ExtraHop 7.4
ExtraHop REST API Guide

For more documentation, see https://docs.extrahop.com/.

Published: 2019-10-17

ExtraHop Networks
Seattle, WA 98101
877-333-9872 (US)
+44 (0)203 7016850 (EMEA)
+65-31585513 (APAC)
www.extrahop.com

Contents

Introduction to the ExtraHop REST API 5
ExtraHop API requirements 5

Access and authenticate to the ExtraHop REST API 6
Privilege levels 6
Manage API key access 8
Generate an API key 8
Set up an SSL certificate 8
Enable CORS for the ExtraHop REST API 9
Configure cross-origin resource sharing (CORS) 9

Learn about the REST API Explorer 10
Open the REST API Explorer 10
View resource information 10
View operation information 11
Identify objects on the ExtraHop system 11

ExtraHop API resources 13
Activity group 13
Activity Map 13
Alert 14
Alert severity levels 15
Analysis Priority 16
APIKey 16
Appliance 17
Application 17
Audit log 18
Bundle 18
Custom device 19
Customization 19
Dashboards 20
Device 20
Supported time units 21
Device group 22
Supported time units 24
Detections 24
Email group 25
Exclusion intervals 25
ExtraHop 26
License 26
Metrics 27
Supported time units 28
Network 29
Network locality entry 30
Node 30
Packet capture 31
Page 31
Record Log 32

Operand values in record queries 32
Supported time units 33
Report 34
Running config 35
SSL decrypt key 35
Support pack 36
Tag 36
Threat Collection 37
Trigger 37
Advanced trigger options 38
User 40
User group 41
VLAN 41
Whitelist (Watchlist) 42

ExtraHop REST API examples 43

Change a dashboard owner through the REST API 43
Retrieve the dashboard IDs 43
Change the dashboard owner 44
Python Script Example 45
Extract the device list through the REST API 46
Extract the device list through the REST API Explorer 46
Python script example 46
Create a trusted SSL certificate through the REST API 47
Create an SSL certificate signing request 47
Add a trusted SSL certificate to your ExtraHop appliance 49
Create custom devices through the REST API 49
Create a custom device 50
Add custom device criteria 50
Python script example 50
Create and assign a device tag through the REST API 52
Query for metrics about a specific device through the REST API 53
Create, retrieve, and delete an object through the REST API 54
Query the record log 55

Introduction to the ExtraHop REST API

The ExtraHop REST API enables you to automate administration and configuration tasks on your ExtraHop 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.

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.

Each ExtraHop appliance provides access to the built-in ExtraHop REST API Explorer, which enables you to view all of the available system resources, methods, properties, and parameters. The REST API Explorer also enables you to send API calls directly to your ExtraHop appliance.

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

ExtraHop API requirements

Before you can begin writing scripts for the ExtraHop REST API or performing operations through the REST 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 generate a valid API key.
- You must have a user account on the ExtraHop appliance with appropriate privileges set for the type of tasks you want to perform.

Access and authenticate to the ExtraHop REST API

Administrators, or users with unlimited 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 assign privileges, but then users generate their own API keys. Users can delete API keys for their own account, and users with unlimited privileges can delete API keys for any user. For more information, see Users and user groups.

After you generate an API key, you must append the key to your request headers. The following example shows a request that retrieves metadata about the firmware running on the ExtraHop appliance:

```
curl -i -X GET --header "Accept: application/json" \
--header "Authorization: ExtraHop apikey=2bc07e55971d4c9a88d0bb4d29ecbb29" \
"https://<hostname-or-IP-of-your-ExtraHop-appliance>/api/v1/extrahop"
```

Privilege levels

User privilege levels determine which ExtraHop Web UI and ExtraHop Admin UI tasks the user can perform through the ExtraHop REST API.

You can view the privilege levels for users through the `granted_roles` and `effective_roles` properties. The `granted_roles` property shows you which privilege levels are explicitly granted to the user. The `effective_roles` property shows you all privilege levels for a user, including those received outside of the granted role, such as through a user group.

The `granted_roles` and `effective_roles` properties are returned by the following operations:

- GET /users
- GET /users/{username}

The `granted_roles` and `effective_roles` properties support the following privilege levels:

Privilege level	Actions allowed
"system": "full"	• Enable or disable API key generation for the ExtraHop appliance. • Generate an API key. • View the last four digits and description for any API key on the system. • Delete API keys for any user. • View and edit cross-origin resource sharing. • Transfer ownership of any non-system dashboard to another user. • Perform any Admin UI task available through the REST API. • Perform any Web UI task available through the REST API.
"write": "full"	• Generate your own API key. • View or delete your own API key. • Change your own password, but you cannot perform any other Admin UI tasks through the REST API. • Perform any Web UI task available through the REST API.
"write": "limited"	• Generate an API key.

Privilege level	Actions allowed
	• View or delete their own API key. • Change your own password, but you cannot perform any other Admin UI tasks through the REST API. • Perform all GET operations through the REST API. • Modify the sharing status of dashboards that you are allowed to edit. • Delete dashboards and activity maps that you own. • Perform metric and record queries.
"write": "personal"	• Generate an API key. • View or delete your own API key. • Change your own password, but you cannot perform any other Admin UI tasks through the REST API. • Perform all GET operations through the REST API. • Delete dashboards and activity maps that you own. • Perform metric and record queries.
"metrics": "full"	• Generate an API key. • View or delete your own API key. • Change your own password, but you cannot perform any other Admin UI tasks through the REST API. • View dashboards and activity maps shared with you. • Perform metric and record queries.
"metrics": "restricted"	• Generate an API key. • View or delete your own API key. • Change your own password, but you cannot perform any other Admin UI tasks through the REST API. • View dashboards and activity maps shared with you.
"packets": "full"	• View and download packets from an ExtraHop Discover appliance through the `GET/packetcaptures/{id}` operation. This is an add-on privilege that can be granted to a user with one of the following privilege levels: • "write": "full" • "write": "limited" • "write": "personal" • "write": null • "metrics": "full" • "metrics": "restricted"
"packets": "full_with_keys"	• View and download packets from an ExtraHop Discover appliance through the `GET/packetcaptures/{id}` operation. This is an add-on privilege that can be granted to a user with one of the following privilege levels: • "write": "full" • "write": "limited" • "write": "personal" • "write": null

Privilege level	Actions allowed
	• "metrics": "full" • "metrics": "restricted" **Note:** Although this privilege level enables a user to view and download session keys through the Web UI, you cannot access session keys through the REST API.

Manage API key access

Users with unlimited privileges can manage which users are able to generate API keys on the ExtraHop appliance.

1. Log into 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 API Access section, select one of the following options:
   - **Allow all users to generate an API key**: Local and remote users can generate API keys.
   - **Only local users can generate an API key**: Remote users cannot generate API keys.
   - **No users can generate an API key**: No API keys can be generated by any user.
4. Click **Save Settings**.

Generate an API key

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

Before you begin
Make sure the ExtraHop appliance is configured to allow API key generation.

1. In the Access Settings section, click **API Access**.
2. In the Generate an API Key section, type a description for the new key, and then click **Generate**.
3. Scroll down to the API Keys section, and copy the API key that matches your description.

You can paste the key into the REST API Explorer or append the key to a request header.

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
```

Enable CORS for the ExtraHop REST API

Cross-origin resource sharing (CORS) allows you to access the ExtraHop REST API across domain-boundaries and from specified web pages without requiring the request to travel through a proxy server.

You can configure one or more allowed origins or you can allow access to the ExtraHop REST API from any origin. Only administrative users with unlimited privileges can view and edit CORS settings.

Configure cross-origin resource sharing (CORS)

Cross-origin resource sharing (CORS) allows you to access the ExtraHop REST API across domain-boundaries and from specified web pages without requiring the request to travel through a proxy server.

You can configure one or more allowed origins or you can allow access to the ExtraHop REST API from any origin. Only administrative users with unlimited privileges can view and edit CORS settings.

1. In the **Access Settings** section, click **API Access**.
2. In the CORS Settings section, specify one of the following access configurations.
   - To add a specific URL, type an origin URL in the text box, and then click the plus (+) icon or press ENTER.

     The URL must include a scheme, such as `HTTP` or `HTTPS`, and the exact domain name. You cannot append a path; however, you can provide a port number.
   - To allow access from any URL, select the Allow API requests from any Origin checkbox.

     Note: Allowing REST API access from any origin is less secure than providing a list of explicit origins.
3. Click **Save Settings** and then click **Done**.

Learn about the REST API Explorer

The REST 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.

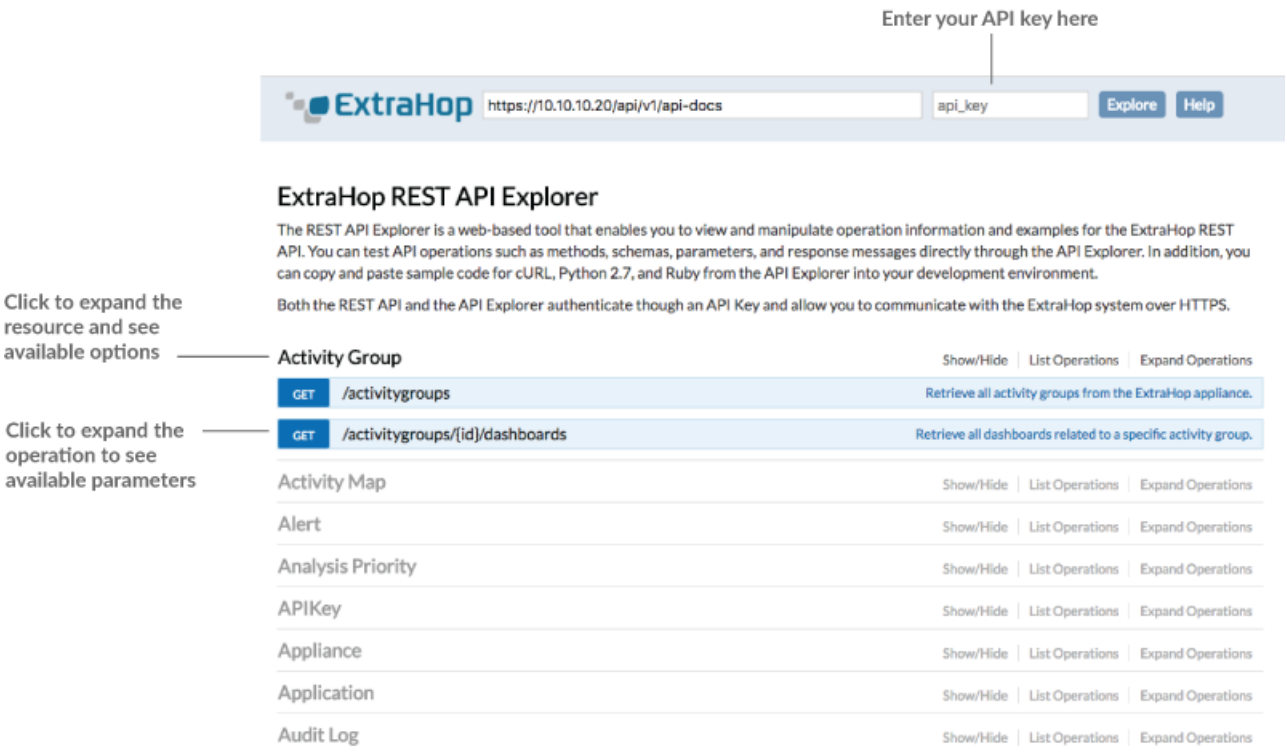
Important: Clicking the **Try it out!** button causes the specified operation to be performed on your ExtraHop appliance.

Open the REST API Explorer

You can open the REST API Explorer from the ExtraHop Admin UI or through the following URL:

```
https://<extrahop-hostname-or-ip-address>/api/v1/explore/
```

1. Log into the Admin UI on the ExtraHop appliance.
2. From the Access Setting section, click **API Access**.
3. On the API Access page, click **REST API Explorer**.
   The REST API Explorer opens in your browser.



View resource information

Click on any resource group in the REST API Explorer to view the available operations 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

From the REST API Explorer, you can click any operation to view configuration information for the resource.

The following table provides information about the sections available for resources in the REST API Explorer. Section availability varies by HTTP method. Not all methods have all of the sections listed in the table.

Section	Description
Implementation Notes	Provides all of the fields for the request body and supported values for each field.
Response Class	Provides the response code and type for successful requests.
Parameters	Provides information about the available query parameters.
Response Messages	Provides 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 strings. Green text indicates Boolean and number values.

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 OIDD=<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 REST 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 REST 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.

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 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. For more information, see Activity groups.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /activitygroups	Retrieve all activity groups from the ExtraHop appliance.
GET /activitygroups/{id}/dashboards	Retrieve all dashboards related to a specific activity group.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Activity Map

An activity map is a dynamic visual representation of the L4-L7 protocol activity between devices in your network. Create a 2D or 3D layout of device connections in real-time to learn about the traffic flow and relationships between devices.

Here are some important considerations about activity maps:

- You can only create activity maps for devices in Standard Analysis and Advanced Analysis. Discovery Mode devices are not included in activity maps. For more information, see Analysis levels.
- If you create an activity map for a device, activity group, or device group with no protocol activity in the selected time interval, the map appears without any data. Change the time interval or your origin selection and try again.
- You can create an activity map in a Command appliance to view device connections across all of your Discover appliances. However, connected Discover appliances must be upgraded to firmware version 7.0 or later.

To learn about configuring and navigating activity maps, see Activity maps.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /activitymaps	Retrieve all activity maps.
POST /activitymaps	Create a new activity map.
POST /activitymaps/query	Perform a network topology query, which returns activity map data in flat file content.

Operation	Description
DELETE /activitymaps/{id}	Delete a specific activity map.
GET /activitymaps/{id}	Retrieve a specific activity map.
PATCH /activitymaps/{id}	Update a specific activity map.
POST /activitymaps/{id}/query	Perform a topology query for a specific activity map, which returns activity map data in flat file content.
GET /activitymaps/{id}/sharing	Retrieve the users and their sharing permissions for a specific activity map.
PATCH /activitymaps/{id}/sharing	Update the users and their sharing permissions for a specific activity map.
PUT /activitymaps/{id}/sharing	Replace the users and their sharing permissions for a specific activity map.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Alert

Alerts are system notifications that are generated upon specified alert criteria. Default alerts are available in the system, or you can create a custom alert.

Detections and threshold alerts can be set to alert you if a metric crosses the value defined in the alert configuration. Trend alerts cannot be configured through the REST API. For more information, see Alerts.

Note: Detections require a connection to the cloud-based ExtraHop Machine Learning Service.

The following table displays all of the operations you can perform this resource:

Operation	Description
GET /alerts	Retrieve all alerts.
POST /alerts	Create a new alert with specified values.
DELETE /alerts{id}	Delete a specific alert.
GET /alerts{id}	Retrieve a specific alert.
PATCH /alerts{id}	Apply updates to a specific alert.
GET /alerts{id}/applications	Retrieve all applications that have a specific alert assigned.
POST /alerts{id}/applications	Assign and unassign a specific alert to applications.
DELETE /alerts{id}/applications/{child-id}	Unassign an application from a specific alert.
POST /alerts{id}/applications/{child-id}	Assign an application to a specific alert.
GET /alerts/{id}/devicegroups	Retrieve all device groups that have a specific alert assigned.
POST /alerts/{id}/devicegroups	Assign and unassign a specific alert to device groups.
DELETE /alerts/{id}/devicegroups/{child-id}	Unassign a device group from a specific alert.

Operation	Description
POST /alerts/{id}/devicegroups/{child-id}	Assign a device group to a specific alert.
GET /alerts/{id}/devices	Retrieve all devices that have a specific alert assigned.
POST /alerts/{id}/devices	Assign and unassign a specific alert to devices.
DELETE /alerts/{id}/devices/{child-id}	Unassign a device from a specific alert.
POST /alerts/{id}/devices/{child-id}	Assign a device to a specific alert.
GET /alerts/{id}/emailgroups	Retrieve all email groups that have a specific alert assigned.
POST /alerts/{id}/emailgroups	Assign and unassign a specific alert to email groups.
DELETE /alerts/{id}/emailgroups/{child-id}	Unassign a email group from a specific alert.
POST /alerts/{id}/emailgroups/{child-id}	Assign a email group to a specific alert.
GET /alerts/{id}/exclusionintervals	Retrieve all exclustion intervals that have a specific alert assigned.
POST /alerts/{id}/exclusionintervals	Assign and unassign a specific alert to exclustion intervals.
DELETE /alerts/{id}/exclusionintervals/{child-id}	Unassign an exclusion interval from a specific alert.
POST /alerts/{id}/exclusionintervals/{child-id}	Assign an exclusion interval to a specific alert.
GET /alerts/{id}/networks	Retrieve all networks that have a specific alert assigned.
POST /alerts/{id}/networks	Assign and unassign a specific alert to networks.
DELETE /alerts/{id}/networks/{child-id}	Unassign a network from a specific alert.
POST /alerts/{id}/networks/{child-id}	Assign a network to a specific alert.
GET /alerts/{id}/stats	Retrieve all additional statistics for a specific alert.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Alert severity levels

The severity level you specify for an alert is displayed in the Alert History, email notifications, and SNMP traps.

The following severity levels are supported. Severity levels 0-2 require immediate attention.

Value	Name	Description
0	Emergency	System functionality is unavailable.
1	Alert	Immediate action is required.
2	Critical	Critical conditions are occurring.
3	Error	Error conditions are occurring.
4	Warning	Warning conditions are occurring.

Value	Name	Description
5	Notice	Normal operations are occurring with significant conditions, such as a restart.
6	Info	Normal operations are occurring with process updates.
7	Debug	Debug-level messages are available.

Analysis Priority

The ExtraHop system analyzes and classifies traffic for every device it discovers. Your license reserves capacity for the ExtraHop system to collect metrics for critical assets. This capacity is associated with two analysis levels: Advanced Analysis and Standard Analysis.

You can specify which devices receive Advanced Analysis and Standard Analysis levels by configuring analysis priority rules. Analysis priorities help inform the ExtraHop system about which devices are important in your environment. A third analysis level, Discovery Mode, is available for devices that are not in Advanced or Standard Analysis.

Note: By default, a Discover appliance manages its own analysis priorities for devices that it discovers. If the Discover appliance is connected to a Command appliance, you can transfer priority management to that Command appliance, which can help save time in a large deployment.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /analysispriority/config/{appliance_id}	Retrieve the analysis priority rules for a specific Discover appliance.
PUT /analysispriority/config/{appliance_id}	Replace the analysis priority rules for a specific Discover appliance.
GET /analysispriority/{appliance_id}/manager	Retrieve the appliance that manages analysis priority rules for a specific Discover appliance.
PATCH /analysispriority/{appliance_id}/manager	Update which appliance manages analysis priority rules for a specific Discover appliance.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

APIKey

An API key enables a user to perform operations through the ExtraHop REST API.

You can generate the initial API key for the setup user account through the REST API. All other API keys are generated through the API Access page in the ExtraHop Admin UI.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /apikeys	Retrieve all API keys.

Operation	Description
POST /apikeys	Create the initial API key for the setup user account.
GET /apikeys/{keyid}	Retrieve information about a specific API key.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Appliance

The ExtraHop system consists of a network of connected appliances that perform tasks such as monitoring traffic, analyzing data, storing data, and identifying detections.

You can retrieve information about ExtraHop appliances connected to the local appliance and establish new connections to remote ExtraHop appliances.

Note: You can only establish a connection to a remote ExtraHop appliance that is licensed for the same edition as the local ExtraHop appliance.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /appliances	Retrieve all remote ExtraHop appliances connected to the local appliance.
POST /appliances	Establish a new connection to a remote ExtraHop appliance.
GET /appliances/{id}	Retrieve a specific remote ExtraHop appliance connected to the local appliance.
GET /appliances/{id}/productkey	Retrieve the product key of the specified appliance.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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.

The following table displays all of the operations you can perform on the application resource:

Operation	Description
GET /applications	Retrieve all applications that were active within a specific timeframe.
POST /applications	Create a new application.
GET /applications/{id}	Retreive a specific application.
PATCH /applications/{id}	Update a specific application.
GET /applications/{id}/activity	Retrieve all activity for a specific application.

Operation	Description
GET /applications/{id}/alerts	Retrieve all alerts that are assigned to a specific application.
POST /applications/{id}/alerts	Assign and unassign alerts to a specific application.
DELETE /applications/{id}/alerts/{child-id}	Unassign an alert from a specific application.
POST /applications/{id}/alerts/{child-id}	Assign an alert to a specific application.
GET /applications/{id}/dashboards	Retrieve all dashboards related to a specific application.
GET /applications/{id}/pages	Retrieve all pages that are assigned to a specific application.
POST /applications/{id}/pages	Assign and unassign pages to a specific application.
DELETE /applications/{id}/pages/{child-id}	Unassign a page from a specific application.
POST /applications/{id}/pages/{child-id}	Assign a page to a specific application.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Audit log

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

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /auditlog	Retrieve all audit log messages.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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. For more information, see Bundles.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /bundles	Retrieve metadata about all bundles on the ExtraHop appliance.
POST /bundles	Upload a new bundle to the ExtraHop appliance.

Operation	Description
DELETE /bundles/{id}	Delete a specific bundle.
GET /bundles/{id}	Retrieve a specific bundle export.
POST /bundles/{id}/apply	Apply a saved bundle to the ExtraHop appliance.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /customdevices	Retrieve all custom devices.
POST /customdevices	Create a custom device.
DELETE /customdevices/{id}	Delete a specific custom device.
GET /customdevices/{id}	Retrieve a specific custom device.
PATCH /customdevices/{id}	Update a specific custom device.
GET /customdevices/{id}/criteria	Retrieve all criteria fro the specific custom device.
POST /customdevices/{id}/criteria	Create a new criterion for a specific custom device.
DELETE /customdevices/{id}/criteria/{child-id}	Delete a criterion for a specific custom device.
GET /customdevices/{id}/criteria/{child-id}	Retrieve a single custom device criterion.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Customization

The Customization resource enables you to manage backups files on the ExtraHop Discover or Command appliance. You must have unlimited privileges to perform operations on this resource.

Backup files contain both customizations and systems resources. Customizations are user-defined objects, such as alerts, dashboards, triggers, and custom metrics. System resources are items such as bundles, local users and groups, and the appliance SSL certificate. For more information, see Backup and Restore.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /customizations	Retrieve all backup files.
POST /customizations	Create a backup file.

Operation	Description
GET /customizations/status	Retrieve status details for the most recent backup attempt.
DELETE /customizations/{id}	Delete a specific backup file.
GET /customizations/{id}	Retrieve a specific backup file.
POST /customizations/{id}/apply	Restore only customizations from a specific backup file.
POST /customizations/{id}/download	Download a specific backup file.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Dashboards

Dashboards are built-in or customized views of your ExtraHop metrics information. For more information, see Dashboards.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /dashboards	Retrieve all dashboards.
DELETE /dashboards/{id}	Delete a specific dashboard.
GET /dashboards/{id}	Retrieve a specific dashboard.
PATCH /dashboards/{id}	Update ownership of a specific dashboard.
GET /dashboards/{id}/reports	Retrieve scheduled reports that contain a specific dashboard. **Important:** This operation is only available from an ExtraHop Command appliance.
GET /dashboards/{id}/sharing	Retrieve the users and their sharing permissions for a specific dashboard.
PATCH /dashboards/{id}/sharing	Update the users and their sharing permissions for a specific dashboard.
PUT /dashboards/{id}/sharing	Replace the users and their sharing permissions for a specific dashboard.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Device

Devices are objects on your network that have been identified and classified by your ExtraHop appliance. For more information, see Devices.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /devices	Retrieve all devices that were active within a specific time period.
GET /devices/{id}	Retrieve a specific device.
PATCH /devices/{id}	Update a specific device.
GET /devices/{id}/activity	Retrieve all activity for a device.
GET /devices/{id}/alerts	Retrieve all alerts that are assigned to a specific device.
POST /devices/{id}/alerts	Assign and unassign a specific device to alerts.
DELETE /devices/{id}/alerts/{child-id}	Unassign an alert from a specific device.
POST /devices/{id}/alerts/{child-id}	Assign an alert to a specific device.
GET /devices/{id}/dashboards	Retrieve all dashboards related to a specific device.
GET /devices/{id}/devicegroups	Retrieve all device groups that are assigned to a specific device.
POST /devices/{id}/devicegroups	Assign and unassign a specific device to device groups.
DELETE /devices/{id}/devicegroups/{child-id}	Unassign a device group from a specific device.
POST /devices/{id}/devicegroups/{child-id}	Assign a device group to a specific device.
GET /devices/{id}/pages	Retrieve all pages that are assigned to a specific device.
POST /devices/{id}/pages	Assign and unassign a specific device to pages.
DELETE /devices/{id}/pages/{child-id}	Unassign a page from a specific device.
POST /devices/{id}/pages/{child-id}	Assign a page to a specific device.
GET /devices/{id}/tags	Retrieve all tags that are assigned to a specific device.
POST /devices/{id}/tags	Assign and unassign a specific device to tags.
DELETE /devices/{id}/tags/{child-id}	Unassign a tag from a specific device.
POST /devices/{id}/tags/{child-id}	Assign a tag to a specific device.
GET /devices/{id}/triggers	Retrieve all triggers that are assigned to a specific device.
POST /devices/{id}/triggers	Assign and unassign a specific device to triggers.
DELETE /devices/{id}/triggers/{child-id}	Unassign a trigger from a specific device.
POST /devices/{id}/triggers/{child-id}	Assign a trigger to a specific device.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Supported time units

For most parameters, the default unit for time measurement is milliseconds. However, the following parameters return or accept alternative time units such as minutes and hours:

- Device
  - active_from
  - active_until
- Device group
  - active_from
  - active_until
- Metrics
  - from
  - until
- Record Log
  - from
  - until
  - context_ttl

The following table displays supported time units:

Time unit	Unit suffix
Year	`y`
Month	`M`
Week	`w`
Day	`d`
Hour	`h`
Minute	`m`
Second	`s`
Millisecond	`ms`

To specify a time unit other than milliseconds for a parameter, append the unit suffix to the value. For example, to request devices active in the last 30 minutes, specify the following parameter value:

