Verifying Data Ingestion is Continuous in Sumo Logic

dev.classmethod.jp/articles/sumologic-data-ingest-monitoring

酒井剛

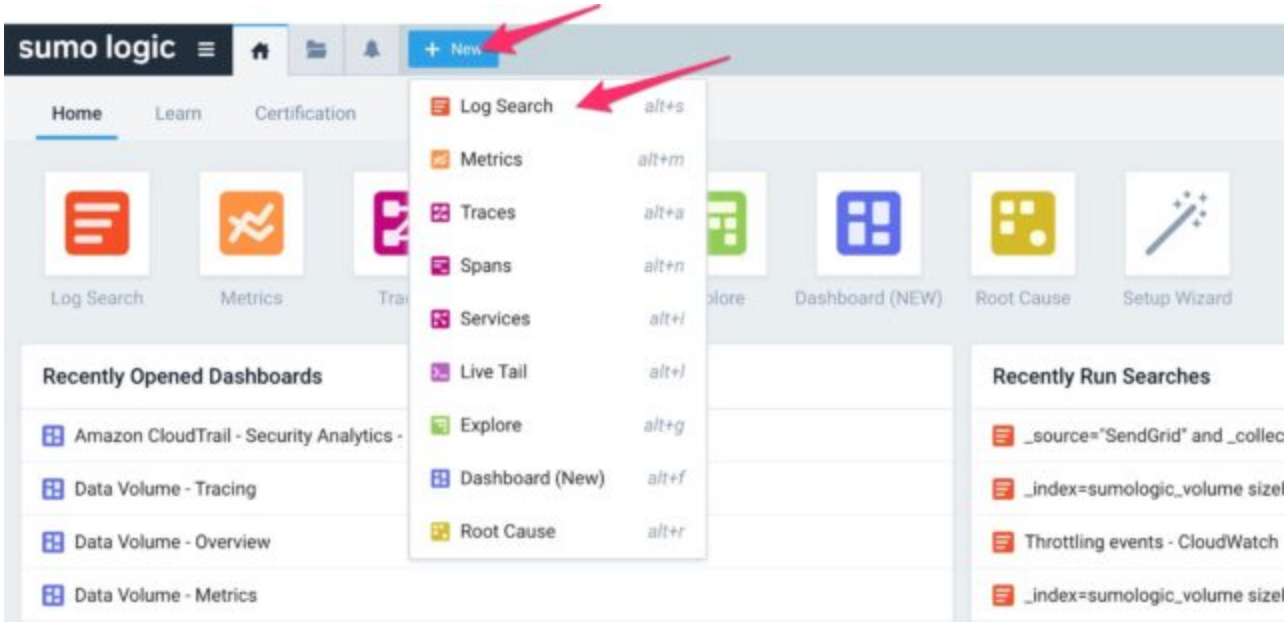
January 20, 2023

Today, as the title suggests, I would like to introduce how to implement a system to monitor whether data ingestion into Sumo Logic is being performed correctly.

This article is based on the official Sumo Logic documentation.

