ExtraHop 8.2
Triggers Best Practices Guide

Published: 2021-09-04

Writing a trigger to collect metrics is a powerful way to monitor your application and network performance. However, triggers consume system resources and can affect system performance—a poorly written trigger can cause unnecessary system load. Before you write a trigger, you should evaluate what you want your trigger to accomplish, identify which events and devices are needed to extract the data you need, and determine whether a solution already exists.

Identify what you want to know

Clearly identify what information you would like to know or what you want the trigger to accomplish, such as the following examples:

- When will my SSL certificates expire?
- Is my network getting connections on non-authorized ports?
- How many slow transactions is my network experiencing?
- What data do I want to send to Splunk through an open data stream?

By identifying the problem you are trying to solve, you can better target a trigger to collect only the information you need and avoid taking an unnecessary performance hit.

Focus the trigger on one logical function

A trigger should perform one logical function. If you need a trigger that performs a different task, create a second trigger. A trigger that performs multiple, unrelated tasks consumes more resources.

Search for metrics in the Metric Catalog

Be sure to review built-in metrics in the Metric Catalog. Built-in metrics do not create additional load on the system, and might already have the information you need.

Identify which system events produce the data that you want to collect

For example, a trigger that monitors cloud application activity in your environment might run on HTTP responses and on the open and close of SSL connections.

Identify the devices or networks that you want to monitor and collect metrics from

A trigger consumes fewer system resources if you target specific devices instead of all devices of a particular type or group. For example, a trigger that looks for slow responses from your online catalog should be assigned only to HTTP servers that handle catalog transactions and not to all HTTP servers.

Determine how you want to visualize or store data collected by the trigger

For example, you can view metrics on a dashboard, you can send records to the ExtraHop Explore appliance, or you can send data to a third-party system, such as Splunk.

Determine if a trigger already exists

It is possible that a trigger that meets your needs or might be easily modified already exists; always start with a pre-existing trigger whenever possible. Search the following resources for available triggers:

- The ExtraHop Solution Bundles Gallery
- The ExtraHop Community Forums

Optimize the trigger configuration

The configuration options available when you create or modify a trigger can affect trigger performance and efficiency. ExtraHop recommends the following practices to optimize and improve trigger performance.

Take advantage of advanced trigger options

If advanced trigger options are available for the event the trigger runs on, we recommend that you configure applicable options to narrow the focus of the trigger and improve performance and results. For example, if you create a trigger that runs on the SSL_PAYLOAD event, you can specify minimum and maximum port numbers so that the trigger only collects data from transactions within the specified port range.

Assign the trigger to minimal resources

Prevent the trigger from running unnecessarily by assigning the trigger to as few sources, such as devices, as possible. Do not select **Assign to all devices** when assigning the trigger, and avoid assignments to large device groups.

Debug only when testing

Debugging should be enabled only while actively working on your trigger. Debugging is useful for testing that the trigger runs and collects expected data; however, debugging is an unnecessary drain on resources and should be avoided in a production environment.

Write an optimized trigger script

The following sections discuss general coding best practices as well as guidelines when working with API components and objects, such as session tables and records.

Applying coding best practices

When writing a trigger script, we recommend the following coding best practices to optimize and improve trigger performance.

Fix trigger exceptions and errors immediately

Exceptions severely affect performance. If exceptions or errors appear in a trigger's debug log, fix those issues immediately. If there are exceptions that cannot be easily avoided, you can attempt to handle the exception with a `try/catch` statement. The `try/catch` statement should be wrapped in a small function that does not perform any other operation.

In the example below, the `JSON.parse` function results in an exception if the input is not in well-formed JSON syntax. Because there is no way to validate that the JSON syntax is well-formed prior to parsing, this task is wrapped in a small helper function.

```
function parseJSON(jsonString) {
    try {
        return JSON.parse(jsonString);
    } catch (e) {
        return null;
    }
}
let obj = parseJSON(HTTP.payload);
```

Keep actions where they are needed in the script

Move actions, such as string operations and access to properties, into the most exclusive branch that requires the action.

```
// inefficient: The cookies variable may not be needed
let cookies = HTTP.cookies;
if (HTTP.method === 'POST') {
 // process cookies
}

// optimized: Cookies are not accessed unless they are needed
if (HTTP.method === 'POST') {
 let cookies = HTTP.cookies;
```

```
 // process cookies
}
```

Locally store often-accessed objects

Conserve CPU cycles by locally storing applications, geolocations, devices, and other objects that are accessed multiple times.

```
// store object locally
let app = Application('example');
app.commit();
app.metricAddCount('custom', 1);
```

Avoid variables in the debug statement

Do not include a variable in the debug statement solely for the purposes of debugging when you are accessing a payload or other large object. Excluding variables from the debug statement enables you to comment out the debug line and maintain access to the payload information.