```
GET /api/v1/devices?active_from=-30m
```

The following example specifies a search for HTTP records created between 1 and 2 hours ago:

```
{
    "from": "-2h",
    "until": "-1h",
    "types": ["~http"]
}
```

Device group

Device groups can be either static or dynamic.

A static device group is user-defined; you create a device 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 device group and then set a rule to classify all devices within a certain IP address range to be added to that group automatically. For more information, see Device groups.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /devicegroups	Retrieve all device groups that were active within a specific time period.
POST /devicegroups	Create a new device group.
DELETE /devicegroups/{id}	Delete a device group.
GET /devicegroups/{id}	Retrieve a specific device group.
PATCH /devicegroups/{id}	Update a specific device group.
GET /devicegroups/{id}/alerts	Retrieve all alerts that are assigned to a specific device group.
POST /devicegroups/{id}/alerts	Assign and unassign a specific device group to alerts.
DELETE /devicegroups/{id}/alerts/{child-id}	Unassign an alert from a specific device group.
POST /devicegroups/{id}/alerts/{child-id}	Assign an alert to a specific device group.
GET /devicegroups/{id}/dashboards	Retrieve all dashboards related to a specific device group.
GET /devicegroups/{id}/devices	Retrieve all devices in the device group that are active within a specific time window.
POST /devicegroups/{id}/devices	Assign and unassign a devices to a specific static device group.
DELETE /devicegroups/{id}/devices/{child-id}	Unassign a device from a specific static device group.
POST /devicegroups/{id}/devices/{child-id}	Assign a device to a specific static device group.
GET /devicegroups/{id}/pages	Retrieve all pages that are assigned to a specific device group.
POST /devicegroups/{id}/pages	Assign and unassign a specific device to pages group.
DELETE /devicegroups/{id}/pages/{child-id}	Unassign a page from a specific device group.
POST /devicegroups/{id}/pages/{child-id}	Assign a page to a specific device group.
GET /devicegroups/{id}/triggers	Retrieve all triggers that are assigned to a specific device group.
POST /devicegroups/{id}/triggers	Assign and unassign a specific device group to triggers.
DELETE /devicegroups/{id}/triggers/{child-id}	Unassign a trigger from a specific device group.
POST /devicegroups/{id}/triggers/{child-id}	Assign a trigger to a specific device group.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Supported time units

For most parameters, the default unit for time measurement is milliseconds. However, the following parameters return or accept alternative time units such as minutes and hours:

- Device
  - active_from
  - active_until
- Device group
  - active_from
  - active_until
- Metrics
  - from
  - until
- Record Log
  - from
  - until
  - context_ttl

The following table displays supported time units:

Time unit	Unit suffix
Year	`y`
Month	`M`
Week	`w`
Day	`d`
Hour	`h`
Minute	`m`
Second	`s`
Millisecond	`ms`

To specify a time unit other than milliseconds for a parameter, append the unit suffix to the value. For example, to request devices active in the last 30 minutes, specify the following parameter value:

```
GET /api/v1/devices?active_from=-30m
```

The following example specifies a search for HTTP records created between 1 and 2 hours ago:

```
{
    "from": "-2h",
    "until": "-1h",
    "types": ["~http"]
}
```

Detections

The Detections class enables you to retrieve detections that have been identified by your appliance.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /detections	Retrieve all detections.
POST /detections/search	Retrieve detections that match the specified search criteria.
GET /detections/{id}	Retrieve a specific detection.
PATCH /detections/{id}	Update a detection.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /emailgroups	Retrieve all email groups.
POST /emailgroups	Create a new email group.
DELETE /emailgroups/{id}	Delete a email group by a unique identifier.
GET /emailgroups/{id}	Retrieve a specific email group by a unique identifier.
PATCH /emailgroups/{id}	Apply updates to a specific email group.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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. For more information, see Alerts.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /exclusionintervals	Retrieve all exclusion intervals.
POST /exclusionintervals	Create a new exclusion interval.
DELETE /exclusionintervals/{id}	Delete a specific exclusion interval.
GET /exclusionintervals/{id}	Retrieve a specific exclusion interval.
PATCH /exclusionintervals/{id}	Apply updates to a specific exclusion interval.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

ExtraHop

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

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /extrahop	Retrieve metadata about the firmware running on the ExtraHop appliance.
GET /extrahop/idrac	Retrieve the iDRAC IP address of the ExtraHop appliance.
GET /extrahop/platform	Retrieve the platform name of the ExtraHop appliance.
GET /extrahop/processes	Retrieve a list of processes running on the ExtraHop appliance.
POST/extrahop/processes/{process}/restart	Restart a process running on the ExtraHop appliance.
POST /extrahop/sslcert	Regenerate the SSL certificate on the ExtraHop appliance.
PUT /extrahop/sslcert	Replace the SSL certificate on the ExtraHop appliance.
POST /extrahop/sslcert/signingrequest	Create an SSL certificate signing request
GET /extrahop/ticketing	Retrieve the ticketing integration status.
PATCH /extrahop/ticketing	Enable or disable ticketing integration.
GET /extrahop/version	Retrieve the version of the firmware running on the ExtraHop appliance.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

License

This resource enables you to retrieve and set product keys or to retrieve and set a license.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /license	Retrieve the license applied to this ExtraHop appliance.
PUT /license	Apply and register a new license to the ExtraHop appliance.
GET /license/productkey	Retrieve the product key to this ExtraHop appliance.

Operation	Description
PUT /license/productkey	Apply the specified product key to the ExtraHop appliance and register the license.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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. For more information, see Metrics.

The following table displays all of the operations you can perform on this resource:

Operation	Description
POST /metrics	Perform a metric query.
GET /metrics/next/{xid}	Retrieve the next chunked results from a metrics query request. This request is only valid on a Command appliance.
POST /metrics/total	Perform a metric query for total values.
POST /metrics/totalbyobject	Perform a metric query for total values that are grouped by object.

For example, if you want to see all HTTP responses that occurred on the network in the last 30 seconds, enter the following request schema into the `POST /metrics` operation:

```
{
  "cycle": "auto",
  "from": -1800000,
  "metric_category": "http",
  "metric_specs": [
    {
      "name": "rsp"
    }
  ],
  "object_ids": [
    0
  ],
  "object_type": "application",
  "until": 0
}
```

The response body returns a list of HTTP responses and the time of each event, similar to the following output:

```
{
  "stats": [
    {
      "oid": 0,
      "time": 1494539640000,
      "duration": 30000,
      "values": [
        354
```

```
      ]
    },
    {
      "oid": 0,
      "time": 1494539640000,
      "duration": 30000,
      "values": [
        354
      ]
    },

    {
      "oid": 0,
      "time": 1494539640000,
      "duration": 30000,
      "values": [
        354
      ]
    },
  ],
  "cycle": "30sec",
  "node_id": 0,
  "clock": 1494541440000,
  "from": 1494539640000,
  "until": 1494541440000
}
```

Enter the same request schema into the **POST /metrics/total** operation to retrieve a count of all HTTP responses that occurred on the network in the last 30 seconds. The response body is similar to the following output:

```
{
  "stats": [
    {
      "oid": -1,
      "time": 1494541380000,
      "duration": 1800000,
      "values": [
        33357
      ]
    }
  ],
  "cycle": "30sec",
  "node_id": 0,
  "clock": 1494541440000,
  "from": 1494539640000,
  "until": 1494541440000
}
```

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Supported time units

For most parameters, the default unit for time measurement is milliseconds. However, the following parameters return or accept alternative time units such as minutes and hours:

- Device
  - active_from
  - active_until

- Device group
  - active_from
  - active_until
- Metrics
  - from
  - until
- Record Log
  - from
  - until
  - context_ttl

The following table displays supported time units:

Time unit	Unit suffix
Year	`y`
Month	`M`
Week	`w`
Day	`d`
Hour	`h`
Minute	`m`
Second	`s`
Millisecond	`ms`

To specify a time unit other than milliseconds for a parameter, append the unit suffix to the value. For example, to request devices active in the last 30 minutes, specify the following parameter value:

```
GET /api/v1/devices?active_from=-30m
```

The following example specifies a search for HTTP records created between 1 and 2 hours ago:

```
{
    "from": "-2h",
    "until": "-1h",
    "types": ["~http"]
}
```

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 connected appliance is identified as a network capture that is looking at the traffic for each ExtraHop Discover appliance that is connected to the Command appliance. For more information, see Networks.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /networks	Retrieve all networks.

Operation	Description
GET /networks/{id}	Retreive a specific network by ID.
PATCH /networks/{id}	Update a specific network by ID.
GET /networks/{id}/alerts	Retrieve all alerts that are assigned to a specific network.
POST /networks/{id}/alerts	Assign and unassign alerts to a specific network.
DELETE /networks/{id}/alerts/{child-id}	Unassign an alert from a specific network.
POST /networks/{id}/alerts/{child-id}	Assign an alert to a specific network.
GET /networks/{id}/pages	Retrieve all pages that are assigned to a specific network.
POST /networks/{id}/pages	Assign and unassign pages to a specific network.
DELETE /networks/{id}/pages/{child-id}	Unassign a page from a specific network.
POST /networks/{id}/pages/{child-id}	Assign a page to a specific network.
GET /networks/{id}/vlans	Retrieve all VLANS assigned to a specific network.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Network locality entry

You can manage a list that specifies the network locality of IP addresses.

For example, you can create an entry in the network locality list that specifies that an IP address or CIDR block is internal or external.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /networklocalities	Retrieve all network locality entries.
POST /networklocalities	Create a network locality entry.
DELETE /networklocalities/{id}	Delete a network locality entry.
GET /networklocalities/{id}	Retrieve a specific network locality entry.
PATCH /networklocalities/{id}	Apply updates to a specific network locality entry.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /nodes	Retrieve all Discover nodes connected to this Command appliance.
GET /nodes/{id}	Retrieve a specific Discover node that is connected to this Command appliance.
PATCH /nodes/{id}	Update a specific Discover node that is connected to this Command appliance.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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. For more information, see Packets.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /packetcaptures	Retrieve metadata about all packet captures stored on this ExtraHop appliance.
DELETE /packetcaptures/{id}	Permanently remove a specific packet capture from the ExtraHop appliance.
GET /packetcaptures/{id}	Download a specific packet capture in PCAP format.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Page

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

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /pages	Retrieve all pages.
POST /pages	Create a page.
DELETE /pages/{id}	Delete a single page.
GET /pages/{id}	Retrieve a single page.
PATCH /pages/{id}	Update a single page.
GET /pages/{id}/applications	Retrieve all applications that have a specific page assigned.
POST /pages/{id}/applications	Assign and unassign a specific page to applications.

Operation	Description
DELETE /pages/{id}/applications/{child-id}	Unassign an application from a specific page.
POST /pages/{id}/applications/{child-id}	Assign an application to a specific page.
GET /pages/{id}/devicegroups	Retrieve all device groups that are assigned to a specific page.
POST /pages/{id}/devicegroups	Assign and unassign a specific page to device groups.
DELETE /pages/{id}/devicegroups/{child-id}	Unassign a device group from a specific page.
POST /pages/{id}/devicegroups/{child-id}	Assign a device group to a specific page.
GET /pages/{id}/devices	Retrieve all devices that have a specific page assigned.
POST /pages/{id}/devices	Assign and unassign a specific page to devices.
DELETE /pages/{id}/devices/{child-id}	Unassign a device from a specific page.
POST /pages/{id}/devices/{child-id}	Assign a device to a specific page.
GET /pages/{id}/networks	Retrieve all networks that have a specific page assigned.
POST /pages/{id}/networks	Assign and unassign a specific page to networks.
DELETE /pages/{id}/networks/{child-id}	Unassign a network from a specific page.
POST /pages/{id}/networks/{child-id}	Assign a network to a specific page.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Record Log

Records are structured flow and transaction information about events on your network.

After you connect an ExtraHop Discover appliance to an ExtraHop Explore appliance, you can generate and send record information to the Explore appliance for storage, and you can query records to retrieve stored information about any object on your network. For more information, see Records.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /records/cursor/{cursor}	Deprecated. Replaced by `POST /records/cursor`.
POST /records/cursor	Retrieve records starting at a specified cursor.
POST /records/search	Perform a record log query.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Operand values in record queries

The `operand` field in the `POST /records/search` method specifies the value that a record query attempts to match. You can specify either the value only or both the data type and the value. If you specify

only the value, the query will refer to the record format associated with the `field` parameter to determine the data type of the value.

For example, if you want to search for an IP address, you can specify an IP address data type, and then provide the actual address as the value.

The following example explicitly specifies the data type and value of the operand:

```
{
    "from": -1000,
    "filter": {
        "field" : "senderAddr",
        "operator": "=",
        "operand" : { "type" : "ipaddr4", "value": "1.2.3.4" }
    }
}
```

The following example specifies only the value of the operand:

```
{
    "from": -1000,
    "filter": {
        "field" : "senderAddr",
        "operator": "=",
        "operand" : "1.2.3.4"
    }
}
```

You can explicitly specify the following data types in the `operand` field:

- ipaddr4
- ipaddr6
- device
- application
- string
- number
- boolean

The `operand` field supports CIDR notation when filtering by IP addresses; the `operator` field must be set to "=" or "!=".

Supported time units

For most parameters, the default unit for time measurement is milliseconds. However, the following parameters return or accept alternative time units such as minutes and hours:

- Device
  - active_from
  - active_until
- Device group
  - active_from
  - active_until
- Metrics
  - from
  - until
- Record Log
  - from

- until
- context_ttl

The following table displays supported time units:

Time unit	Unit suffix
Year	`y`
Month	`M`
Week	`w`
Day	`d`
Hour	`h`
Minute	`m`
Second	`s`
Millisecond	`ms`

To specify a time unit other than milliseconds for a parameter, append the unit suffix to the value. For example, to request devices active in the last 30 minutes, specify the following parameter value:

```
GET /api/v1/devices?active_from=-30m
```

The following example specifies a search for HTTP records created between 1 and 2 hours ago:

```
{
    "from": "-2h",
    "until": "-1h",
    "types": ["~http"]
}
```

Report

A report is a PDF file of a dashboard that you can schedule for email delivery to one or more recipients. You can specify how often the report email is delivered and the time interval for dashboard data included in the PDF file.

Important: You can only schedule reports from an ExtraHop Command appliance.

Here are some important considerations about scheduled reports:

- You can only create a report for dashboards that you own or have been shared with you. Your ability to create a report is determined by your user privileges. Contact your ExtraHop administrator for help.
- Each report can only link to one dashboard.
- If you created a report for a dashboard that was later deleted or became inaccessible to you, the scheduled email will continue to be sent to recipients. However, the email will not include the PDF file and will instead notify recipients that the dashboard is unavailable to the report owner.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /reports	Retrieve all reports.
POST /reports	Create a report.

Operation	Description
DELETE /reports/{id}	Delete a specific report.
GET /reports/{id}	Retrieve a specific report.
PATCH /reports/{id}	Update a specific report.
GET /reports/{id}/contents	Retrieve the contents of a specific report.
PUT /reports/{id}/contents	Replace the contents of a specific report.
POST /reports/{id}/emailgroups	Change the email group assigned to a specific scheduled report.
GET /reports/{id}/emailgroups	Retrieve a list of email groups assigned to a specific scheduled report.
DELETE /reports/{id}emailgroups/{group-id}	Remove an email group from a specific scheduled report.
POST /reports/{id}emailgroups/{group-id}	Add an email group to a specific scheduled report.
POST /reports/{id}/queue	Immediately generate and send a specific report.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Running config

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

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /runningconfig	Retrieve the current running configuration file.
PUT /runningconfig	Replace the current running configuration file. Configuration file changes are not automatically saved.
POST /runningconfig/save	Save the current changes to the running configuration file.
GET /runningconfig/saved	Retrieve the saved running configuration file.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

SSL decrypt key

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

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /ssldecryptkeys	Retrieve all SSL decryption keys.
POST /ssldecryptkeys	Create a new SSL decryption key.
DELETE /ssldecryptkeys/{id}	Remove an SSL key from the ExtraHop appliance.
GET /ssldecryptkeys/{id}	Retrieve an SSL PEM and metadata.
PATCH /ssldecryptkeys/{id}	Update an existing SSL decryption key.
GET /ssldecryptkeys/{id}/protocols	Retrieve all protocols assigned to an SSL decryption key.
POST /ssldecryptkeys/{id}/protocols	Create a new protocol for an SSL decryption key.
DELETE /ssldecryptkeys/{id}/protocols/{child-id}	Delete a protocol from an SSL decryption key.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Support pack

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

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /supportpacks	Retrieve metadata about all support packs.
POST /supportpacks/execute	Run a new support pack.
GET /supportpacks/queue/{id}	Check on the status of an in-progress, running support pack.
GET /supportpacks/{filename}	Downlolad an existing support pack by filename.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /tags	Retrieve all tags.
POST /tags	Create a a new tag.
DELETE /tags/{id}	Delete a specific tag.
GET /tags/{id}	Retrieve a specific tag.
PATCH /tags/{id}	Apply updates to a specific tag.

Operation	Description
GET /tags/{id}/devices	Retrieve all devices that are assigned to a specific tag.
POST /tags/{id}/devices	Assign and unassign a specific tag to devices.
DELETE /tags/{id}/devices/{child-id}	Unassign a device from a specific tag.
POST /tags/{id}/devices/{child-id}	Assign a device to a specific tag.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Threat Collection

The Threat Collection class enables you to create threat collections by uploading Structured Threat Information eXpression (STIX) files to your appliance. You must first obtain STIX files from a TAXII server or threat intelligence platform, and then upload those STIX files to ExtraHop Reveal(x).

Note: This topic applies only to ExtraHop Reveal(x) Premium and Ultra.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /threatcollections	Retrieve all threat collections.
DELETE /threatcollections/{id}	Delete a threat collection.
PUT /threatcollections/{id}	Upload a new threat collection. ExtraHop currently supports STIX versions 1.0 - 1.2. **Note:** If a threat collection with the same name already exists on the appliance, the existing threat collection is overwritten.
GET /threatcollections/{id}/observables	Retrieve the number of STIX observables loaded from a threat collection.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

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. For supplemental implementation notes about advanced options, see Advanced trigger options.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /triggers	Retrieve all triggers.
POST /triggers	Create a new trigger.

Operation	Description
DELETE /triggers/{id}	Delete a specific identifier.
GET /triggers/{id}	Retrieve a specific trigger by unique identifier.
PATCH /triggers/{id}	Update an existing trigger.
GET /triggers/{id}/devicegroups	Retrieve all device groups that are assigned to a specific trigger.
POST /triggers/{id}/devicegroups	Assign and unassign a specific trigger to device groups.
DELETE /triggers/{id}/devicegroups/{child-id}	Unassign a device group from a specific trigger.
POST /triggers/{id}/devicegroups/{child-id}	Assign a device group to a specific trigger.
GET /triggers/{id}/devices	Retrieve all devices that are assigned to a specific trigger.
POST /triggers/{id}/devices	Assign and unassign a specific trigger to devices.
DELETE /triggers/{id}/devices/{child-id}	Unassign a device from a specific trigger.
POST /triggers/{id}/devices/{child-id}	Assign a device to a specific trigger.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Advanced trigger options

Advanced trigger options are configuration options that you can set depending on the system events associated with the trigger. For example, you can configure the number of payload bytes to buffer on HTTP request events.

Advanced options are contained in the `hints` object of the trigger resource as shown in the following example:

```
"hints": {
"flowClientPortMin": null,
"flowClientBytes": 16384,
"flowClientPortMax": null,
"flowServerBytes": 16384,
"flowPayloadTurn": true,
"flowServerPortMin": 135,
"flowServerPortMax": 49155
}
```

The following table describes available advanced options and applicable events:

Option	Description	Applicable events
`"snaplen": number`	Specifies the number of bytes to capture per packet. Capture starts with the first bytes in the packet. Specify this option only if the trigger script performs packet capture.	All events except: • ALERT_RECORD_COMMIT • METRIC_CYCLE_BEGIN • METRIC_CYCLE_END • FLOW_REPORT • NEW_APPLICATION • NEW_DEVICE

Option	Description	Applicable events
	A value of `0` specifies that the capture should collect all bytes in each packet.	• SESSION_EXPIRE
`"payloadBytes": number`	Specifies the minimum number of payload bytes to buffer.	• CIFS_REQUEST • CIFS_RESPONSE • HTTP_REQUEST • HTTP_RESPONSE • ICA_TICK
`"clipboardBytes": number`	Specifies the number of bytes to buffer on a Citrix clipboard transfer.	• ICA_TICK
`"cycle": [30sec, 5min, 1hr, 24hr]`	Specifies the length of the metric cycle, expressed in seconds.	• METRIC_CYCLE_BEGIN • METRIC_CYCLE_END • METRIC_RECORD_COMMIT
`"metricTypes": string`	Specifies the metric type by the raw metric name such as extrahop.device.http_server.	• ALERT_RECORD_COMMIT • METRIC_RECORD_COMMIT
`"flowPayloadTurn": boolean`	Enables packet capture on each flow turn. Per-turn analysis continuously analyzes communication between two endpoints to extract a single payload data point from the flow. If this option is enabled, any values specified for the `flowClientString` and `flowServerString` options are ignored.	• SSL_PAYLOAD • TCP_PAYLOAD
`"flowClientPortMin": number`	Specifies the minimum port number of the client port range. Valid values are `0` to `65535`. A value of `0` specifies matching of any port.	• SSL_PAYLOAD • TCP_PAYLOAD • UDP_PAYLOAD
`"flowClientPortMax": number`	Specifies the maximum port number of the client port range. Valid values are `0` to `65535`. Any value specified for this option is ignored if the value of the `flowClientPortMin` option is `0`.	• SSL_PAYLOAD • TCP_PAYLOAD • UDP_PAYLOAD
`"flowClientBytes": number`	Specifies the number of client bytes to buffer. The value of this option cannot be set to `0` if the value of the	• SSL_PAYLOAD • TCP_PAYLOAD

Option	Description	Applicable events
	`flowServerBytes` option is also set to `0`.	
`"flowClientString": string`	Specifies the format string of client data to process. Any value specified for this option is ignored if the `flowPayloadTurn` option is enabled.	• SSL_PAYLOAD • TCP_PAYLOAD • UDP_PAYLOAD
`"flowServerPortMin": number`	Specifies the minimum port number of the server port range. Valid values are `0` to 65535. A value of `0` specifies matching of any port.	• SSL_PAYLOAD • TCP_PAYLOAD • UDP_PAYLOAD
`"flowServerPortMax": number`	Specifies the maximum port number of the server port range. Valid values are `0` to `65535`. Any value specified for this option is ignored if the value of the `flowServerPortMin` option is `0`.	• SSL_PAYLOAD • TCP_PAYLOAD • UDP_PAYLOAD
`"flowServerBytes": number`	Specifies the number of server bytes to buffer. The value of this option cannot be set to `0` if the value of the `flowClientBytes` option is also set to `0`.	• SSL_PAYLOAD • TCP_PAYLOAD
`"flowServerString": string`	Specifies the format string of server data to process. Returns the entire packet upon a string match. Any value specified for this option is ignored if the `flowPayloadTurn` option is enabled.	• SSL_PAYLOAD • TCP_PAYLOAD • UDP_PAYLOAD
`"flowUdpAll": boolean`	Enables capture of all UDP datagrams.	• UDP_PAYLOAD
`"fireClassifyOnExpiration": boolean`	Enables running the event upon expiration in order to accumulate metrics for flows that were not classified before expiring.	• FLOW_CLASSIFY

User

The user resource enables you to create and manage the list of users who have access to the ExtraHop appliance and the privilege levels for those users.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /users	Retrieve all users.
POST /users	Create a new user.
DELETE /users/{username}	Delete a specific user.
GET /users/{username}	Retrieve a specific user.
PATCH /users/{username}	Update settings for a specific user.
GET /users/{username}/apikeys	Retrieve all API keys for a specific user.
GET /users/{username}/apikeys/{keyid}	Retrieve information about a specific API key and user.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

User group

The user group resource enables you to manage and update groups of users and their dashboard sharing associations.

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /usergroups	Retrieve all user groups.
POST /usergroups/refresh	Query LDAP for the most recent user memberships for all remote user groups.
GET /usergrups/{id}	Retrieve a specific user group.
PATCH /usergroups/{id}	Update a specific user group.
DELETE /usergroups/{id}/associations	Delete all dashboard sharing associations with a specific user group.
GET /usergroups/{id}/members	Retrieve all members of a specific user group.
POST /usergroups/{id}/refresh	Query LDAP for the most recent user membership of a specific remote user group.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

VLAN

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

The following table displays all of the operations you can perform on this resource:

Operation	Description
GET /vlans	Retrieve all VLANs

Operation	Description
GET /vlans/{id}	Retrieve a specific VLAN.
PATCH /vlans/{id}	Update a specific VLAN.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

Whitelist (Watchlist)

To guarantee that a critical asset, such as an important server, database, or laptop, is guaranteed Advanced Analysis, you can add that device to the whitelist, which is referred to as the watchlist in the Web UI.

Tip: If you want to add several devices to the whitelist, consider creating a device group and then prioritizing that group for Advanced Analysis.

Here are important considerations about the whitelist:

- The whitelist only applies to Advanced Analysis.
- The whitelist can contain as many devices as allowed by the Advanced Analysis capacity, which is determined by your license.
- A device stays on the whitelist whether it is inactive or active. A device has to be active for the ExtraHop system to collect Advanced Analysis metrics.

For more information about Advanced Analysis, see Analysis levels.

The following table displays all of the operations you can perform on this resource:

Operation	Description
DELETE /whitelist/device/{id}	Remove a device from the whitelist.
POST /whitelist/device/{id}	Add a device to the whitelist.
GET /whitelist/devices	Retrieve all devices that are in the whitelist.
POST /whitelist/devices	Add or remove devices from the whitelist.

Implementation information and instructions for each operation are documented in the REST API Explorer. You can click on any operation in the REST API Explorer to view implementation information such as parameters, response class and messages, and JSON model and schema.

ExtraHop REST API examples

The following examples are available:

- Change a dashboard owner through the REST API
- Extract the device list through the REST API
- Create and assign a device tag through the REST API
- Query for metrics about a specific device through the REST API
- Create, retrieve, and delete an object through the REST API
- Query the record log

Change a dashboard owner through the REST API

Dashboards are owned by the logged in user that created them. If a user is no longer with your company, you might need to change the owner of the dashboard to maintain that dashboard.

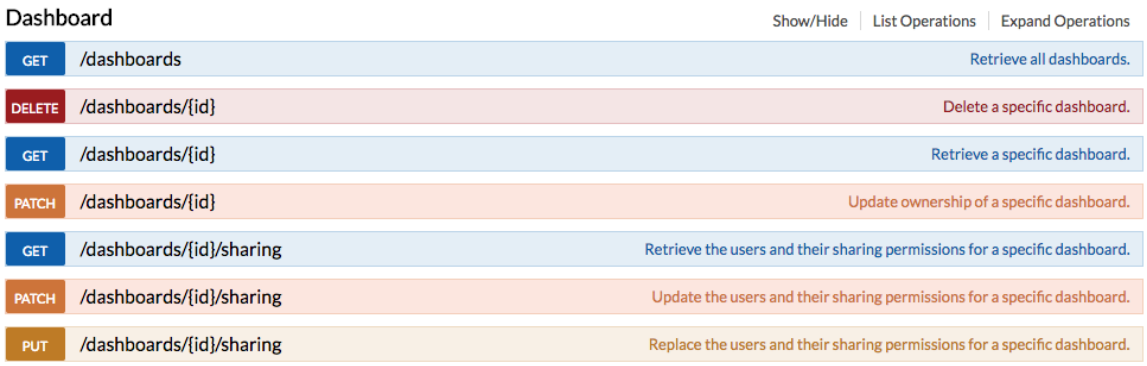
To transfer ownership of a dashboard, you need the dashboard ID and the username of the dashboard owner. You can only view the username of the owner of a dashboard through the REST API.

Before you begin

- You must log into the ExtraHop appliance with an account that has unlimited 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 REST API Explorer.

Retrieve the dashboard IDs

1. In a browser, navigate to the REST API Explorer.
   The URL is the hostname or IP address of your ExtraHop Discover or Command appliance, followed by `/api/v1/explore/`. For example, if your hostname is seattle-eda, the URL is `https://seattle-eda/api/v1/explore/`.
2. Paste or type your API Key into the **api_key** field at the top of the page.
3. Click **Dashboard** to display dashboard operations.



4. Click **GET /dashboards**.
5. Click **Try it out!** to send the request to your appliance.
   The request returns a response body with information about each dashboard.
6. Search for the dashboards by the dashboard name or by the user account listed in the `"owner"` field. If your list of dashboards is long, you can press control-F and search the response body.

For our example, we want to change the `"LDAP Server Health"` dashboard created by the user account for `"marksmith"`:

```
{
    "id": 1876,
    "comment": null,
    "mod_time": 1507576983922,
    "author": "Mark Smith",
    "name": "LDAP Server Health",
    "owner": "marksmith",
    "built-in": false,
    "short_code": "MpXgk",
    "rights": [
      "transfer",
      "view",
      "edit",
      "share",
      "delete"
    ]
  }
```

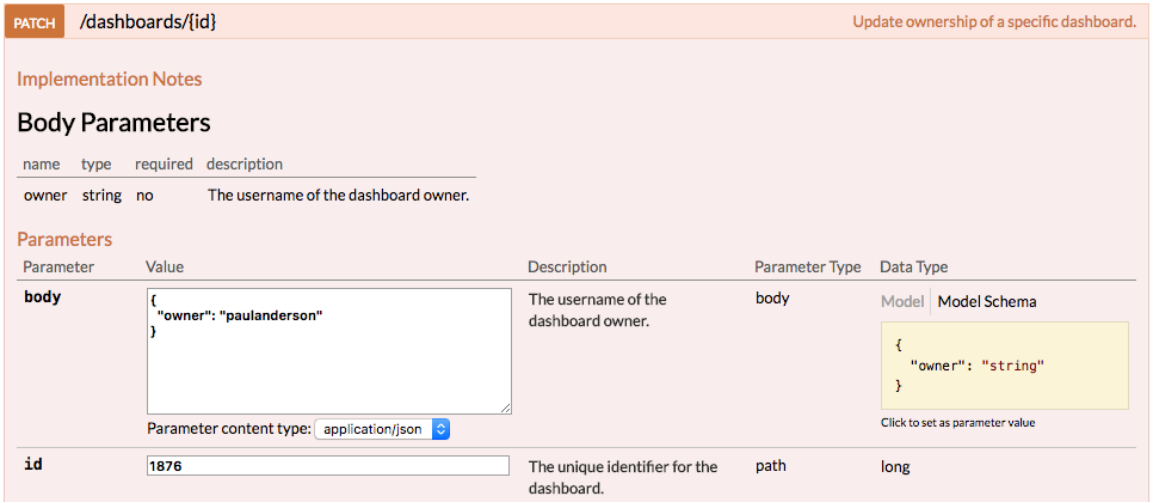
7. Note the number in the `"id"` field for each dashboard you want to modify.

Change the dashboard owner

1. Scroll down the page of Dashboard operations to the /dashboards/{id} section.
2. Click **PATCH /dashboards/{id}**.
3. In the Parameters section, click **Model Schema**, and then click in the box to automatically add the JSON schema to the **body** parameter text box.
4. In the `"owner"` field, replace `string` with the username of the new owner.
5. In the **id** field, type the number you previously noted for the dashboard.

   For our example, this value is `1876`. (You can only modify one dashboard at a time through the REST API Explorer.)

   In the following figure, we added the JSON `"string"` for the `"owner"` parameter to the body parameter text box, changed `"string"` to `"paulanderson"`, and typed `"1876"` in the id field.



6. Click **Try it out!** to send the request to your appliance.

   Warning: The **Try it out!** button sends the request to your appliance and can make permanent changes.

The Response Code field displays `204` if the operation is successful. You can click **GET /dashboards** again to verify that the `"owner"` field has changed. Note that you can only change the dashboard owner. You cannot change the dashboard name or author fields through the REST API.

```
{
    "id": 1876,
    "comment": null,
    "mod_time": 1507576983922,
    "author": "Mark Smith",
    "name": "LDAP Server Health",
    "owner": "paulanderson",
    "built-in": false,
    "short_code": "MpXgk",
    "rights": [
      "transfer",
      "view",
      "edit",
      "share",
      "delete"
    ]
  }
```

The dashboard is now available under **My Dashboards** in the ExtraHop Web UI for the new user. As the new owner, you can now log into your ExtraHop appliance and change other dashboard properties, such as the dashboard name or author.

Tip: If you want to permanently delete the dashboard, instead of changing the owner, click **DELETE /dashboards/{id}**, type the value in the **id** field, and then click **Try it out!**.

Tip: After you click **Try it out!**, the REST API Explorer provides scripts for the operation in Curl, Python 2.7, or Ruby.

Python Script Example

The following example script searches for all dashboards owned by the user account `marksmith` on an ExtraHop appliance with the hostname `example.extrahop.com` and then changes the owner for all of those dashboards to the user account `paulanderson`.

```
import httplib
import json

HOST = 'example.extrahop.com'
APIKEY = 'f6876657888a7c1f24ac77827'

headers = {'Accept': 'application/json',
           'Authorization': 'ExtraHop apikey=%s' % APIKEY}
conn = httplib.HTTPSConnection(HOST)
conn.request('GET', '/api/v1/dashboards', headers=headers)
resp = conn.getresponse()
parsed_resp = json.loads(resp.read())

for dashboard in parsed_resp:
    if dashboard['owner'] == 'marksmith':
        print('Dashboard {id} owned by marksmith.'
              ' Switching ownership...'.format(id=dashboard['id']))
        config = {'owner': 'paulanderson'}
        conn.request('PATCH', '/api/v1/dashboards/{id}'.format(
                id=dashboard['id']), json.dumps(config), headers=headers)
        resp = conn.getresponse()
        resp.read()
```

Extract the device list through the REST API

You can extract the list of devices being monitored by an ExtraHop appliance through the ExtraHop REST API. If you extract the list through the REST API with a script, you can export the list in a format that can be read by third-party CMDB applications. In this topic, we will demonstrate methods for both the REST API and the REST API Explorer.

Before you begin

- You must log into the ExtraHop appliance with an account that has unlimited 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 REST API Explorer.

Extract the device list through the REST API Explorer

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

   The URL is the hostname or IP address of your ExtraHop Discover or Command appliance, followed by `/api/v1/explore/`. For example, if your hostname is seattle-eda, the URL is `https://seattle-eda/api/v1/explore/`.
2. Paste or type your API Key into the `api_key` field at the top of the page.
3. Click **Device** and then click **GET/devices**.
4. In the limit field, set the maximum number of devices you want to include in your list.
5. Click **Try it out!**.

   The Response Body displays the device list in JSON format.

Python script example

The following example Python script extracts the device list from an ExtraHop appliance and writes the list to a csv file that can be read by Microsoft Excel. The script includes the following configuration variables:

- **HOST:** The IP address or hostname of the Discover appliance
- **APIKEY:** The API key
- **FILENAME:** The file that output will be written to
- **MAXDEVICES:** The maximum number of devices to extract
- **SAVEL2:** Determines whether L2 devices are included

```
import httplib
import json
import httplib
import json
import re
import csv
import datetime

HOST = 'example.extrahop.com'
APIKEY = "f6876657888a7c1f24ac77827"
FILENAME = "devices.csv"
MAXDEVICES = 250
SAVEL2 = False

headers = {}
headers['Accept'] = 'application/json'
headers['Authorization'] = 'ExtraHop apikey='+APIKEY
```

```
conn = httplib.HTTPSConnection(HOST)
conn.request('GET', '/api/v1/devices?limit=%d&offset=
%d&search_type=any'%(MAXDEVICES,0), headers=headers)
resp = conn.getresponse()
if resp.status == 200:
    dTable = json.loads(resp.read())
    conn.close()
else:
    print "Error retrieving Device list"
    print resp.status, resp.reason
    resp.read()
    dTable = None
    conn.close()

if (dTable != None):
    print " - Saving %d devices in CSV file" % len(dTable)
    with open(FILENAME, 'w') as csvfile:
        csvwriter = csv.writer(csvfile,dialect='excel')
        csvwriter.writerow(dTable[0].keys())
        w = 0
        s = 0
        for d in dTable:
            if d['is_l3'] | SAVEL2:
                w += 1
                d['mod_time'] =
 datetime.datetime.fromtimestamp(d['mod_time']/1000.0)
                d['user_mod_time'] =
 datetime.datetime.fromtimestamp(d['user_mod_time']/1000.0)
                d['discover_time'] =
 datetime.datetime.fromtimestamp(d['discover_time']/1000.0)
                csvwriter.writerow(d.values())
            else:
                s += 1
        print " - Wrote %d devices, skipped %d devices " % (w,s)
```

Create a trusted SSL certificate through the REST API

By default, ExtraHop appliances include a self-signed SSL certificate. However, you can improve security for your appliance by adding a trusted certificate signed by a certification authority (CA). You can create the certificate signing request to send to your CA through the ExtraHop REST API. After you receive the signed certificate, you can also add it to your ExtraHop appliance through the REST API.

Before you begin

- You must log into the ExtraHop appliance with an account that has unlimited 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 REST API Explorer.

Note: You can also perform the procedures in this topic through the ExtraHop Admin UI. For more information, see the following topics:

- Create a certificate signing request from your ExtraHop appliance
- Add a trusted certificate to your ExtraHop appliance

Create an SSL certificate signing request

To create a signed SSL certificate, you must send a certificate signing request to a trusted CA.

1. In a browser, navigate to the REST API Explorer.
   The URL is the hostname or IP address of your ExtraHop Discover or Command appliance, followed by `/api/v1/explore/`. For example, if your hostname is seattle-eda, the URL is `https://seattle-eda/api/v1/explore/`.
2. Paste or type your API Key into the `api_key` field at the top of the page.
3. Click **ExtraHop** and then click **POST/extrahop/sslcert/signingrequest**.
4. In the Parameters section, click **Model Schema**, and then click in the box to automatically add the JSON schema to the body parameter text box.
5. In the body parameter text box, specify the certificate signing request fields.
   a) In the `common_name` field, replace `string` with the fully qualified domain name of your ExtraHop appliance.
   b) In the `subject_alternative_names` field, add one or more alternative domain names or IP addresses for your appliance.

      Note: The `subject_alternative_names` field is required. If your appliance has only one domain name, duplicate the value from the `common_name` field. You must include at least one subject alternative name with the type set to `dns`, but additional alternative names can have the type set to `ip` or `dns`.

   c) (Optional) In the `email_address` field, replace `string` with the email address of the certificate owner.
   d) (Optional) In the `organization_name` field, replace `string` with the registered legal name of your organization.
   e) (Optional) In the `country_code` field, replace `string` with the 2-character ISO country code of the country that your organization is located in.
   f) (Optional) In the `state_or_province_name` field, replace `string` with the name of the state or that your organization is located in.
   g) (Optional) In the `locality_name` field, replace `string` with the name of the city that your organization is located in.
   h) (Optional) In the `organizational_unit_name` field, replace `string` with the name of your department within your organization.

   The Value section should look similar to the following example:

```
{
  "subject": {
    "common_name": "example.com",
    "email_address": "admin@example.com",
    "organization_name": "Example",
    "country_code": "US"
  },
  "subject_alternative_names": [
    {
      "name": "www.example.com",
      "type": "dns"
    }
  ]
}
```

6. Click **Try it out!** to create the signing request.
   The signing request appears in the `pem` field in the `Response Body` section.

Next steps

Send the signing request to your CA to create your signed SSL certificate.

Important: The signing request contains escape sequences that represent line breaks (\n). Replace each instance of \n with a line break before sending the request to your CA. You can

modify the PEM request manually in a text editor or automatically through a JSON parsing utility, as shown in the following example command:

```
echo '<json_output>' | python -c 'import sys, json; print
 json.load(sys.stdin)["pem"]'
```

Replace the `<json_output>` variable with the entire JSON string returned in the Response Body section.

Add a trusted SSL certificate to your ExtraHop appliance

You can add an SSL certificate signed by a trusted CA to your ExtraHop appliance through the REST API Explorer.

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

   The URL is the hostname or IP address of your ExtraHop Discover or Command appliance, followed by `/api/v1/explore/`. For example, if your hostname is seattle-eda, the URL is `https://seattle-eda/api/v1/explore/`.
2. Paste or type your API Key into the `api_key` field at the top of the page.
3. Click **ExtraHop** and then click **PUT/extrahop/sslcert**.
4. In the **Certificate and Key** field, paste the SSL certificate.

   The certificate should look similar to the following text:

```
-----BEGIN CERTIFICATE-----
a0O8zvV4MlDhWX4e0VyvGAJx+9d4AqQB4Czy/P7z36CmHe2Y7PPdVSeWHNCQoJ0g
CnO42u2V9YKNFYRQejIJv8CxGVJKsdfV0iP0WnCvpZXkaBOYIrDvE5xn010WPUls
6qe3mCXsUK87i++mYuVDA1U0A5YVXRO2OOWIWy7P+MCU/cR/op3Jpekng2cxN4qD
FqGbtRpLdCuJ/xGWL1FFRHBg76+TbO+pxgZhiCtHYXfMKIaoPmDwsAqEtLbizz1W
mbMig9hs4QNcJ+aMNSnTZpkbeBR4a2nkGnQoYvnFOXV/nWzvfHmI4ydSH9g4I8qt
4ArqFepInvm70n07FYAKL6Mdd1i+7ieo9AqckltVzzKFzkakHm04214wtsYmle94
4HqIJ7p7NH5maXxttXMzHFlArbnjHWCl0gIv8lAu+IvLJ8aiGAb3zqveNz6ZAZ5j
PGAUsP+dVYV/8VjvqhkiP/1jWzUHwzpdlHbcD8qOkAF41fnbv+2EXqFJ096JSSiU
rqeJpgNuH3LbkT0KORAiLoGLMZKEKxF+3OpLVD7ox7NQh9pMdZlB8tcTbTmsvD8T
3L2tMVZssqYOANcidtd17t72VW4hzQURT1me5tGWxpN6od/q6B+FIvRq/7Vq0UE1
c2AG/om5UN/Vj3pUjXzq/B1IWUS9TicRcKdl5wrKEkPUGjK4w1R/87bj5HSn8nyd
lMCcOpLTokHj0B5+8O1ylNhVXNPlj3eY0n6OQOdClBqTDM0/4sB3XgeC/pjpleU3
3uot+wM/GoN/Dqb1LPt3BNpUQuCzSfmGSSOXiWELsEhz3ix/36a9eUWjfhmtPsW5
dne5Lf+G7cf+ebsRTb7R89GmgKzTpUl1KAzKINAebkT6WrWWljugpA0BcfANjS6o
mik4ZbY8d54UtA17evprr2+8UotIgVIrCbfLgA2DY8QOTCBYIFKJ3GZAedqRK9Sm
I2qdaB6QBczYNaVYSeCsBdHHw1+h7dBeqdUUwYKtmPW96/djj/6vJSXh9/UX/3c0
eqXG36w/lqJAYu8QtAydJsVC85IzqzikkX0f0KE315Doginpg59yix9dHD2sxLb1
X39BRpLkZ9nvW6ke2YHU/VKBVIxqSslukGoTUIcUtPJrtMQOwCi/EQQXbPK9a2pW
K51938h6OuLjNbDTFuxfhE4zITWHTgyAs2MNVR9+uDUiVJclX+CIPjhZzjyPqmD6
6uh8Sr3zndOMabqDquo69rMQyvclF0xOUMVgUw1Rb8Y=
-----END CERTIFICATE-----
```

> **Note:** If you want the certificate to be signed with your own private key, you can include your key after the SSL certificate, separated by a line break. However, we recommend that you do not specify your own key; by default, the appliance will sign the certificate with the private key on the appliance.

5. Click **Try it out!** to add the certificate.

Create custom devices through the REST API

You can create custom devices through the REST API that track network traffic across multiple IP addresses and ports. For example, you might want to add a custom device for each branch office. If you create the devices through a script, you can read the list of devices from a CSV file. In this topic, we will demonstrate methods for both the REST API and the ExtraHop REST API Explorer.

Before you begin

- You must log into the ExtraHop appliance with an account that has unlimited 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 REST API Explorer.

Create a custom device

Before you can associate a custom device with an IP address, you must create the device in the ExtraHop system.

1. In the REST API Explorer, click **Custom Device**, and then click **POST /customdevices**.
2. In the body field, specify properties for the custom device that you want to create.

```
{
  "description": "The location of our office in Washington",
  "name": "Seattle"
}
```

Add custom device criteria

When you initially add a custom device, it is an empty object in the ExtraHop system. To associate the custom device with a real-world device or application, you must add one or more IP addresses.

1. In the REST API Explorer, click **Custom Device**, and then click **POST /customdevices/{id}/criteria**.
2. In the body field, specify criteria for the custom device that you created.

   For example, the following body matches the custom device to the CIDR block "192.168.0.0/24":

```
{
  "ipaddr": "192.168.0.0/24"
}
```

3. In the ID field, specify the numeric ID for the custom device.

Tip: You can view the IDs of custom devices by navigating to **Custom Device > GET/customdevices** and clicking **Try it out!**. The ID appears in the response body under `id`.

Python script example

This example python script creates custom devices by reading criteria from a CSV file. Each row of the CSV file must contain the following columns in the specified order:

Name	ID	Description	IP address or CIDR block

Note: The script does not accept a header row in the CSV file. There is no limit to the number of columns in the table; each column after the first four specifies an additional IP address for the device. The first four columns are required for each row.

For example, the following CSV list contains criteria for offices in France, Holland, and California:

```
France,francehq,The location of our office in
 France,192.168.0.103,192.168.0.105,192.168.0.101
Holland,hollandhq,The location of our office in Holland,192.168.0.102
California,californiahq,The location of our office in
 California,192.168.0.104,192.168.0.103
```

The script includes the following configuration variables:

- **HOST:** The IP address or hostname of the Discover appliance
- **APIKEY:** The API key
- **CSV_FILE:** The path of the CSV file relative to the location of the script file

```
#!/usr/bin/env python2

import json
import httplib
import csv
import os.path

HOST = 'example.extrahop.com'
APIKEY = 'f6876657888a7c1f24ac77827'
CSV_FILE = 'device_list.csv'

headers = {'Content-Type': 'application/json',
           'Accept': 'application/json',
           'Authorization': 'ExtraHop apikey=%s' % APIKEY}

def readCSV():
    devices = []
    with open(CSV_FILE, 'rb') as f:
        reader = csv.reader(f)
        for row in reader:
            device = {}
            device['name'] = row.pop(0)
            device['extrahop_id'] = row.pop(0)
            device['description'] = row.pop(0)
            device['ipaddr'] = row
            devices.append(device)
    return devices

def createDevice(device):
    ips = device.pop('ipaddr')
    conn = httplib.HTTPSConnection(HOST)
    conn.request('POST', '/api/v1/customdevices', body=json.dumps(device),
 headers=headers)
    resp = conn.getresponse()
    if resp.status != 201:
        print "Could not create device: " + device['name']
        print "    " + json.loads(resp.read())['error_message']
        return -1, ips
    else:
        print "Created custom device: " + device['name']
        device_id = os.path.basename(resp.getheader('location'))
        return device_id, ips

def addIPs(device_id, ips, name):
    url = '/api/v1/customdevices/' + device_id + '/criteria'
    for ip in ips:
        conn = httplib.HTTPSConnection(HOST)
        body = {"ipaddr": ip}
        conn.request('POST', url, body=json.dumps(body), headers=headers)
        resp = conn.getresponse()
        if resp.status != 201:
            print "    Could not add IP " + ip + " for " + name
            print "        " + json.loads(resp.read())['error_message']
        else:
            print "    Added IP " + ip + " for " + name

devices = readCSV()
for device in devices:
```

```
    device_id, ips = createDevice(device)
    if device_id == -1:
        continue
    addIPs(device_id, ips, device['name'])
```

Create and assign a device tag through the REST API

The following Python script creates a device tag and then assigns 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")
```

Query for metrics about a specific device through the REST API

The following Python script queries for metrics from an HTTP client device with the ID 9363 and prints 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
      ]
     }
```

Create, retrieve, and delete an object through the REST API

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"
```

Query the record log

The following request body queries the record log to retrieve 100 HTTP records where the method is GET and the status code is 404.

```
{
  "filter": {
    "operator": "and",
    "rules": [
```

```
        {
            "field": "method",
            "operand": "GET",
            "operator": "="
        },
        {
            "field": "statusCode",
            "operand": "404",
            "operator": "="
        }
    ]
  },
  "from": -900000,
  "limit": 100,
  "types": [
    "~http"
  ]
}
```