Application Inspection Triggers Quick Start Guide

In its default configuration, the ExtraHop system extracts and records a core set of Layer 7 (L7) metrics for the devices it monitors. The core set includes metrics such as response counts, error counts, and processing times because of their usefulness across a broad segment of the ExtraHop customer base. Once these metrics are recorded for a given L7 protocol message, the ExtraHop system by default discards the packets that make up the message, freeing resources for continued processing.

Application Inspection Triggers enable users to insert user-defined code that performs additional operations on protocol messages (for example, an HTTP request or a database response) prior to the packets being discarded.

Triggers enable the following use cases:

- Extracting and storing custom metrics to the onboard datastore.

  For example, while the ExtraHop system does not record information about a client's HTTP user agent that makes an HTTP request, that detail is available in the Application Inspection Triggers API. When you create a trigger, you can choose to access user agent detail and commit it to the datastore. You can also view custom data stored in the datastore in the ExtraHop Web UI using custom pages and by using the Metric Explorer and the new summary dashboards introduced in version 4.0.

- Associating an arbitrary cross-section of monitored traffic with an ExtraHop application container to enable tier-by-tier application-based views of data. Application views augment the device-based views that the ExtraHop system constructs in its default configuration.

- Extracting custom metrics and sending them to syslog consumers such as Splunk or third-party databases such as MongoDB.

- Initiating a packet capture to record individual flows to disk based on user-specified criteria. You can download captured flows to be processed using third-party tools. Your ExtraHop system must be licensed for packet capture to use this feature.

Related Documentation

This guide is designed to provide a general foundation for writing triggers. For more information about writing triggers, refer to the following documentation:

- Sample triggers on the ExtraHop Customer Forum: https://forum.extrahop.com
- *ExtraHop Application Inspection Triggers API* on the ExtraHop Support Forum: https://forum.extrahop.com/static/ExtraHopTriggersAPI.pdf
- *Triggers* section of the *ExtraHop Web UI Users Guide* in the ExtraHop Web UI

Planning a Trigger

When creating a trigger, you must answer three questions:

- Which *devices* require further processing?
- Which *events* on your target devices are you interested in processing further?
- How do you want to *process* the events?

For example, you might answer the above questions with the following statement:

When web servers web01, web02 and web03 *(devices)* receive an HTTP request *(event)* from subnet 10.10.1.0/24, record the information to the ExtraHop datastore about the user agent *(process)*.

Such statements underlie all triggers.

Example Trigger

The following example of a simple HTTP_RESPONSE trigger records a custom metric that counts all HTTP status codes greater than or equal to 400. In the default configuration, the ExtraHop system records statistics about each status code individually, but not in this particular grouping. Therefore, for statistics about this particular grouping, a trigger and a custom metric are required.

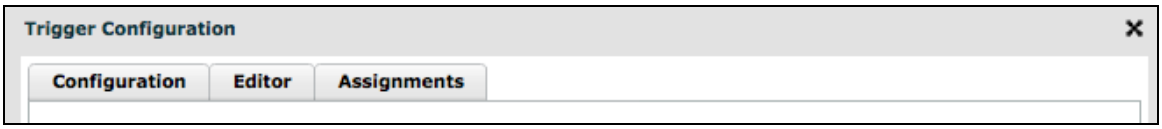
```
if (HTTP.statusCode >= 400) {
   Device.metricAddCount("http_4xx_5xx", 1);
}
```

Once captured, the custom metric is available for display on an HTML dashboard or on a custom page. For more information, refer to the Viewing Metrics with Application Containers and Custom Pages section.

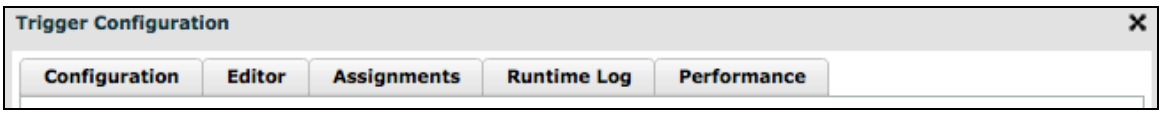
Creating a New Trigger

Creating a new trigger involves using the **Trigger Configuration** window in the ExtraHop Web UI to address the three questions from the Planning a Trigger section.

The answers to *which event, how to process the event,* and *which device* correspond respectively to the three tabs below: **Configuration**, **Editor**, and **Assignments**.



When the above three tabs are populated, the essential components of a trigger are in place. Two additional panels, **Runtime Log** and **Performance**, then become available to assist with trigger debugging and performance analysis.

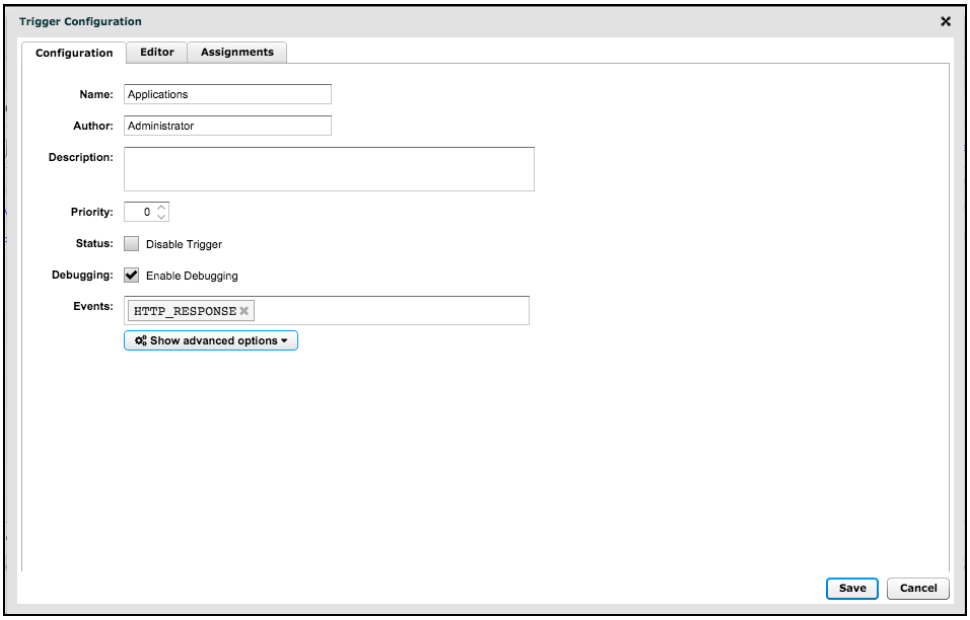


To create a new trigger:

