Hands-on Labs: Using Sumo Logic

Search, Analyze, Visualize and Monitor

Introduction

Using Sumo Logic
These labs will provide you hands-on experience with Sumo Logic, where you will learn basic and advanced search operators to analyze your logs and metrics.

Please Note: These labs can be completed as a standalone item; however, they are designed to complement our Using Sumo Logic webinar. If you haven't done so, for your benefit, please register and attend this webinar before completing these labs.

Accessing the Training Environment
These labs are meant to be done in our Training environment using curated sample data. However, you are welcome to use your own environment by editing the query samples to fit your data and metadata.

To access the Sumo Logic Training environment:
1. Go to: https://service.sumologic.com
2. Use the following credentials:
   User: training+labs@sumologic.com
   Password: Sum0Labs! (DO NOT copy/paste, as PDFs add extra characters)

NOTE:
Because of data variability and timing, your results might not exactly match those listed in these pages.
Search and Analyze

Lab 1 - Search Basics: Metadata and Keywords
In this lab, you will learn the use of metadata and keywords to narrow your search scope and improve performance.

1. Search for all messages with _sourceCategory=Labs/Apache/Access for the last 15 minutes.
   _sourceCategory=Labs/Apache/Access

2. To identify client errors, narrow your search for only those with "404".
   _sourceCategory=Labs/Apache/Access AND 404

3. Use Boolean logic to search for log lines matching "Error" or "check" or "checkout".
   _sourceCategory=Labs/Apache/Access AND (Error OR check*)

4. In a new tab, search for messages with _sourceCategory=Labs/Apache/Error for the last 45 minutes (-45m).
   _sourceCategory=Labs/Apache/Error

5. Search across both Labs/Apache/Access and Labs/Apache/Error Source Categories:
   _sourceCategory=Labs/Apache/Error OR _sourceCategory=Labs/Apache/Access

6. Search across both Labs/Apache/Access and Labs/Apache/Error Source Categories using wildcards:
   _sourceCategory=Labs/Apache/*

7. Run a Live Tail search for Labs/Apache/Access for messages containing 404s. Notice Live Tail sessions do support wildcards searches.
   _sourceCategory=Labs/Apache/Access 404

QUIZ: True or False
1. Keywords are case-sensitive
2. AND is implicit and OR is explicit
3. Keywords and metadata can use wildcards for Live tail
Lab 2 - Simple Parsing, Grouping, and Filtering
Learn basic operators to parse and group your search results. Future labs will go deeper in each area.

1. Search Apache logs (Access and Error logs) for the last 15 minutes and count results by _source.
   _sourceCategory=Labs/Apache/*
   | count by _source

2. Update the query to parse the ip address using parse regex and count by ip address.
   _sourceCategory=Labs/Apache/*
   | parse regex "(?<ip_address>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})"
   | count by ip_address

3. Filter out those with a count less than 2500. Display your results in a pie chart.
   _sourceCategory=Labs/Apache/*
   | parse regex "(?<ip_address>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})"
   | count by ip_address
   | where _count < 2500

4. In a new query, identify the top 10 source IP addresses by bandwidth usage. This will require parsing the byte sizes as well as the source IPs.
   _sourceCategory=Labs/Apache/Access
   | parse regex "(?<ip_address>\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3})"
   | parse " 200 * " as size
   | sum(size) as total_bytes by ip_address
   | top 10 ip_address by total_bytes

Bonus: Find the average size by source ip.
Lab 3 - Parsing Options
Parsing your logs allow you to provide structure to your messages, identifying the fields that are meaningful to you.

1. Use the `json auto` option to automatically parse all fields from AWS CloudTrail messages
   ```
   _sourceCategory=Labs/AWS/CloudTrail
   | json auto
   ```

2. In the previous example, notice all the parsed fields shown in the Field Browser. You can now use the parsed `awsregion` field to count messages by region.
   ```
   _sourceCategory=Labs/AWS/CloudTrail
   | json auto
   | count by awsregion
   ```

3. The `nodrop` option for the parse operator allow users to include messages in your results that do not meet the pattern criteria. Run a search for Apache Error logs for the last 15 minutes and notice that not all messages have a client ip.
   ```
   _sourceCategory=Labs/Apache/Error
   ```

4. Run the same search, but this time, parse the client ip. Notice how all other messages without the `[client *)` pattern are dropped.
   ```
   _sourceCategory=Labs/Apache/Error
   | parse "[client *)" as client_ip
   ```

5. Add the nodrop option. Notice how non-matched messages are kept, with an empty client_ip. Notice how a nodrop combined with additional parse statements can allow you to parse logs of varying patterns/formats.
   ```
   _sourceCategory=Labs/Apache/Error
   | parse "[client *)" as client_ip nodrop
   | parse "mod_log_sql: *" as message
   ```
6. Filter those parsed by one or the other statement by using the isEmpty, isBlank or isNull operators.