Sudden answer

Open the Sumo Logic query screen,

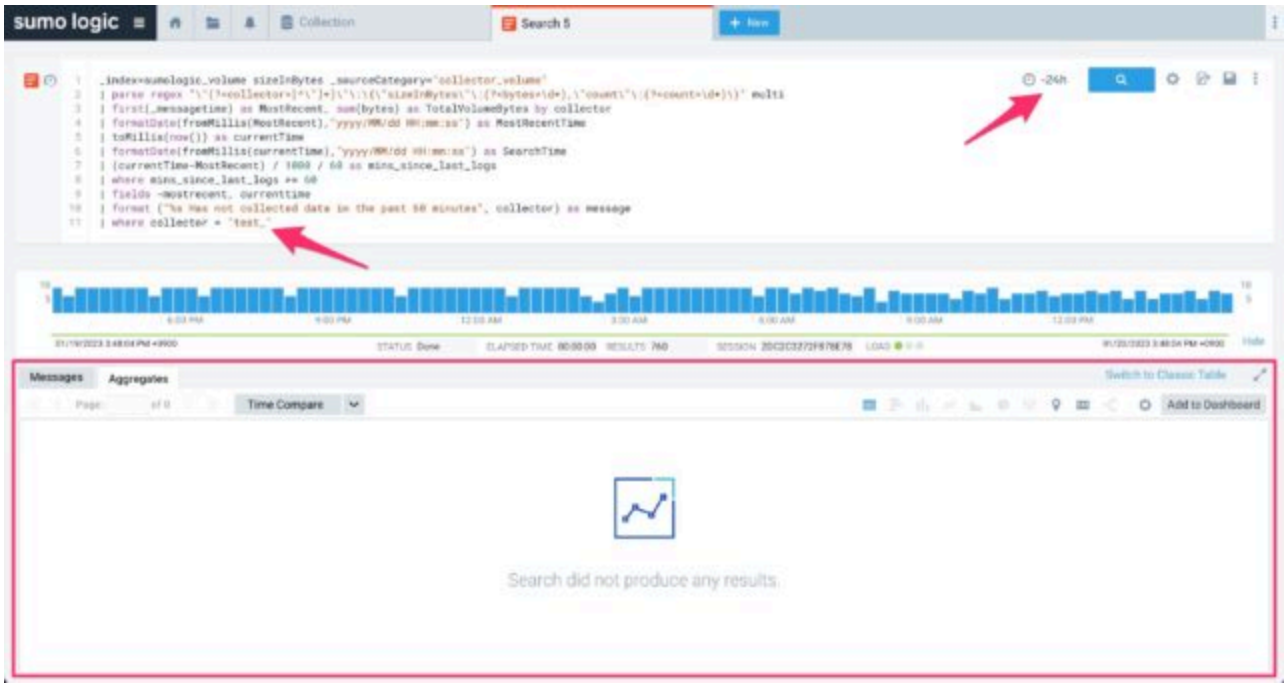


Create a query like the following:

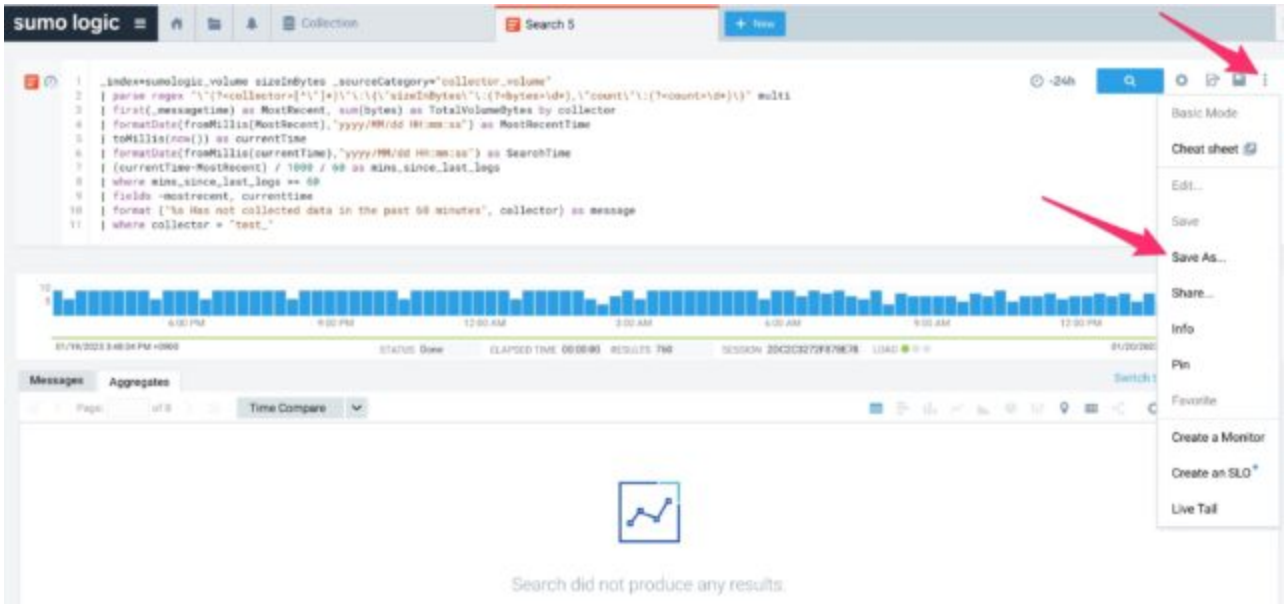
```
_index=sumologic_volume sizeInBytes _sourceCategory="collector_volume"
| parse regex "\"(?<collector>[^\"]+)\"\:\{\"sizeInBytes\"\:(?
<bytes>\d+),\"count\"\:(?<count>\d+)\}" multi
| first(_messagetime) as MostRecent, sum(bytes) as TotalVolumeBytes by collector
| formatDate(fromMillis(MostRecent),"yyyy/MM/dd HH:mm:ss") as MostRecentTime
| toMillis(now()) as currentTime
| formatDate(fromMillis(currentTime),"yyyy/MM/dd HH:mm:ss") as SearchTime
| (currentTime-MostRecent) / 1000 / 60 as mins_since_last_logs
| where mins_since_last_logs >= 60
| fields -mostrecent, currenttime
| format ("%s Has not collected data in the past 60 minutes", collector) as message
| where collector = <collector名>
```