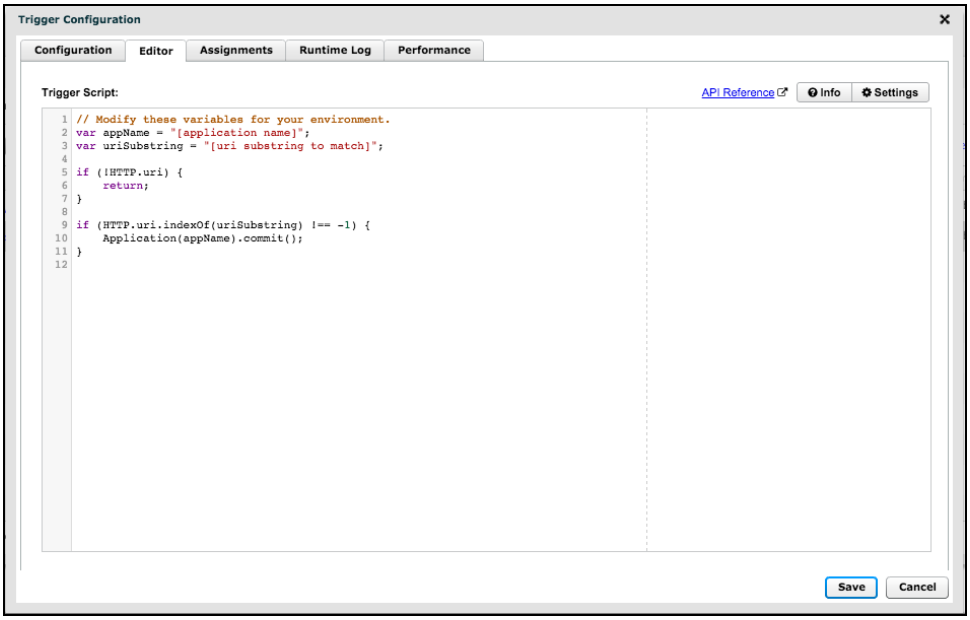
1. In the ExtraHop Web UI, click **Settings**, click **Triggers**, and then click **New**.

2. Enter a name for the trigger, and add the event that activates the trigger.

> ExtraHop recommends you do not use dynamic trigger names. Dynamic content should be placed in the key labels field, not in the trigger name itself.



3. Enter your trigger source code in the **Trigger Script** text box on the **Editor** panel. Trigger syntax follows that of JavaScript.



4. Click **Save**.

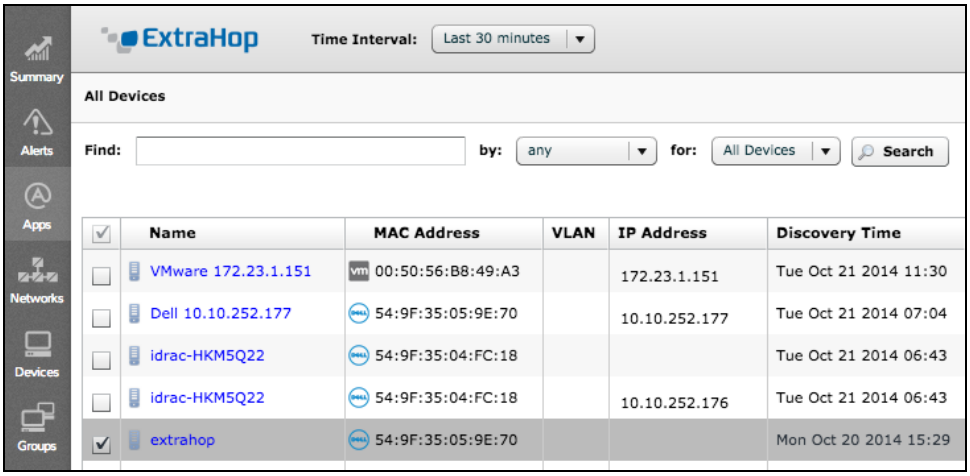
> Once you test the trigger to ensure it works, remove any `debug()` statements to avoid both potential performance issues and excessive debug messages in the Runtime Log.

5. Once you have finished writing your trigger code and saved it, click **Cancel** to exit the **Trigger Configuration** window.

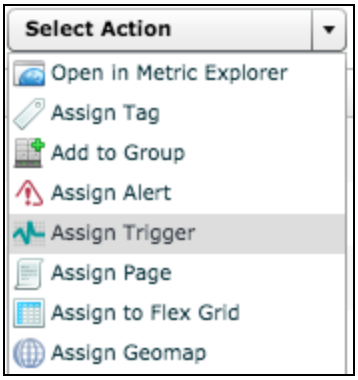
Assigning the Trigger to Devices

To assign the trigger to devices:

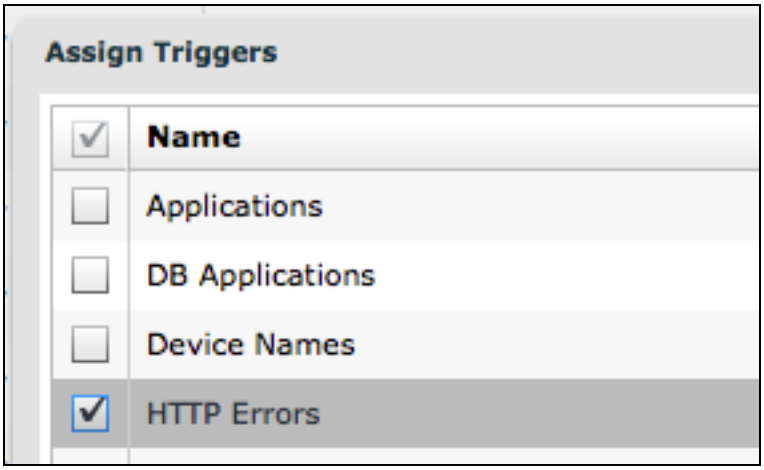
1. Click **Devices**, and select the devices to assign the trigger to.