Do not include the exit() function in the script

Insert a `return` statement instead of the `exit()` global function, which is deprecated due to poor performance.

```
// avoid: slows the trigger down
HTTP.uri || exit();

// recommended
if (!HTTP.uri) return;
```

Note: The behavior of the `return` statement is different when inside of a nested function; in a nested function, the `return` statement exits only the current function, not the entire trigger.

Do not include the eval() function in the script

Triggers should not attempt to dynamically generate code at runtime; therefore, the `eval()` function is not supported by the ExtraHop Trigger API. Including the function in the trigger might cause unpredictable runtime errors.

Prioritize regular expressions

Apply the `test()` method with regular expressions instead of the `match()` method, when possible. Extracting a value with a regular expression is often faster than full parsing XML or query strings.

```
// slower search
let user = HTTP.parseQuery(payload).user;

// faster search
let match = /user=(.*?)\&/i.exec(payload);
let user = match ? match[1] : null;
```

Apply strict equality operators

For comparisons, apply strict equality operators, such as triple equals (===) instead of loose equality (==), whenever possible. Strict equality operators are faster because they do not attempt type coercions, such as converting IP addresses to strings before comparing them.

Do not include undeclared variables

Declare variables with `let`, `var`, or `const` statements; never include undeclared variables.

```
// inefficient: The variable "cookies" is never declared
cookies = HTTP.cookies;

// optimized: The variable "cookies" is declared
let cookies = HTTP.cookies;
```

Organize the script with functions

Functions are good for performance and code organization. Define functions at the beginning of the trigger if your trigger performs the same operation in different places in the trigger script, or if trigger operations can be logically separated into discrete parts.

Organize helper functions at the top of the script

Define helper functions at the top of the trigger script, rather than at the bottom, to improve readability and help the ExtraHop system better understand and run your code.

Adding session tables

The session table globally shares simple data types (such as strings) across all triggers and flows. Adhere to the following best practices when adding session tables to your trigger script.

Increment counts in session tables

When counting, call the `Session.increment` function instead of the `Session.lookup` or `Session.replace` functions. The `Session.increment` function is atomic and is therefore guaranteed to be correct across multiple trigger threads.

Do not collect distinct count metrics with a session table

A session table is not an optimal tool for counting the number of unique values, such as IP addresses, ports, or usernames, and is likely to cause performance problems. Instead, commit a custom distinct count metric with one of the following methods:

```
metricAddDistinct (
   name: String,
   item: String|Number|IPAddress
)

metricAddDetailDistinct (
   name: String,
    key: String|IPAddress,
   item: String|Number|IPAddress
)
```

Do not create or iterate very large arrays or objects

Adding large arrays or objects to a session table might reduce performance. For example, do not include multiple expiring keys, which have a `notify=true` value, in a trigger that runs on the `SESSION_EXPIRE` event. In addition, do not call `JSON.parse` or `JSON.stringify` methods on large objects in the session table.

Placing values in the Flow store

The Flow store collects values across events on a single flow, such as a request/response pair. Adhere to the following best practices when placing values in the Flow store in a trigger script.

Determine whether values are already available

Before working with the `Flow.store` property, check whether the values you want are already available directly from the ExtraHop Trigger API. Recent firmware versions have removed the need for a Flow store for a number of request properties that were commonly consumed in the response, such as `HTTP.query` and `DB.statement`.

Add only needed values to the Flow store

Avoid adding large objects to the `Flow.store` property if you only need a subset of the object values. For example, the following code stores only the `TTL` property value from the `DNS.answers` object, instead of storing all property values from the `DNS.answers` object:

```
Flow.store.ttl = DNS.answers[0].ttl;
```

Clear values after they are no longer needed

Clear `Flow.store` property values when they are no longer needed by setting the property value to `null`. Clearing the Flow store helps prevent errors in which the trigger processes the same value multiple times.

Do not share Flow store values between triggers

Do not access the `Flow.store` object to share values between different triggers running on the same event. The order in which triggers run when an event occurs is not guaranteed. If one trigger depends on the value of the `Flow.store` property from a second trigger, the value might not be as expected and could yield incorrect results in the first trigger.

