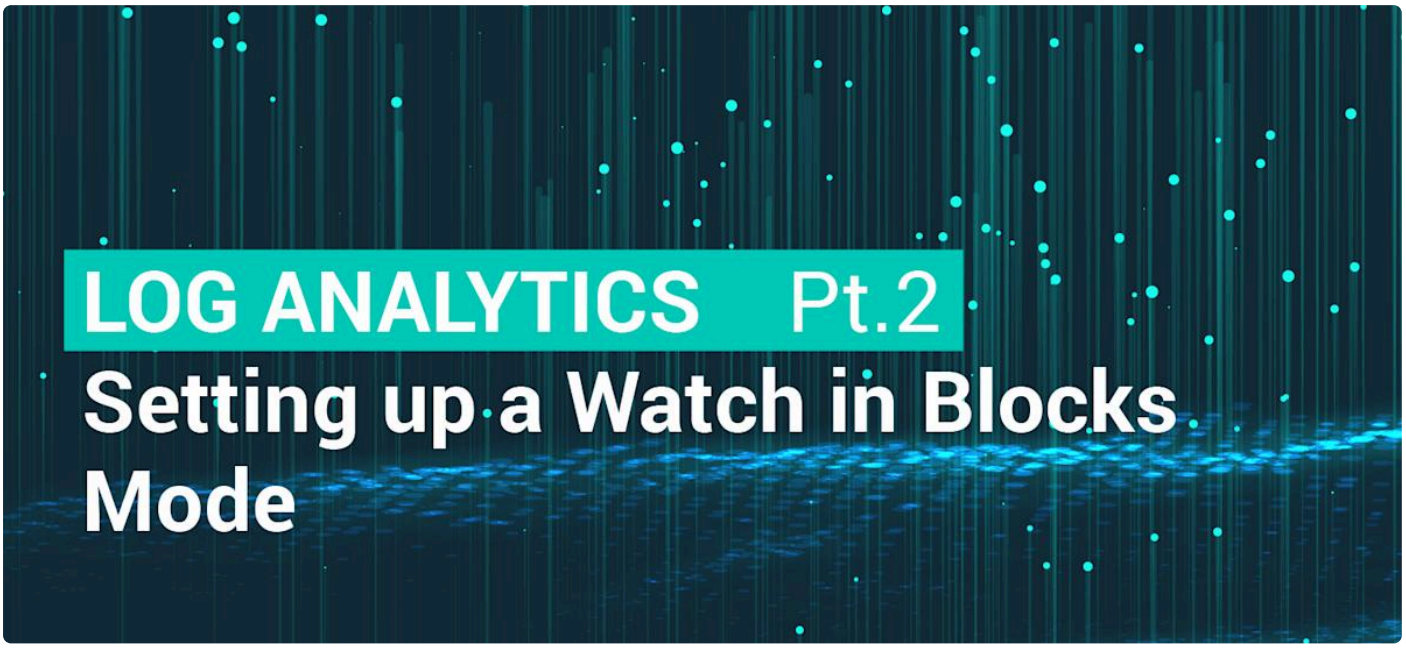
Log Monitoring with Signals Pt.2: Configuring a Watch

Jochen Kressin · 2024-08-01



In our previous article, we imported sample web log data into Elasticsearch and configured Email and Slack notification channels. Today, we'll set up a Watch that periodically scans the log data for errors.

Setup and Objectives

The sample data is stored in an index named *kibanasampledata_logs*. Each log entry is represented by a document like this:

```
{
    "agent": "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)",
    "bytes": 123,
    "clientip": "166.168.152.39",
    "index": "kibana_sample_data_logs",
    "ip": "166.168.152.39",
    "machine": {
        "ram": 3221225472,
        "os": "osx"
    },
    "referer": "http://nytimes.com/success/philip-k-chapman",
    "response": 503,
    "tags": ["success", "security"],
    "timestamp": "2023-07-25T16:11:47.086Z",
    "url": "https://www.opensearch.org/downloads/dataprepper",
    "utc_time": "2023-07-25T16:11:47.086Z"
    ...
}
```

In this scenario, we'll focus on the `response` field, which records the HTTP response code of the server, indicating both successful and failed requests. We'll also use the `timestamp` field to track errors within a specific timeframe.