2. Click the **Select Action** drop-down list in the upper-right portion of the screen, and select **Assign Trigger**.



3. Select the trigger from the list to assign it to these devices. You can also select the top checkbox in the left column to assign the trigger to all devices.



The trigger now executes on the selected devices whenever the trigger event occurs.

You may also assign the trigger to a device group, which assigns the trigger to each device in the group.

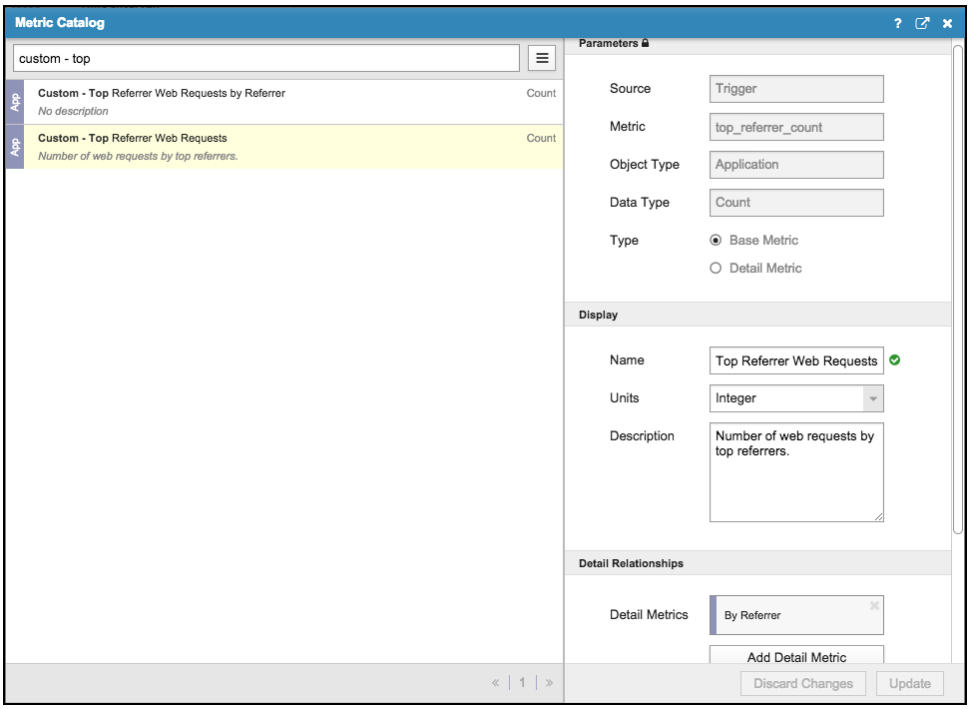
Editing Custom Metrics in the Metric Catalog

After you create your trigger, edit your entry in the Metric Catalog in order to make it as polished and informative as possible. These edits include the following:

- Update the display name to make the trigger more meaningful when used in a summary dashboard widget.
- Edit the description to improve understanding of the metric.
- Associate base and detail metrics.
- Add key labels to indicate how base metrics and detail metrics are related.

To edit your metric in the metric catalog (version 4.0+):

1. Open the Metric Catalog and type the name of the metric you want to edit in the **Filter** text box at the top of the page.



2. Select the metric you want to edit and modify the metric fields in the right column:
   - In the **Parameters** pane:
     - Do not modify the information in the **Parameters** pane as changes can potentially cause your custom metric to stop functioning correctly.
   - In the **Display** pane:
     - **Name**: The name field is populated based on what you entered in the Metric field in the **Configuration** pane.
     - **Units**: Select the unit of measure.
     - **Description**: Type a description of the metric. ExtraHop recommends you add a description, especially for complicated metrics.
     - **Key**: This optional field only appears if you select the Detail Metric radio button in the **Configuration** pane. Enter a key label to help determine how the base metric and detail metrics are related.
   - In the **Detail Relationship** pane:
     - If you selected the **Base Metric** radio button in the **Configuration** pane, click the **Add Detail Metric** button and select the appropriate detail metrics from the list.

- If you selected the **Detail Metric** radio button in the **Configuration** pane, click the **Set Base Metric** button and select the appropriate base metric from the list.