Creating custom records

Custom metrics provide the flexibility to capture the data you need if the data is not already provided by the built-in protocol metrics on the ExtraHop system. Adhere to the following best practices when creating custom metrics through a trigger.

Do not dynamically generate custom metric names

Dynamically generating custom metric names will degrade performance and might prevent metrics from being automatically discovered in the Metric Catalog.

Do not convert IP addresses to a string before adding them to custom metrics

If you convert an IP address to a string before adding it as the detail in a custom metric, the ExtraHop system cannot retrieve attributes associated with the IP address, such as the associated device or hostname.

```
// avoid
Application("SampleApp").metricAddDetailCount("reqs.byClient",
 Flow.client.ipaddr.toString(), 1);

// recommended
Application("SampleApp").metricAddDetailCount("reqs.byClient",
 Flow.client.ipaddr, 1);
```

Storing data in records

Records enable you to store and retrieve structured information about transaction, message, and network flows. Adhere to the following best practices when creating and storing records through a trigger.

Limit accessing properties through the record object

Avoid accessing properties through a record object, such as `HTTP.record`, if the properties are available in the protocol object or Flow object. When you access the record object, the record data is allocated in memory. Instead, access the record object when storing data in a modified custom record.

```
// avoid
let uri = HTTP.record.uri;

// recommended
let uri = HTTP.uri;
```

Do not convert IP addresses to strings before storing them in records

The ExtraHop system filters and sorts by IP addresses, which cannot be applied to strings such as CIDR blocks for IPv4 addresses.

Accessing the datastore

Datastore triggers enable you to access aggregate metrics that are not available on transaction-based events such as HTTP_REQUEST. Adhere to the following best practices when accessing the datastore through a trigger.

Create datastore triggers sparingly

Datastore trigger resources are very limited. Where possible, avoid running triggers on the following datastore events:

- ALERT_RECORD_COMMIT
- METRIC_RECORD_COMMIT
- METRIC_CYCLE_BEGIN
- METRIC_CYCLE_END

For example, if you only want to access a session table periodically, run the trigger on the SESSION_EXPIRE event instead of the METRIC_CYCLE_END event.

Configure advanced options when committing metric records

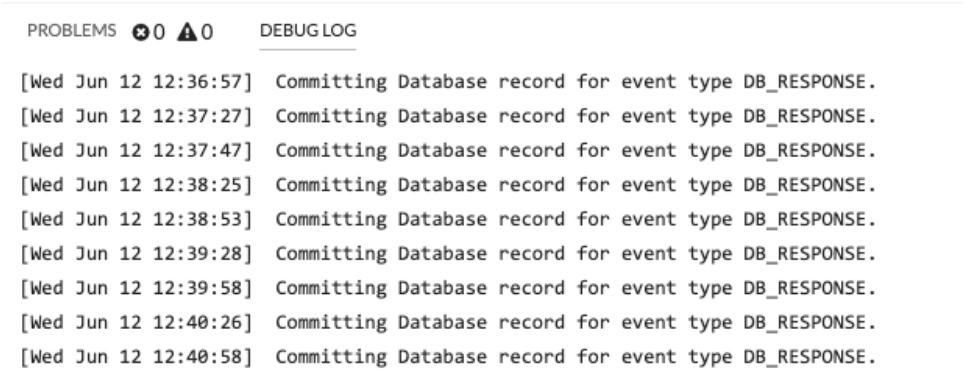
Ensure that advanced trigger options are configured for triggers that run on the METRIC_RECORD_COMMIT event. Specifically, we recommend that you configure a 30-second metric cycle for triggers that run on the METRIC_RECORD_COMMIT event.