<collector名>On the last line, enter the name of the collector you want to monitor to see if data collection has stopped.

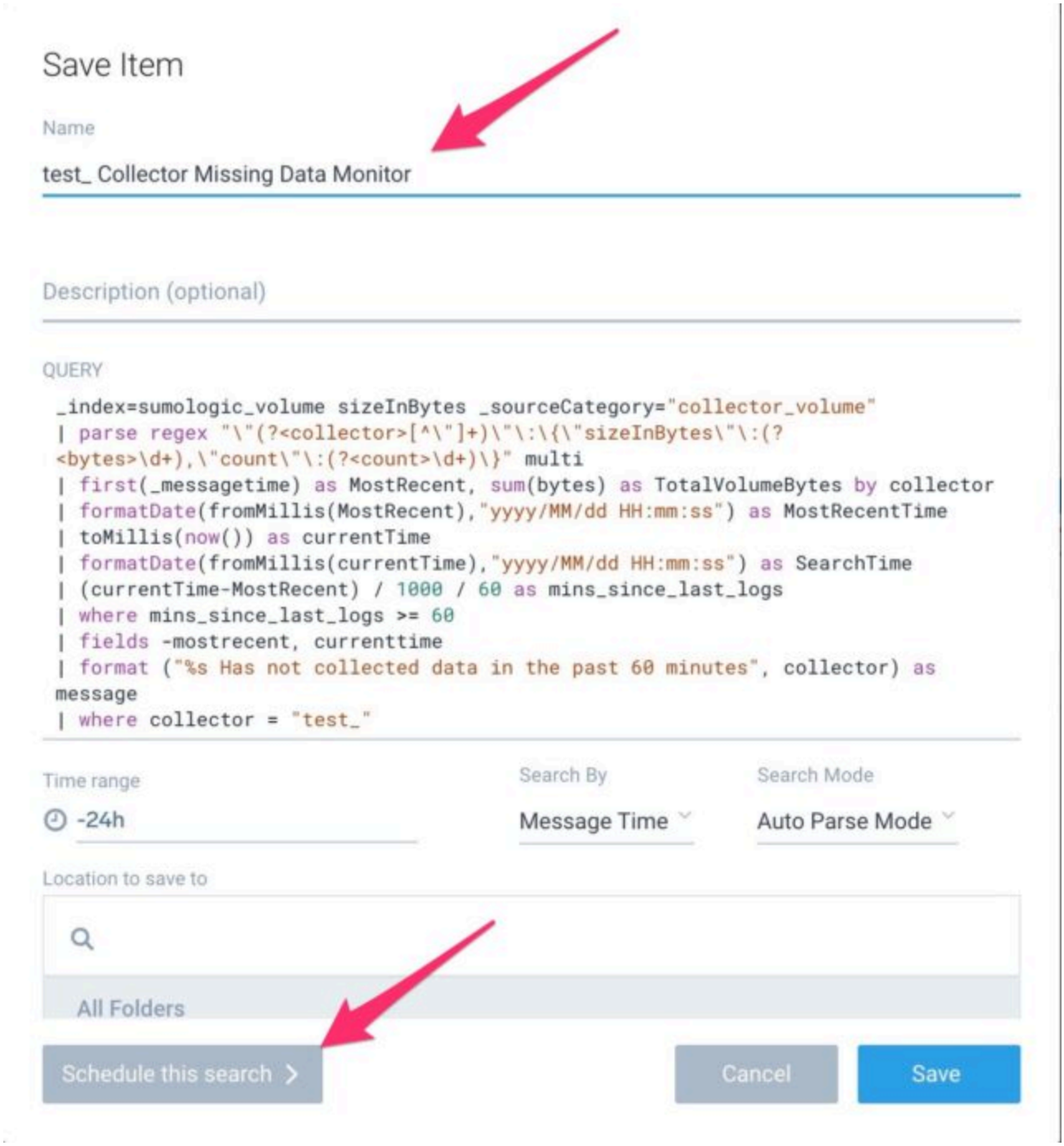
In this example, we will check whether there is any missing data in the collector named "test_". A search range of about 24 hours would be appropriate.
If there is no output in the "Aggregates" section after executing the query, this means there is no missing data.