3. Click **Save** to save your changes to the custom metric.

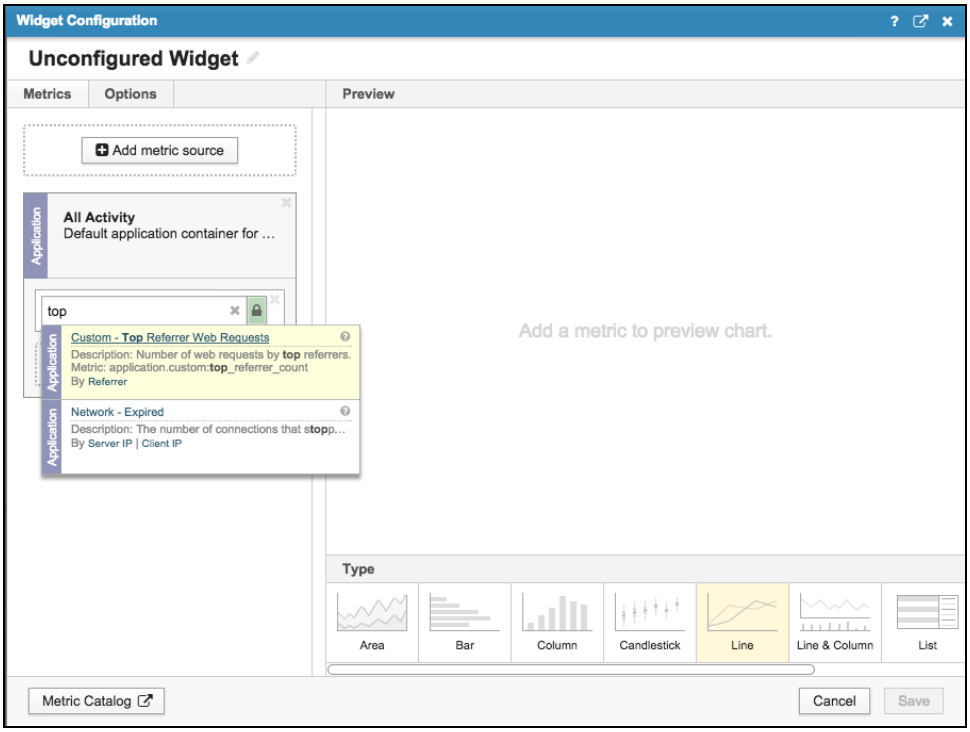
Viewing Custom Metrics on a Dashboard

In ExtraHop version 4.0, you can add both base and detail custom metrics to a dashboard and view them alongside built-in metrics.

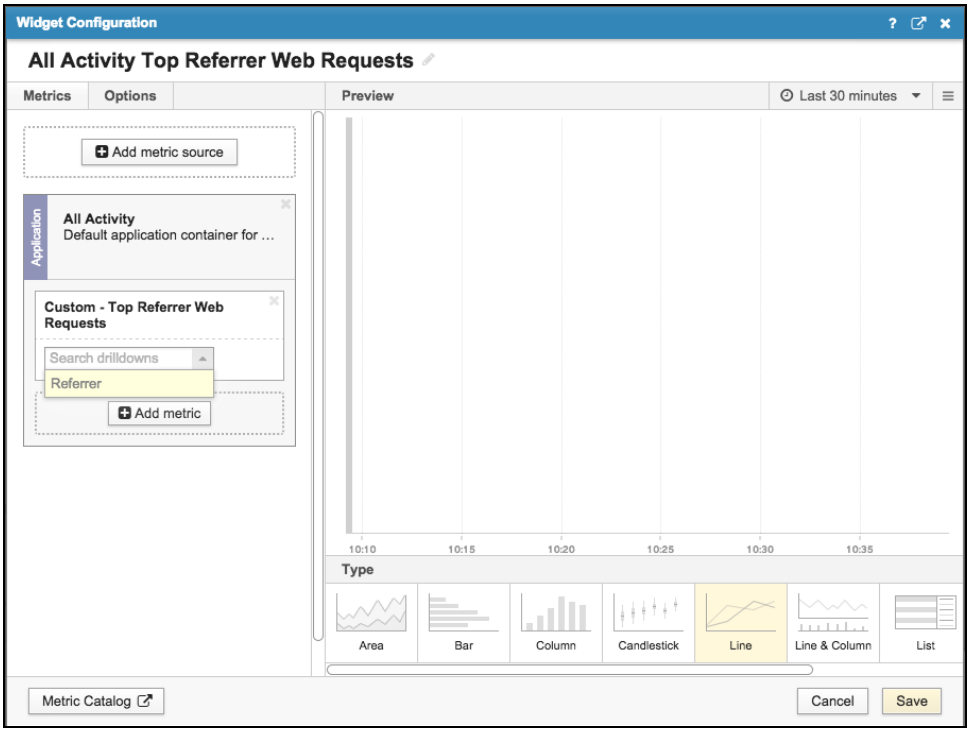
To view your custom metrics on a dashboard:

1. Create a new dashboard.
   OR
   In the dashboard dock, select the dashboard on which you want to place the widget for your custom metric, then click the **Configuration** button in the upper right corner of the screen and select **Edit Layout**.
2. Click the **Region** icon and drag it onto the dashboard. You can resize the region by dragging the lower right corner.
3. Click a **Widget** icon and drag it onto the region. You can resize the widget by dragging the lower right corner.
4. Click the **Configuration** button in the top right corner of the widget and select **Edit**.
5. On the **Summary** page, click the **Configuration** button in the upper right corner and select **Metric Explorer**.
6. (Optional) Click **Unconfigured Widget** in the upper left corner and enter a title.

7. Click the **Add Metric Source** button and enter all or part of your custom metric name.



8. Select your metric from the list. A preview chart appears on the right.



9. In the upper left corner, click **Options** to change the units, show the metric as a rate, or change the

suffix notation.

10. In the upper right corner, select the time interval to display the metric.
11. Select a chart type at the bottom to change the appearance of the chart.
12. Click **Save**. The widget appears in the region.
13. Click the **Exit Layout Mode** button in the upper right corner of the dashboard to return to the **Summary** page.

Viewing Metrics with Application Containers and Custom Pages

Using application containers and custom pages, you can view custom metrics that were created or populated by triggers.

Application Containers

Application containers are preconfigured page templates that are added by triggers. Use the Application.commit() method to add and populate an application container. This application container page displays data only for devices that have been committed to this application by a trigger.

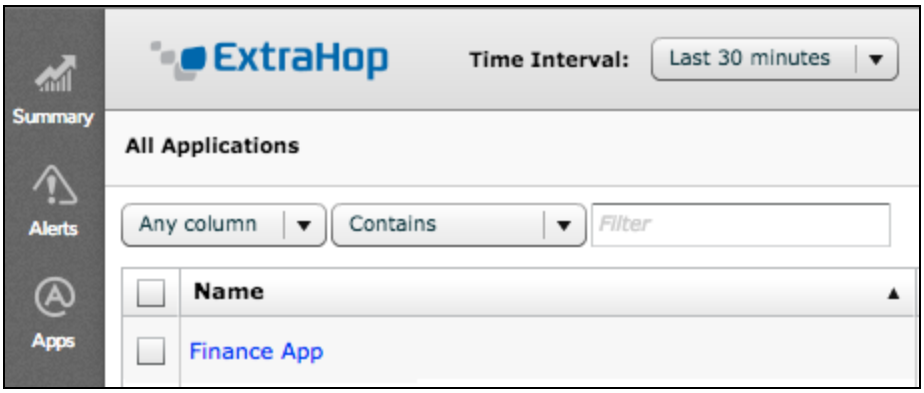
The following example adds and populates an application container page called Finance App, which includes all HTTP responses that contain the string finance.

```
if (HTTP.uri.indexOf("finance") > -1) {
   Application("Finance App").commit();
}
```

In order for this trigger to run properly, the event must be set to `HTTP_RESPONSE` and the trigger must be assigned to any HTTP server that runs **Finance App**.