Evaluate trigger performance

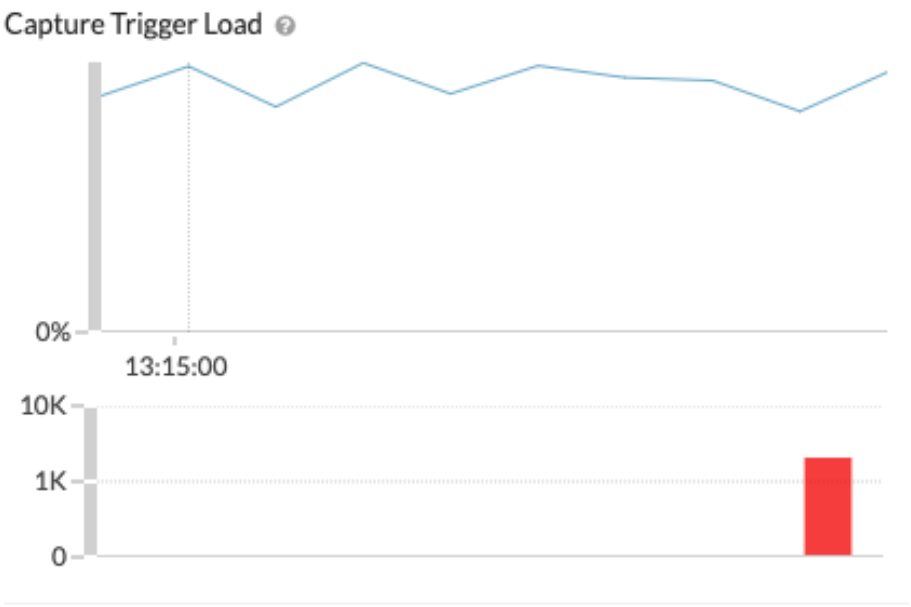
You can monitor and assess the impact of triggers running on your ExtraHop system in a couple of ways. You can view performance information for an individual trigger and you can view several charts that indicate the collective impact of all of your triggers on the system.

View the performance of an individual trigger

You can check whether a trigger has any errors or exceptions from the Debug Log tab in the Edit Trigger pane. Access the Triggers page by clicking the System Settings icon in the ExtraHop system.



You can view the performance cost of a running trigger on the Capture Trigger Load chart in the Edit Trigger tab. The chart displays a trigger performance graph that tracks the number of cycles the trigger has consumed within a given time interval. You can hover over a data point to display key performance metrics at a single point in time.



The hover tip includes the following information:

- The most and least cycles the trigger consumed to process a single event.
- The number of times the trigger ran and the percentage of times the trigger ran compared to all triggers that ran in the same time range.
- The total number of cycles consumed by the trigger and the percentage of cycles consumed compared to all triggers that ran in the same time range.

View the performance of all triggers on the system

The System Health page contains several charts that provide an at-a-glance view of the triggers running on the ExtraHop system. Access the System Health page by clicking the System Settings icon in the ExtraHop system.

By viewing the system health charts described in the following guidelines, you can monitor for problems that can affect system performance or result in incorrect data.

Verify that the trigger is running

The Trigger Details chart displays all triggers running on the system. If the trigger you just created or modified is not listed, you might have an issue with the trigger script.

Monitor for unexpected activity

The Trigger Executes and Drops chart can display bursts of trigger activity that might indicate inefficient behavior from one or more triggers. If any bursts are displayed, view the Trigger Executes by Trigger chart to locate any trigger consuming higher resources than average, which can indicate that the trigger has a poorly-optimized script that is affecting performance.

Check for unhandled trigger exceptions

The Trigger Exceptions by Trigger chart displays any exceptions caused by triggers. Exceptions are a large contributor to system performance issues and should be corrected immediately.

Check for triggers dropped from the queue

The Trigger Executes and Drops chart displays the number of triggers that have been dropped from the trigger queue. A common cause of dropped triggers is a long-running trigger that is dominating resource consumption. A healthy system should have 0 drops at all times.

Monitor resource consumption

The Trigger Load chart tracks the usage of all available resources by triggers. A high load is approximately 50%. Look for spikes in consumption that can indicate that a new trigger has been introduced or that an existing trigger is having issues.

You can monitor whether your datastore triggers, also referred to as bridge triggers, are running properly with the following charts:

- Datastore Trigger Executes and Drops
- Datastore Trigger Load
- Datastore Trigger Exceptions by Trigger

See the System Health FAQ to review frequently asked questions about how system health charts can help you assess trigger performance on your ExtraHop system.

View trigger resources

To learn more about triggers and the ExtraHop Trigger API, see the following resources:

Documentation

- ExtraHop Trigger API Reference
- Get started with triggers section of the Web UI Guide.

Walkthroughs

- Trigger walkthrough: Build a trigger to collect custom metrics for HTTP 404 errors
- UPA walkthrough: Build a trigger to monitor responses to NTP monlist requests
- Open data stream walkthrough: Configure an open data stream to send metric data to AWS Cloudwatch

Training

- Planning a Trigger
- Creating a Simple Trigger
- Creating and Using App Containers
- Creating a Multi-Event Trigger

Community

- The ExtraHop Solution Bundles Gallery
- The ExtraHop Community Forums