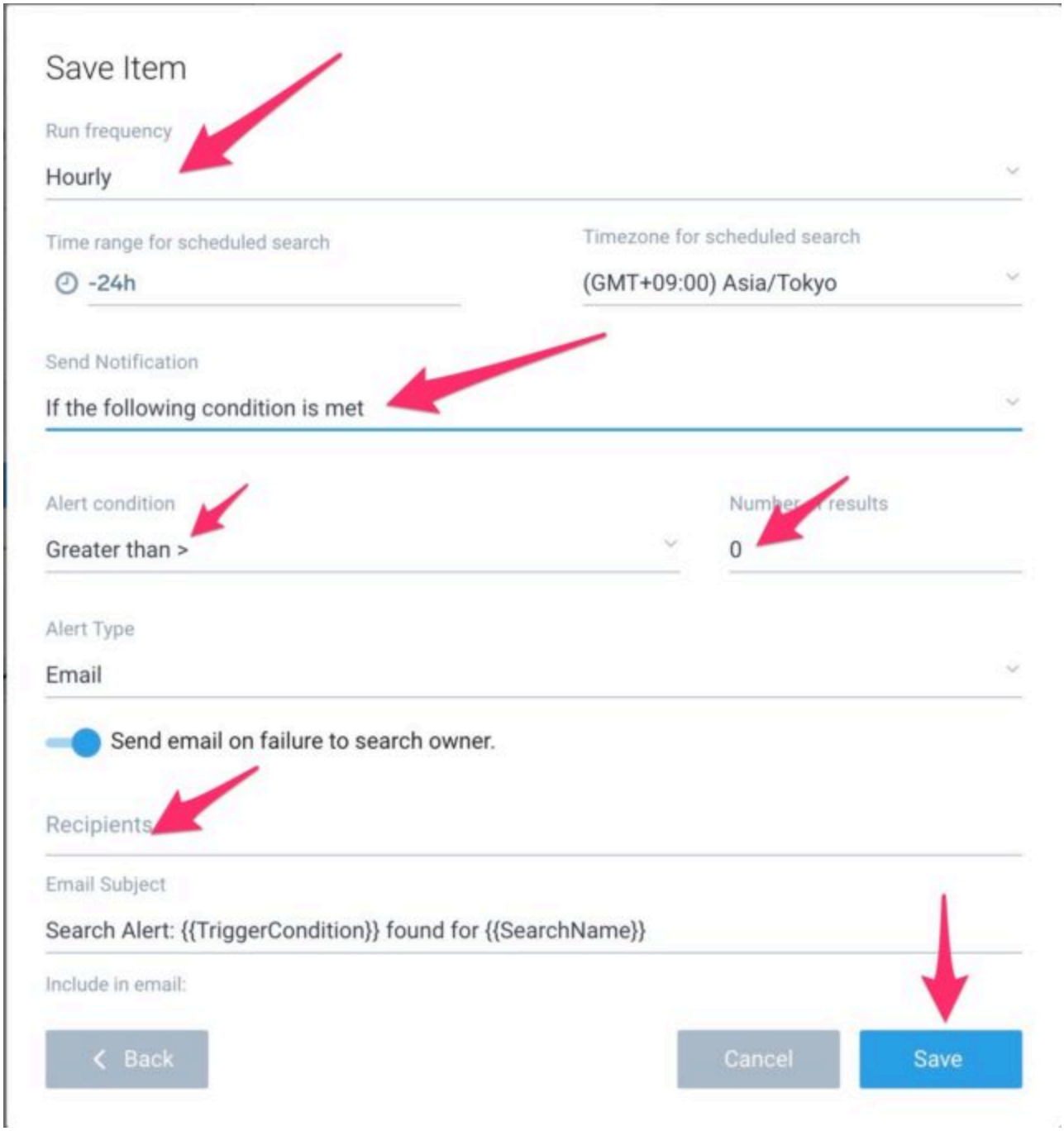
We'll set this query as a monitoring alert.
Click "Save As..." in the three dots in the top right.