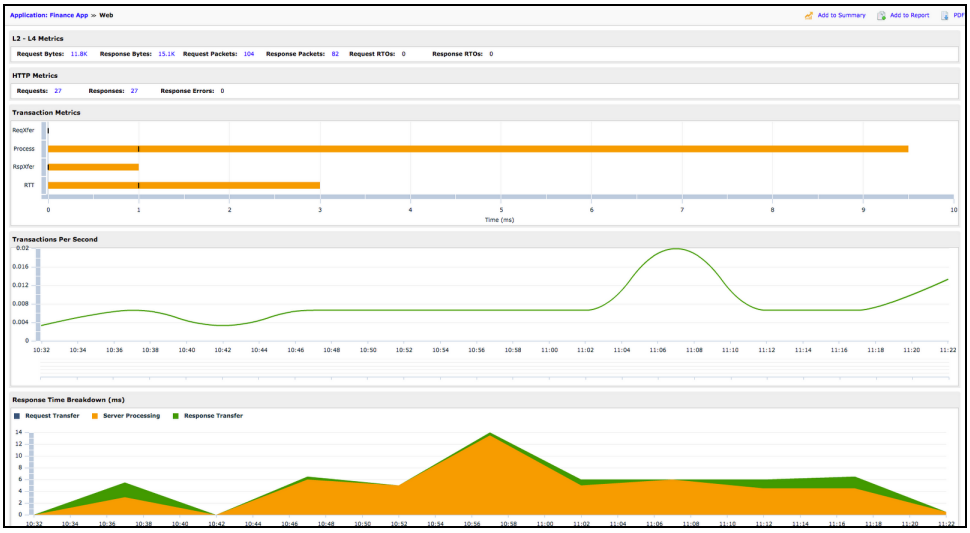
To view metrics through application containers:

1. Click **Applications** to view the Finance App page.



2. Click **Finance App**, and a summary page of metrics associated with that application appears.

3. Click one of the top-level metrics to view the detailed metrics, which also apply specifically to this application.



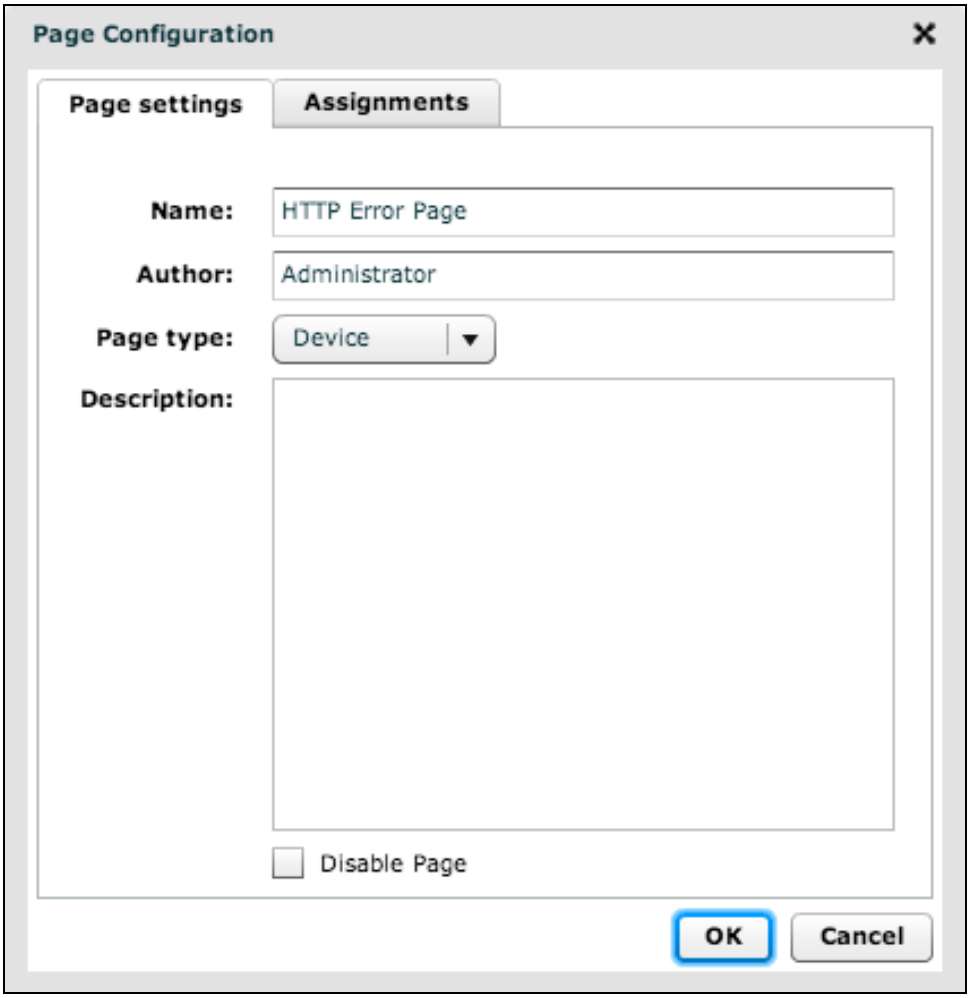
Custom Pages

Custom pages provide the ability to add graphs and charts that display recorded metrics. After you create a new custom page, you must assign devices to it and add charts to it. The content on these charts comes from the trigger you created earlier.