Our objective is to set up a Watch that runs every 10 minutes. If the number of errors exceeds 20 in the past 10 minutes, a notification should be triggered.

Creating a Watch with Signals Blocks Mode

We could configure and setup everything from scratch using the Signals REST API and a JSON-based Watch definition. However, we'll use the Signals Kibana UI and its user-friendly *Blocks Mode*. Signals Blocks Mode simplifies the process of configuring a Watch by breaking it down into manageable steps.

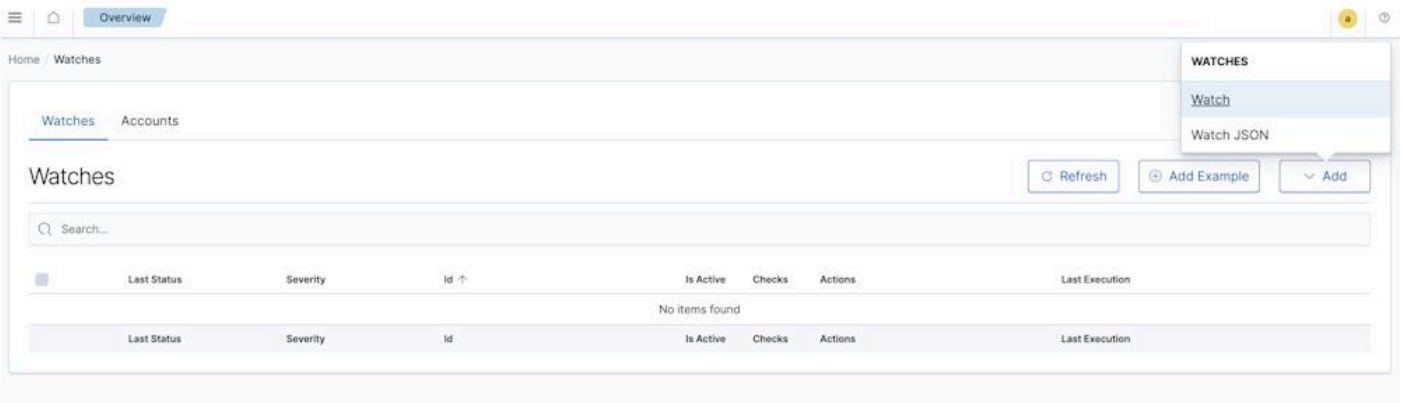
Understanding Blocks Mode

A Watch consists of three main components:

1. **Triggers**: Define when a Watch will be executed. Each Watch must have at least one trigger.
2. **Checks**: Gather data and evaluate it against specific conditions. Each Watch can have multiple checks, executed sequentially. Alerting Plus provides:
   - **Inputs**: Retrieve data from sources like an OpenSearch index or an HTTP service.
   - **Transformations and Calculations**: Process the retrieved data to suit subsequent operations.
   - **Conditions**: Analyze the data using scripts to decide whether to continue execution or halt.
3. **Actions**: Execute specified actions if all prior conditions are met. In our example, we want to send notifications via Email and Slack.