Enter the name of the scheduled search you want to set as an alert and click "schedule this search."



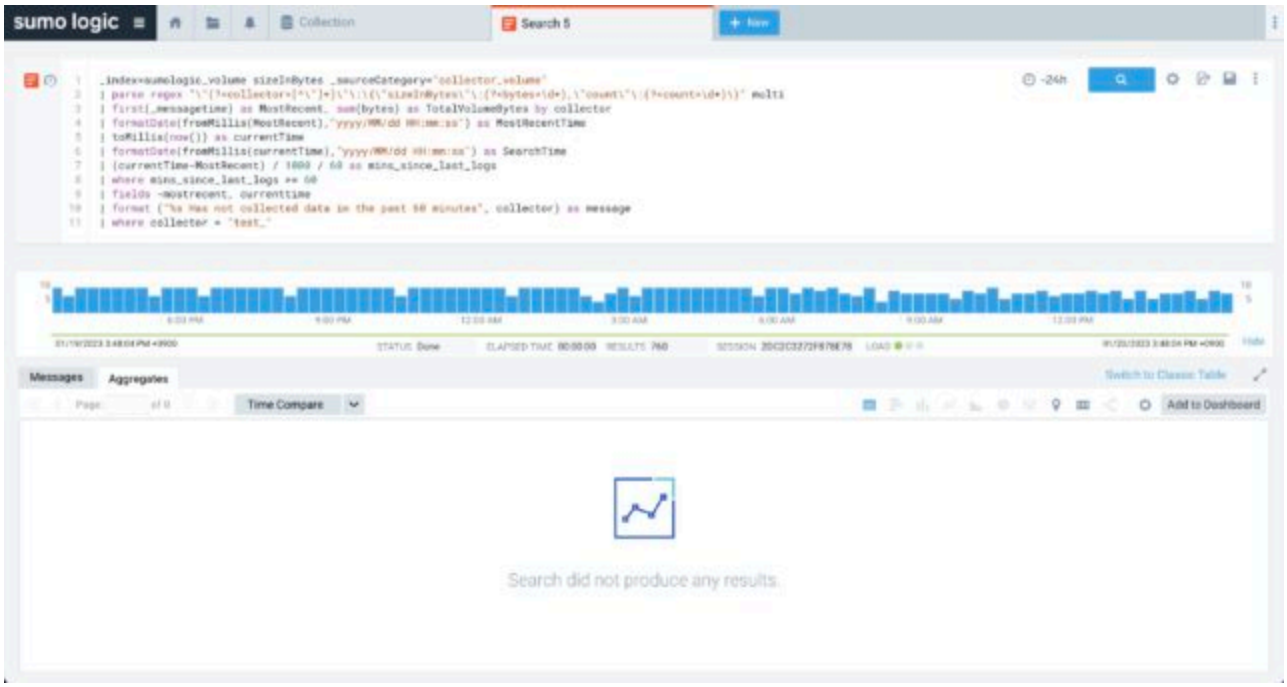
A good run frequency would be "Hourly." Set it to send an alert when a query result greater than 0 is returned, and configure the email address to which the alert should be sent.