To create a new custom page:

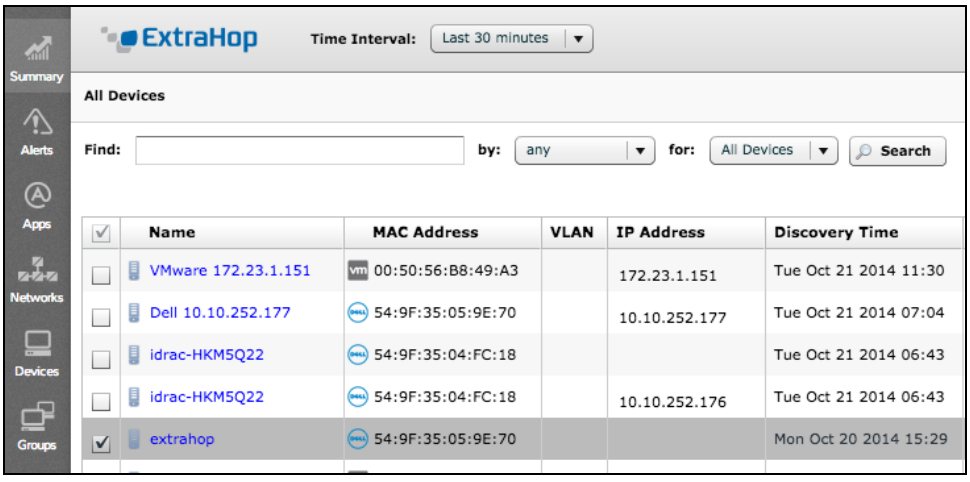
1. Click **Settings**, click **Pages**, and then click **New**.

2. Enter a name for the page, enter optional comments in the **Description** box, and then click **OK**.

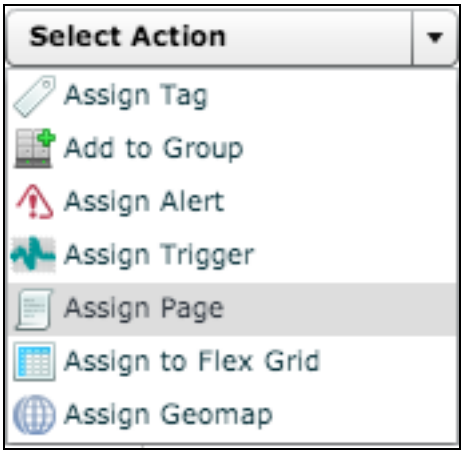


To assign a custom page to a device:

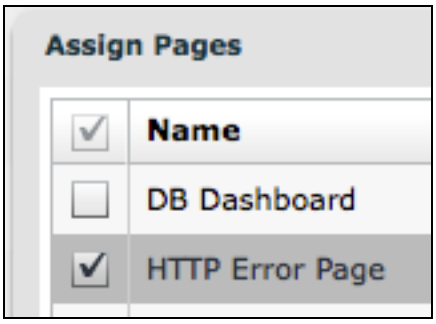
1. Click **Devices** and select the devices to assign the custom page to.



2. Click the **Select Action** drop-down list in the upper-right portion of the screen, and select **Assign Page**.



3. Select the custom page(s) from the list to be assigned to these devices.



The custom page is now assigned to the selected devices.

Newly created custom pages are blank until you add charts.

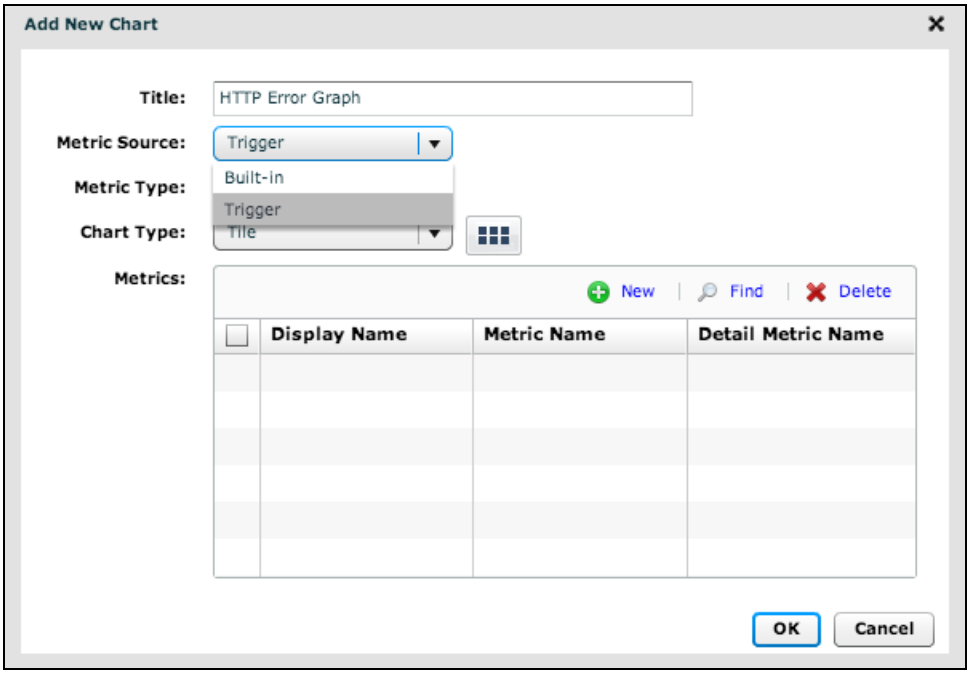
To add a chart to the custom page:

1. Navigate to a device that the custom page is assigned to.
2. Click the custom page from the device's tree control on the left.

3. Go to the custom page, and click **Edit Page**.

4. Click **Add Chart**.

5. Enter a title for the chart. Click the **Metric Source** drop-down list and select either **Built-in** or **Trigger** for the metric source.