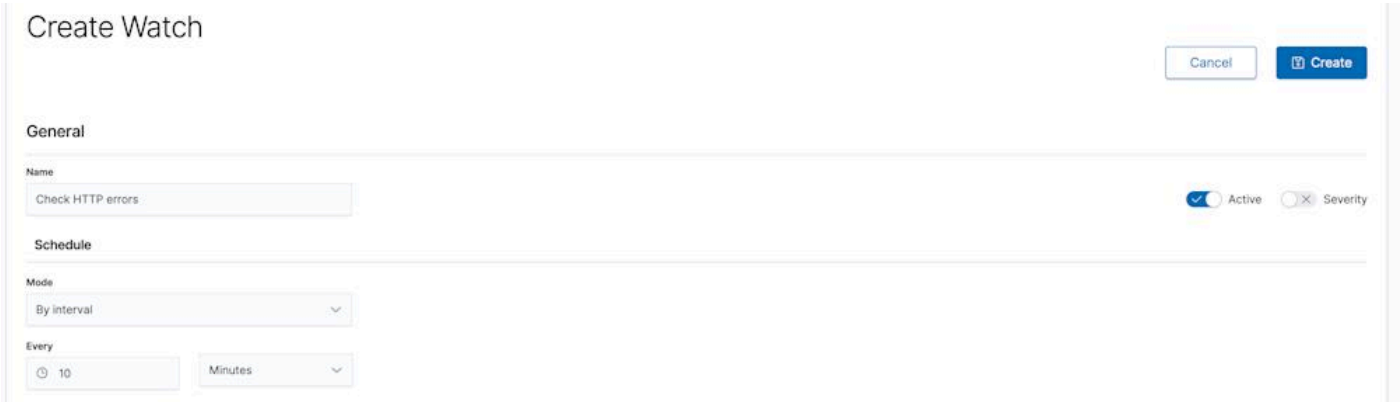
The Signals Dashboards UI guides you through these steps, allowing you to configure your Watch block by block. Let's set up a new Watch.

On the Alerting Plus Watches page, click Add → Watch:



Scheduling the Watch

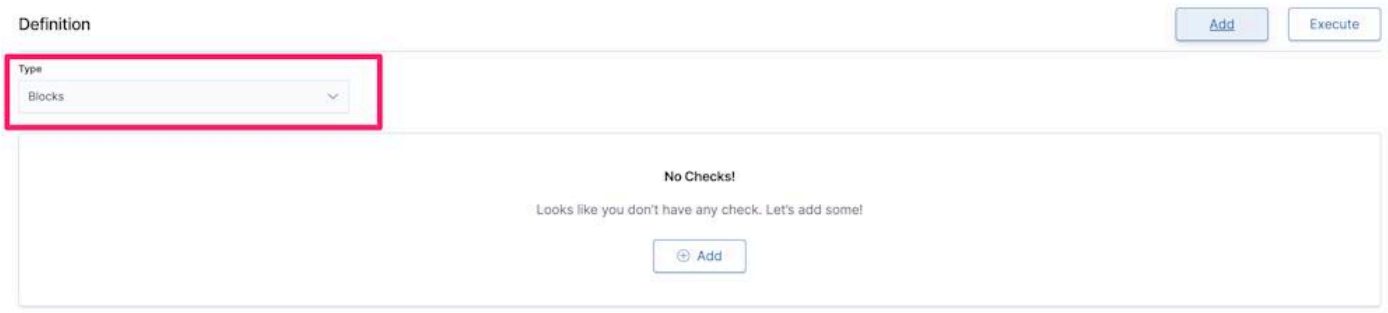
First, name your Watch and define the schedule. We'll set the Watch to run every 10 minutes. Select "By Interval" and enter "Every 10 Minutes":



You can choose from various schedules, such as daily, hourly, or weekly, or use Cron syntax for more advanced scheduling.

Defining the Data Input

Next, specify the data source for your Watch. Since we're using Blocks Mode, select it via *Definition → Type → Blocks*.