Now the monitoring is complete. You have achieved what you wanted.

Query Explanation

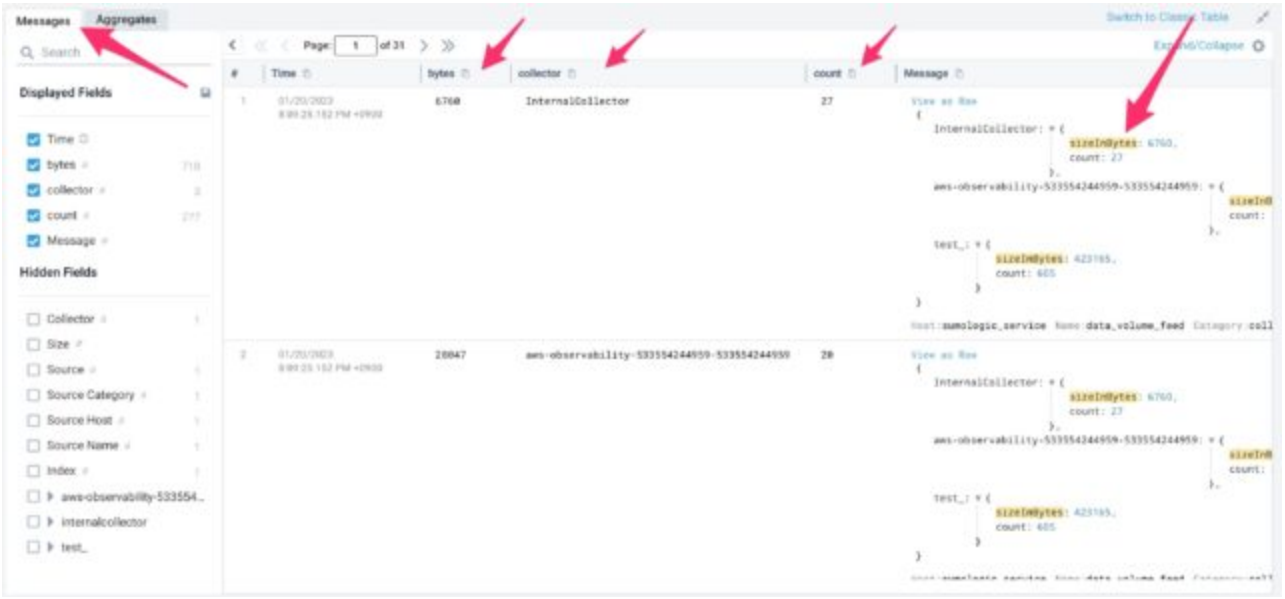
From here, we will explain the query we created for monitoring earlier, going back to the output results of the query we just created.



If we look at the query again in detail, we can see that the "where" operator is a conditional search.
We can see that the condition is that the log must be more than 60 minutes old since the last log output. We also need to match the collector name we want to monitor, which we set up initially.

```
_index=sumologic_volume sizeInBytes _sourceCategory="collector_volume"
| parse regex "\"(?<collector>[^\"]+)\"\:\{\"sizeInBytes\"\:(?
<bytes>\d+),\"count\"\:(?<count>\d+)\}" multi
| first(_messagetime) as MostRecent, sum(bytes) as TotalVolumeBytes by collector
| formatDate(fromMillis(MostRecent),"yyyy/MM/dd HH:mm:ss") as MostRecentTime
| toMillis(now()) as currentTime
| formatDate(fromMillis(currentTime),"yyyy/MM/dd HH:mm:ss") as SearchTime
| (currentTime-MostRecent) / 1000 / 60 as mins_since_last_logs
| where mins_since_last_logs >= 60
| fields -mostrecent, currenttime
| format ("%s Has not collected data in the past 60 minutes", collector) as message
| where collector = <collector名>
```

Next, click Messages to see the search results, including those that did not match the query criteria. You
can then see the fields parsed by the "parse" operator and the raw log messages.