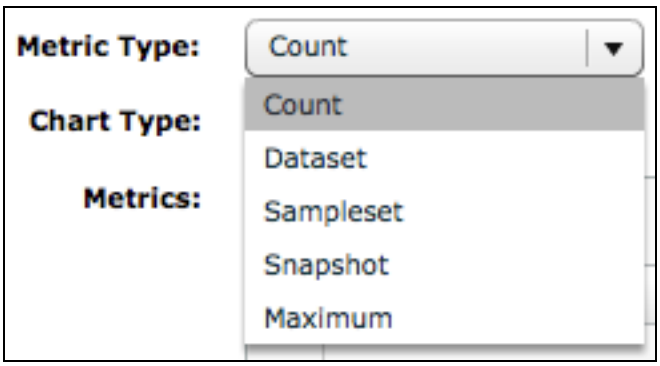
Working with Built-In and Custom Metrics

Built-in metrics are metrics that the ExtraHop system automatically collects without requiring triggers.

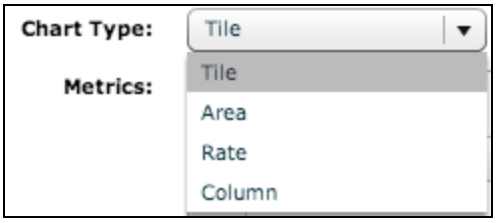
Custom metrics are user-defined metrics that are added to the datastore through the `metricAdd*()` methods in the trigger's source code.

To create a custom metric:

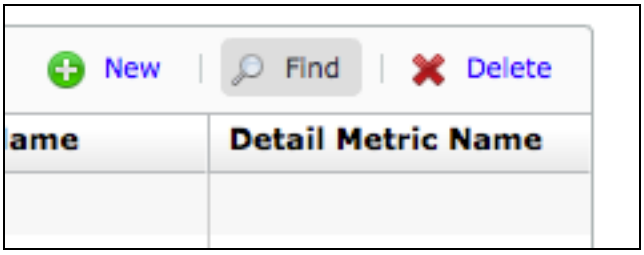
1. Click the **Metric Type** drop-down list, and select **Count**.



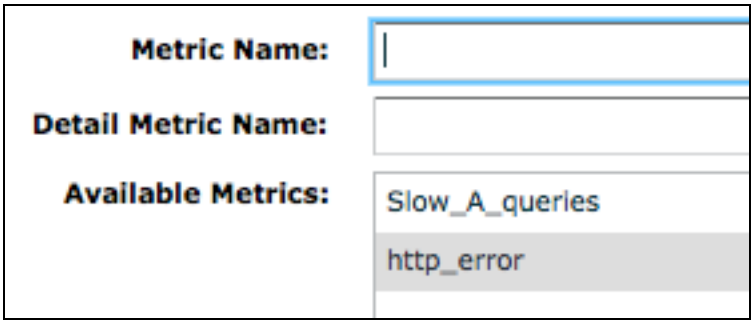
2. Click the **Chart Type** drop-down list, and select one of the options. The following example uses **Tile** as the chart type.



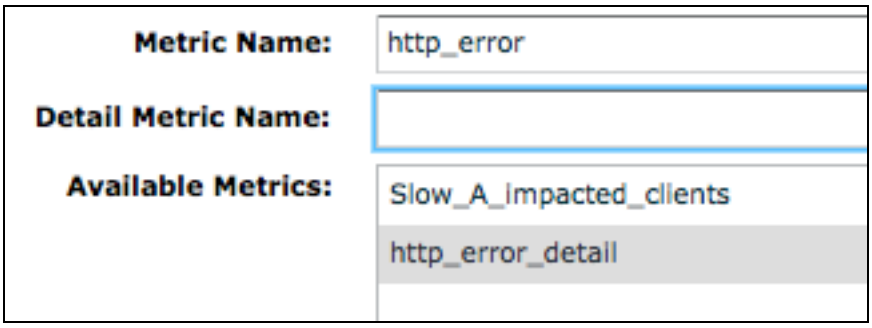
3. Click **Find** to open the **Select Metric** dialog box.



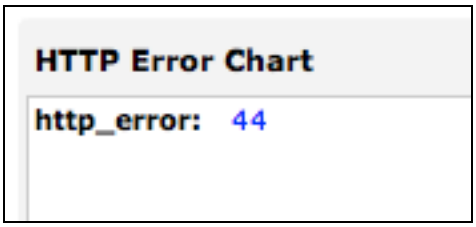
4. Select the metric to include in the chart's top-level view from the list of available metrics.



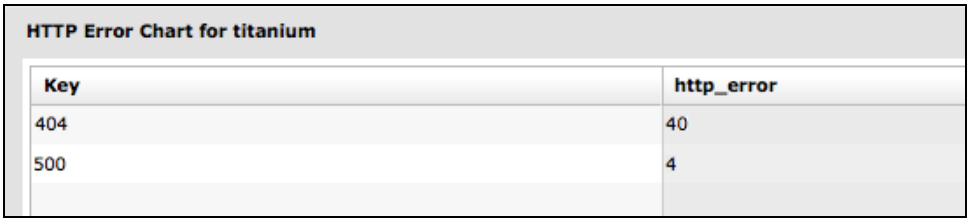
(Optional) You can also click in the **Detail Metric Name** text box to display a list of detailed metrics. Select the metric to include in the chart's detail view from the list of available metrics.



These selections create a chart that displays the number of HTTP errors from the included devices. The following is an example of how the result may look:



5. Click the **http_status** count to display the detail chart. The following is an example of how the result may look:



Notes

- Metrics are of a specific type. If your metric is defined as type **Count** using the `metricAddCount()` method, it appears in the above search only if the **Metric Type** is **Count**. If the metric is of type **Dataset**, **Sampleset**, etc., it appears only if the type selected in the **Metric Type** drop-down list matches it.

  For type definitions, refer to the *ExtraHop Application Inspection Triggers API.*

- In the example above, a **Chart Type** of **Tile** was selected in the **Chart Configuration** dialog box. This chart type generated a simple chart. However, you can select other chart types to generate graphs and other displays.

- Only metrics that have collected at least one data point appear on the chart. If the metric is defined inside a trigger but has not yet received any data, it does not appear. However, you can manually enter the metric name the **Chart Configuration** dialog box and it will appear in your chart with a value of 0. When it begins to collect data, the value automatically updates to reflect the new information.