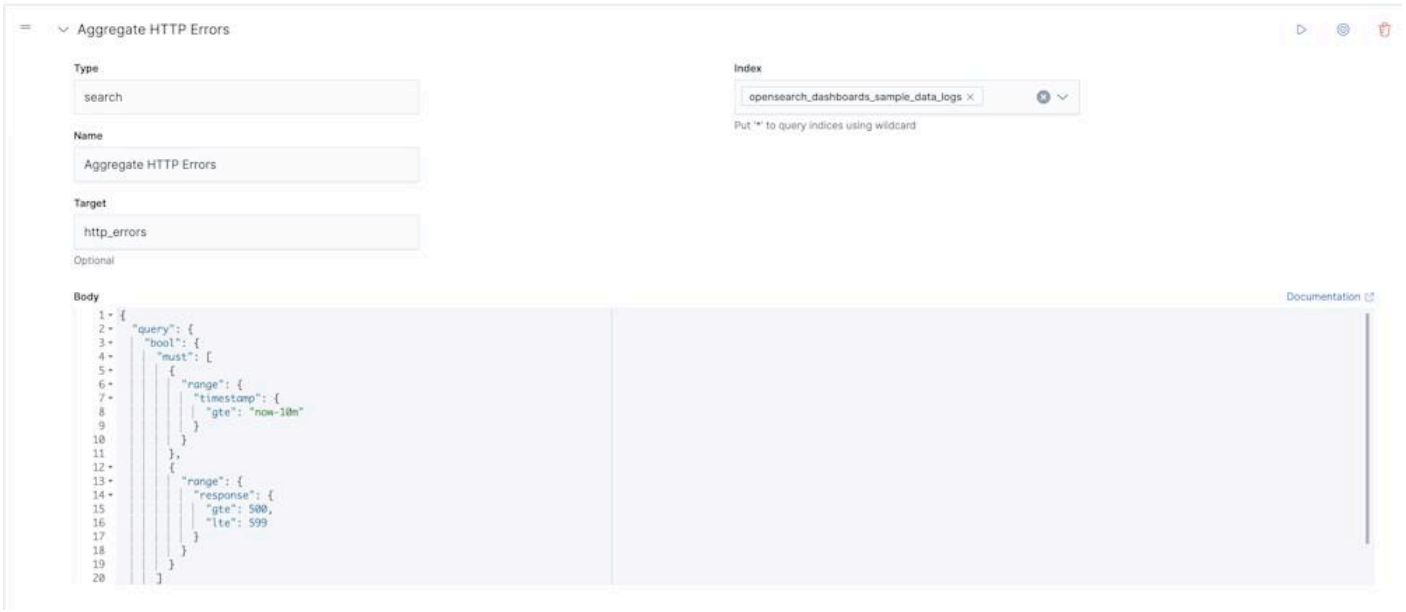
Now, add the first block in the execution chain by clicking on *Add*. This will open a menu with predefined examples to simplify the setup. Typically, the first block is an *Input*, which defines the data source.

Since our data is in an OpenSearch index, we'll use a regular OpenSearch query. Select *Input → Full Text* to add the first block to the execution chain. Then, customize it to fit our needs.

Name the block and select *opensearchdashboardssampledatalogs* as the index. In the *Target* field, choose a unique name to reference this block's result in subsequent blocks. Let's use *http_errors*. Next, define an Elasticsearch query to fetch the data. We want to retrieve documents with an HTTP response code between 500 and 599 from the last 10 minutes. The query looks like this:

```
{
  "query": {
    "bool": {
      "must": [
        {
          "range": {
            "timestamp": {
              "gte": "now-10m"
            }
          }
        },
        {
          "range": {
            "response": {
              "gte": 500,
              "lte": 599
            }
          }
        }
      ]
    }
  }
}
```

