraHop System

Objects on the ExtraHop system can be identified by any unique value, such as the IP address, MAC address, name, or system ID. To perform API operations on a specific object, you must locate the object ID.

You can locate an object ID through the following methods:

- The object ID is provided in the headers returned from a POST request. For example, if you send a POST request to create a page, the response headers display a location URL, such as:

```
{
 "date": "Wed, 25 Nov 2015 17:39:06 GMT",
 "via": "1.1 localhost",
 "server": "Apache",
 "content-type": "text/plain; charset=utf-8",
 "location": "/api/v1/pages/221",
 "cache-control": "private, max-age=0",
 "connection": "Keep-Alive",
 "keep-alive": "timeout=45, max=89",
 "content-length": "0"
}
```

  The location for the newly created page is /api/v1/pages/221 and the ID for the page is 221.

- The object ID is provided for all objects returned from a GET request. For example, if you perform a GET request on all devices, the response body contains information for each device, including the ID.

  The following response body displays an entry for a single device, with an ID of 10212:

```
{
 "mod_time": 1448474346504,
 "node_id": null,
 "id": 10212,
 "extrahop_id": "test0001",
 "description": null,
 "user_mod_time": 1448474253809,
```

```
 "discover_time": 1448474250000,
 "vlanid": 0,
 "parent_id": 9352,
 "macaddr": "00:05:G3:FF:FC:28",
 "vendor": "Cisco",
 "is_l3": true,
 "ipaddr4": "10.10.10.5",
 "ipaddr6": null,
 "device_class": "node",
 "default_name": "Cisco5",
 "custom_name": null,
 "cdp_name": "",
 "dhcp_name": "",
 "netbios_name": "",
 "dns_name": "",
 "custom_type": "",
 "analysis_level": 1
},
```


To perform further requests on a specific device, add the ID in the request for that device.

- The object ID is provided in the URL for most objects. For example, in the ExtraHop Web UI, click on Metrics, and then Devices. Select any device. The URL for that device page displays an OID-D=<number>, such as:

```
https://10.10.10.205/extrahop/#/Devices?details=true&device
Oid=10180&from=6&interval_type=HR&until=0&view=l2stats
```

To perform further requests for that device, add 10180 to the **id** field in the ExtraHop API Explorer or to the body parameter in your request.

The URL for dashboards displays a short_code, which appears after /Dashboard. When you add the short_code to the ExtraHop API Explorer or to your request, you must prepend a tilde to the short code.

In the following example, kmC9Y is the short_code:

```
https://10.10.10.205/extrahop/#/Dashboard/kmC9Y/?from=6&interval_
type=HR&until=0
```

You can also find the short_code in Dashboard Properties in the command menu from any dashboard.

View Code Samples

Sample 1: Set up an SSL Certificate

Before making requests to an ExtraHop appliance with a self-signed certificate, you must set up an SSL certificate for each user who will access the ExtraHop appliance from a particular computer.

In each of the following examples, replace {HOST} with the hostname of your ExtraHop system and replace {API KEY} with a valid API key from your ExtraHop system.

Note: The SSL certificate applies only to the user performing the command. Each user must run the command with their login credentials to set up the SSL certificate.

Set up SSL through Windows Powershell

```
Invoke-WebRequest "http://{HOST}/public.cer" -OutFile ($env:USERPROFILE +
"\ex.cer"); Import-Certificate ($env:USERPROFILE + "\ex.cer")
-CertStoreLocation Cert:\CurrentUser\Root
```

Set up SSL through OS X

```
curl -O http://{HOST}/public.cer; security add-trusted-cert -r trustRoot -k
~/Library/Keychains/login.keychain public.cer
```

Sample 2: Create and assign device tags

The following example shows how you can create a device tag and then assign that tag to all of the devices in a specified subnet.

```
#!/usr/bin/env python

import httplib
import urllib
import json
import sys

# Configuration Options:
host = "{HOST}"
apikey = "{API KEY}"
tag_name = "MyTestTag"
subnet = "10.20.0.[0-9]+"
batch_limit = 100
headers = {'Accept': 'application/json',
           'Authorization': "ExtraHop apikey=%s" % apikey}
conn = httplib.HTTPSConnection(host)

def execute_req(method, path, expected_code, failure_message, body=None):
"""
   Returns the body of a successful request,
   otherwise prints error and terminates
"""

conn.request(method, "/api/v1" + path, headers=headers, body=body)
resp = conn.getresponse()
if resp.status is not expected_code:
    print(failure_message)
    print(resp.read())
    sys.exit(1)
```

```
return resp

def execute_get(path, expected_code, failure_message):
    resp = execute_req("GET", path, expected_code, failure_message)
    return json.loads(resp.read())

def execute_create(path, body, expected_code, failure_message):
    """Returns ID of newly created resource"""
    resp = execute_req("POST", path, expected_code, failure_message, body)
    resp.read()  # drain the response
    return int(resp.getheader("location").split("/")[-1])

# First, search for the specified tag, by name
resp = execute_get("/tags", 200, "Unable to retrieve tags from ExtraHop")
tags = [tag for tag in resp if tag["name"] == tag_name]

if not tags:
    # tag is not found, create it
    body = json.dumps({"name": tag_name})
    tag_id = execute_create('/tags', body, 201, "Unable to create tag")
else:
    tag_id = tags[0]["id"]

query_params = {'limit': batch_limit,
                'search_type': 'ip address',
                'value': subnet}
query_string = urllib.urlencode(query_params)

# Paginate device results, building up a list of all devices to assign
device_ids = []
offset = 0

while True:
    path = "/devices?" + query_string + ("&offset=%d" % offset)
    resp = execute_get(path, 200, "Unable to retrieve devices")
    if not resp:
        break

    device_ids += [device["id"] for device in resp]
    offset += batch_limit

# Perform the assignments
resp = execute_req("POST", "/tags/%d/devices" % tag_id,
                   204, "Unable to perform assignments",
                   body=json.dumps({"assign": device_ids}))
resp.read()  # drain the response

# Check that assignments were successful
resp = execute_get("/tags/%d/devices" % tag_id,
                   200, "Unable to retrieve tag assignments")
```

```
assigned_device_ids = [device["id"] for device in resp]

successful = set(device_ids).issubset(set(assigned_device_ids))
if successful:
    print("%d devices assigned to tag" % len(device_ids))
else:
    print("Unable to assign all devices to tag")
```