- Custom metrics and custom detail metrics are added to devices using the `metricAdd*()` methods. For the above example, the following methods were used to create and populate the **http_error** and

http_error_detail metric names:

```
Device.metricAddCount("http_error", 1);
Device.metricAddDetailCount("http_error_detail", http_status.toString(),
```

Troubleshooting Triggers and Custom Pages

I don't see a New button on my trigger page. How do I create one?

A **New** button appears only when you are logged into an account that has write permission. Without write permission, you cannot create triggers. You must log in with an account that has write permission or contact your ExtraHop administrator to enable write permissions for your account.

How can I tell if my trigger is being executed?

To see when the trigger was last executed, click **Settings**, click **Triggers**, click the trigger you want to view, and click the **Performance** tab.

With the **Enable Debugging** checkbox selected, use the `debug()` statement to enable debugging. Click the **Runtime Log** tab to view unique `debug()` statements. The runtime log updates every 30 seconds.

How can I debug my trigger?

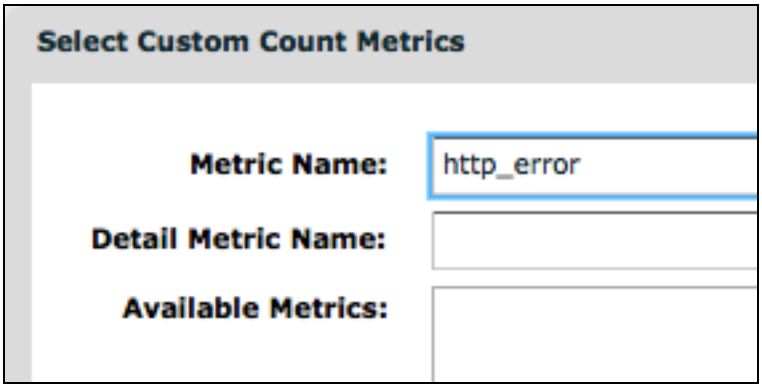
Use the `debug()` statement extensively when developing and testing your trigger to ensure it runs and generates the data you expect. The debug output appears in the **Runtime Log** tab. Each unique debug statement is printed only once.

Once you verify that your trigger performs as expected, remove the `debug()` statements. You can also disable debugging by unchecking the **Enable Debugging** checkbox in the **Trigger Configuration** window to optimize performance.

Why don't my custom metrics appear in my custom page?

The custom page type must match the custom metric type for metrics to be displayed.

Metrics appear only when they contain at least one data point. A custom metric that has not received any data points does not appear in the list of custom metrics. However, you can manually enter the metric name so it appears in your custom chart with a count of 0. When the ExtraHop system begins collecting samples, this number updates.



How can I improve the performance of my trigger?

ExtraHop recommends the following practices to improve performance:

- Assign triggers only to devices that require the collection of custom metrics. Assigning triggers to all devices causes unnecessary trigger executions that may cause the system to run slowly.
- Monitor how frequently your triggers execute by using the **Performance** tab in the **Trigger Configuration** window. ExtraHop recommends optimizing triggers that execute frequently.
- Remove all `debug()` statements once you test your trigger and know that it works correctly, or just disable debugging on the trigger.
- Avoid unnecessarily complex code, such as loops with a potentially high number of iterations, nested `while` and `for` loops, and inefficient regular expressions.
- Use exclusion logic so your trigger does the minimum amount of processing and object creation. For example, you can optimize the following code:

```
var foo = HTTP.cookies;
var bar = HTTP.method;
if (bar === 'POST') {
       // process cookies
}
```

  to use variables only when necessary:

```
var foo;
var bar = HTTP.method;
if (bar === 'POST') {
foo = HTTP.cookies;
   // process cookies
}
```

- Always use `var` to declare variables before using them. For example, use `var foo = HTTP.uri;` instead of `uri = HTTP.uri;`
- Cache the length of arrays before iterating. For example, you can optimize the following code:

```
for (i = 0; i < list.length; i++) { do something; }
```

  to evaluate the length of the list once, as opposed to each time through the loop:

```
var len = list.lengthfor (i = 0; i < len; i++) { do something; }
```

- Use strict equality (triple equals) whenever possible, for example:

```
if (some_thing !== null) { do something; }
```

- Do not put large objects into `Flow.store`. For example, instead of storing all of `DNS.answers`, keep only the information that will be used for future events:

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

- Clear `Flow.store` values when you no longer need them by setting the property to null, for example:

```
if (event === 'DB_REQUEST') {
   Flow.store.stmt = DB.statement;
}
else if (event === 'DB_RESPONSE') {
   Device.metricAddCount('count', 1);
```

```
    Device.metricAddDetailCount('count', Flow.store.stmt, 1);
    Flow.store.stmt = null;
}
```

This also helps to prevent errors in which the trigger reads previously recorded data from `Flow.store`.

> There are separate flow stores for every flow, so how much you can store on each flow store depends on the number and the size of the values being stored in each.

- Conserve CPU cycles by caching created applications that are used multiple times:

```
var myapp = Application('foo');
// now commit like
myapp.commit();
myapp.metricAddCount('custom', 1);
```

- Avoid property lookups when possible, especially when using them more than once. For example, you can optimize the following code:

```
if (HTTP.uri.indexOf('foo') > -1) // blah
else if (HTTP.uri.indexOf('bar') > -1) // blah
```

cache it once and save the property lookups:

```
var uri = HTTP.uri;
if (uri.indexOf('foo') > -1) // blah
else if (uri.indexOf('bar') > -1) // blah
```

- Use `return` instead of `exit()`. The `exit()` global function is supported, but may cause the system to run slowly. For example, use `if (!HTTP.uri) return;` instead of `HTTP.uri || exit();`
- (Version 3.10 and earlier) Instead of using regular expressions, consider using primitive string operations to conserve CPU resources.
- (Version 3.10 and later) The following are no longer valid:
- `Application[...]` syntax
- `for each` syntax
- E4X syntax extensions. Existing uses of new `XML()` are still valid via a native XML class.
- Internal objects that previously compared equal to each other no longer compare equal. For example, `Flow.client.device == new Device(id)` is not valid.