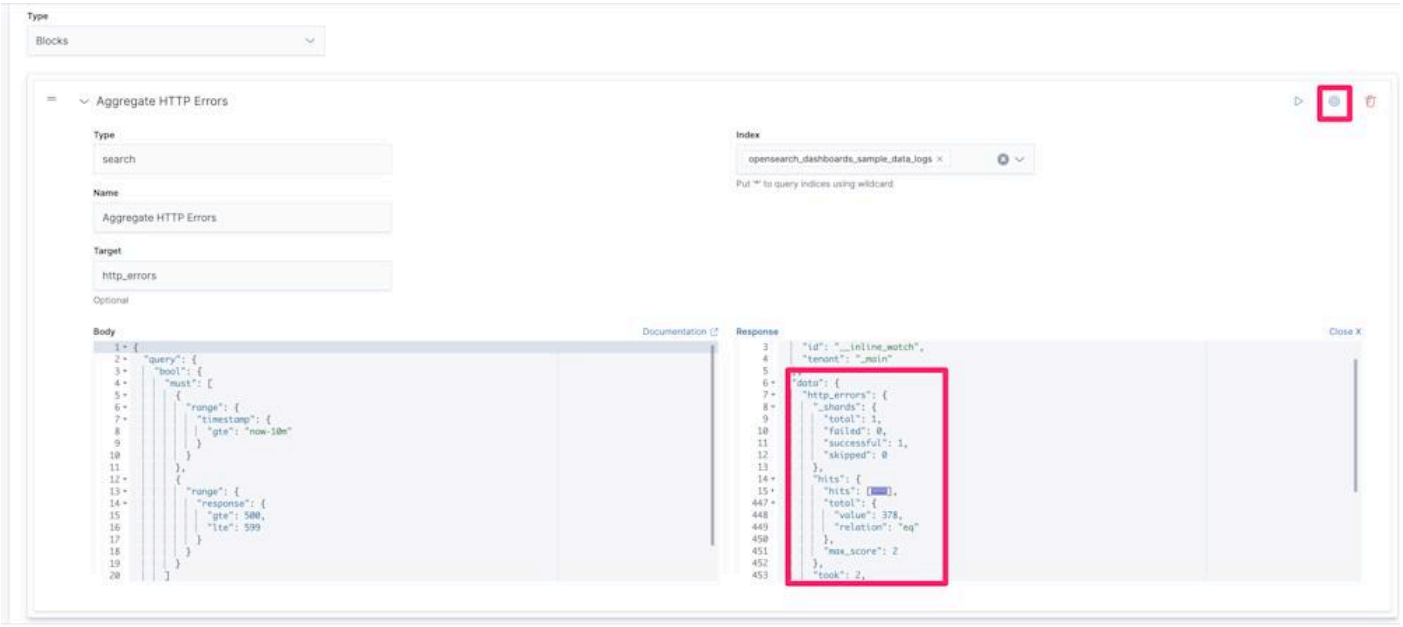
Your Input Block should look like this:



Testing the Query

To verify our query and see the output, Signals provides two options: execute a single block or the entire chain. Currently, we only have one block, so choose "Execute Only This Block":

The JSON result of our query will be displayed on the right side, under the *http_errors* key, as specified in the *Target* field.



Each block can produce data, which is accessible to other blocks using the *Target* key. We will use this key in the next step to define a condition.

Adding a Condition

Next, we'll determine whether the Watch should proceed based on the condition. We want to send a notification if the error count exceeds 20 in the last 10 minutes.

Let's add a Condition Block. We select *Add* again but choose *Condition* this time. We now see two Blocks in our execution chain:

We've already set the query to retrieve documents from the last 10 minutes with error codes between 500 and 599. Now, we'll add a condition to count these documents. If the count is below 20, the Watch execution stops. If it meets or exceeds 20, the execution continues.

Add the condition to check if the document count from the input query exceeds 20:

The condition uses the *data* key to access the watch data. We reference *http_errors* to access the data produced by our input query, then check the *hits.total.value* field returned by the OpenSearch query.

If you're unsure about the data structure, use the Execute button to view the output at each step.

That's it for today. To summarize, we set up a basic Watch using the Signals UI in Blocks Mode. We defined the execution schedule, added an input block to run an OpenSearch query, and specified a condition to halt or continue execution based on the error count.

In the next article, we'll add our configured notification channels to the Watch, so we can send out alerts.

Articles in this series

- Log monitoring with Signals, Pt.1: Setting up Notification Channels
- Log monitoring with Signals, Pt.2: Setting up a Watch in Blocks Mode (this article)
- Log monitoring with Signals, Pt.3: Sending Notifications
- Log monitoring with Signals, Pt.4: Implementing Escalation Levels

---

This article was published on the Search Guard blog: https://search-guard.com/blog/log-monitoring-signals-pt-2-configuring-a-watch/