Sample 3: Query for metrics about a specific device

The following request example shows how you can query for metrics from an HTTP client device with the ID 9363 and print the response.

```
import httplib

headers = {'Content-Type': 'application/json',
       'Accept': 'application/json',
       'Authorization': 'ExtraHop apikey={API KEY}'
body = r"""{
  "cycle": "auto",
  "from": -1800000,
  "until": 0,
  "metric_category": "http_client",
  "metric_specs": [
    {
       "name": "req"
    }
   ],
   "object_ids": [
      9363
   ],
   "object_type": "device"
}"""
conn = httplib.HTTPSConnection('{HOST}')
conn.request('POST', '/api/v1/metrics', headers=headers, body=body)
resp = conn.getresponse()
print resp.status, resp.reason
print resp.read()
```

The following response shows entries for the device with ID 9363:

```
{
 "date": "Thu, 19 Nov 2015 23:20:07 GMT",
 "via": "1.1 localhost",
 "server": "Apache",
 "vary": "Accept-Encoding",
 "content-type": "application/json; charset=utf-8",
 "cache-control": "private, max-age=0",
 "connection": "Keep-Alive",
```

```
 "content-encoding": "gzip",
 "keep-alive": "timeout=45, max=44",
 "content-length": "277"
}

{
 "stats": [
   {
    "oid": 9363,
    "time": 1447973460000,
    "duration": 30000,
    "values": [
     2
    ]
   },
   {
    "oid": 9363,
    "time": 1447973490000,
    "duration": 30000,
    "values": [
     0
    ]
   },
   {
    "oid": 9363,
    "time": 1447973520000,
    "duration": 30000,
    "values": [
      1
     ]
    },
    {
     "oid": 9363,
     "time": 1447973550000,
     "duration": 30000,
     "values": [
       2
      ]
     }
```

Sample 4: Create, retrieve, and delete an object

This example shows how you can create and successfully retrieve information about a device tag. Then, after the device tags are deleted, the example shows how an attempt to retrieve information subsequently fails.

The following example shows how to create a device tag called my_test_tag.

```
curl -i -X POST --header "Content-Type: application/json" \
--header "Accept: application/json" \
--header "Authorization: ExtraHop apikey={API KEY}" \
```

```
-d "{
\"name\": \"my_test_tag\"
}" "https://{HOST}/api/v1/tags"
```

A 201 status returns upon success with the following response headers, which display that the tag was created, and provides the device tag location and ID of /api/v1/tags/1.

```
{
 "date": "Wed, 18 Nov 2015 20:24:13 GMT",
 "via": "1.1 localhost",
 "server": "Apache",
 "content-type": "text/plain; charset=utf-8",
 "location": "/api/v1/tags/1",
 "cache-control": "private, max-age=0",
 "connection": "Keep-Alive",
 "keep-alive": "timeout=45, max=88",
 "content-length": "0"
}
```

Next, the ID (1) is added to the following GET request, which returns a 200 status upon success and the JSON representation of the retrieved tag:

```
curl -i -X GET --header "Accept: application/json" \
--header "Authorization: ExtraHop apikey={API KEY}" \
"https://{HOST}/api/v1/tags/1"
{
 "mod_time": 1447878253953,
 "id": 1,
 "name": "my_test_tag"
}
```

Next, the following example shows a DELETE request to remove the device tag from the system, which returns a 204 status upon success:

```
curl -i -X DELETE --header "Accept: application/json" \
--header "Authorization: ExtraHop apikey={API KEY}" \
"https://{HOST}/api/v1/tags/1"
```

Finally, when another GET request is sent for that deleted device tag, the operation fails, and a 404 status is returned upon failure, indicating that the tag is no longer available.

```
curl -i -X GET --header "Accept: application/json" \
--header "Authorization: ExtraHop apikey={API KEY}" \
"https://{HOST}/api/v1/tags/1"
```