"parse regex" parses raw logs using regular expressions to find collector names such as "InternalCollector" and "aws-observability-533554244959-533554244959" into a field called " **collector** ". Similarly, it parses the "bytes" and "count" parts.
By parsing them as fields, they can be used for various processes such as calculations and aggregations in subsequent query statements.

```
_index=sumologic_volume sizeInBytes _sourceCategory="collector_volume"
| parse regex "\"(?<collector>[^\"]+)\"\:\{\"sizeInBytes\"\:(?
<bytes>\d+),\"count\"\:(?<count>\d+)\}" multi
| first(_messagetime) as MostRecent, sum(bytes) as TotalVolumeBytes by collector
| formatDate(fromMillis(MostRecent),"yyyy/MM/dd HH:mm:ss") as MostRecentTime
| toMillis(now()) as currentTime
| formatDate(fromMillis(currentTime),"yyyy/MM/dd HH:mm:ss") as SearchTime
| (currentTime-MostRecent) / 1000 / 60 as mins_since_last_logs
| where mins_since_last_logs >= 60
| fields -mostrecent, currenttime
| format ("%s Has not collected data in the past 60 minutes", collector) as message
| where collector = <collector名>
```

In the subsequent aggregation process, the timestamp of the most recent message is obtained and the time lag from the current time is checked.
This aggregation allows us to check whether there are any collectors that have not output logs for more than 60 minutes by using "where mins_since_last_logs >= 60".

```
_index=sumologic_volume sizeInBytes _sourceCategory="collector_volume"
| parse regex "\"(?<collector>[^\"]+)\"\:\{\"sizeInBytes\"\:(?
<bytes>\d+),\"count\"\:(?<count>\d+)\}" multi
| first(_messagetime) as MostRecent, sum(bytes) as TotalVolumeBytes by collector
| formatDate(fromMillis(MostRecent),"yyyy/MM/dd HH:mm:ss") as MostRecentTime
| toMillis(now()) as currentTime
| formatDate(fromMillis(currentTime),"yyyy/MM/dd HH:mm:ss") as SearchTime
| (currentTime-MostRecent) / 1000 / 60 as mins_since_last_logs
| where mins_since_last_logs >= 60
| fields -mostrecent, currenttime
| format ("%s Has not collected data in the past 60 minutes", collector) as message
| where collector = <collector名>
```

* Only collectors that are constantly importing logs under normal circumstances can be monitored. Collectors that have already stopped or that have not imported logs for the time set in the threshold (e.g., 60 minutes) cannot be specified as targets.

Other ways to monitor log ingestion

Up until now, we have been checking whether log output has stopped for each collector, but there may be cases where you want to check by source category.
In that case, you can change the query to the following to view by source category.

```
_index=sumologic_volume sizeInBytes _sourceCategory="sourceCategory_volume"
| parse regex "\"(?<sourceCategory>[^\"]+)\"\:\{\"sizeInBytes\"\:(?
<bytes>\d+),\"count\"\:(?<count>\d+)\}" multi
| first(_messagetime) as MostRecent, sum(bytes) as TotalVolumeBytes by
sourceCategory
| formatDate(fromMillis(MostRecent),"yyyy/MM/dd HH:mm:ss") as MostRecentTime
| toMillis(now()) as currentTime
| formatDate(fromMillis(currentTime),"yyyy/MM/dd HH:mm:ss") as SearchTime
| (currentTime-MostRecent) / 1000 / 60 as mins_since_last_logs
| where mins_since_last_logs >= 60
| fields -mostrecent, currenttime
| format ("%s Has not collected data in the past 60 minutes", sourceCategory) as
message
| where sourceCategory = <sourceCategory名>
```

In addition, source categories are generally designed with a naming convention, so you can monitor multiple source categories by specifying an asterisk such as "aws/*".

summary

These are the tips for monitoring whether data is being imported into Sumo Logic correctly.
I hope this blog will be helpful to someone.