   ```log
   _sourceCategory=Labs/Apache/Error
   | parse "[client *]" as client_ip nodrop
   | parse "mod_log_sql: *" as message
   | where isBlank(client_ip)
   ```

7. The `parse field` option allows you to do further parsing on an already extracted field. In this example, we want to identify the top 5 committers in GitHub. Search committers in the last 30 days, and parse their email address.

   ```log
   _sourceCategory=Labs/Github and "committer"
   | parse ""email":":"*" as email
   ```

8. Now use the `parse field` option to further parse the email address into user and domain. Lastly, count by user and identify the top 5 committers.

   ```log
   _sourceCategory=Labs/Github and "committer"
   | parse ""email":":"*" as email
   | parse field=email "*@*" as users, domain
   | count by users
   | top 5 users by _count
   ```

9. The `parse multi` option allows you to extract multiple occurrences of the same pattern within one message. By default, parse only extracts the first occurrence. First, search the Snort data and extract the ip address.

   ```log
   _sourceCategory=labs/snort
   | parse regex "(?<ip_address>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})"
   ```

10. Now use `parse multi` and notice how each message is repeated for each occurrence of an ip address, allowing you to do accurate counts.

    ```log
    _sourceCategory=labs/snort
    | parse regex "(?<ip_address>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})" multi
    ```

11. **Field Extraction Rules** extract fields at the time the log messages are ingested. You can see all FERs available (and their details) under Manage Data → Settings → Field Extraction Rules. Taking advantage of the Apache Access rule, run a search to identify the count of 404s by source ip.

    ```log
    _sourceCategory=Labs/Apache/Access and status_code=404
    | count by src_ip
    ```

---

**QUIZ: True or False**

1. `csv, json, split, keyvalue` are all parsing operators.
2. Once a field has been parsed, it cannot be parsed any further.
3. Fields parsed by the Field Extraction Rules are available in the Field Browser.
Lab 4 - Conditional Operations

Conditional operators allow you to do if/then operations.

1. Write a query that scans through Labs/Apache/Access logs for the last 60 minutes, and returns the number of successful HTTP requests returned (where status_code=200), versus the number of error HTTP requests returned (where status_code=404).

```ruby
/_sourceCategory=Labs/Apache/Access
| parse "HTTP/1.1" * " as status_code
| if(status_code=200, 1, 0) as successes
| if(status_code=404, 1, 0) as client_errors
| sum(successes) as success_cnt, sum(client_errors) as client_errors_cnt
```

2. Write a query that scans through Labs/Apache/Access logs for the last 60 minutes, and returns the number of client errors (where status_code matches 4*), versus the number of server errors (where status_code matches 5*).

```ruby
/_sourceCategory=Labs/Apache/Access
| parse "HTTP/1.1" * " as status_code
| if(status_code matches "4\*", 1, 0) as client_err
| if(status_code matches "5\*", 1, 0) as server_err
| sum(server_err) as server_errors_cnt, sum(client_err) as client_errors_cnt
```

Lab 5 - Filtering Operations

We've used the `where` operator to filter out results. But what if we want to filter out by some sub-query? We can do this using the `filter` operator.

1. In this lab, we're going to inspect our CloudTrail data. We'd like to identify the most common events (top 10) across the various AWS regions. A good start would be to do a count by eventname and awsregion.

```ruby
/_sourceCategory= Labs/aws/cloudtrail
| json auto
| count by eventname, awsregion
| sort by _count desc
```

2. This is a good start, but the count is broken by eventname and awsregion. We need to aggregate all eventname counts across all regions, and select only those eventnames that have the highest count. For this, let's create a second query to help us identify those eventnames with the highest count.

```ruby
/_sourceCategory= Labs/aws/cloudtrail
| json auto
| count by eventname
| top 10 eventname by _count
```

3. Now let's use the appropriate Filter operator syntax to put these 2 queries together. In essence, we are taking the results from the first query, and filtering them with the results from the second query.

```ruby
/_sourceCategory= Labs/aws/cloudtrail
| json auto
| count by awsregion, eventname
| filter eventname in
   (sum(_count) by eventname
    | top 10 eventname by _sum)
```
4. Lastly, a transpose to format the results turns the list into an easy to read table.

```plaintext
_sourceCategory= Labs/aws/cloudtrail
| json auto
| count by awsregion, eventname
| filter eventname in
  (sum(_count) by eventname
  | top 10 eventname by _sum)
| transpose row eventname column awsregion
```

## Lab 6 - Plotting Clients on a Map

Learn to use the [geo lookup](http://example.com) operator to plot the location of your incoming requests on a map.

1. Run a search for all Apache Access logs and parse the ip address.
2. Use the ip address to search for the latitude and longitude coordinates
   ```plaintext
   _sourceCategory=Labs/Apache/Access
   | parse "* - -" as client_ip
   | lookup latitude, longitude from geo://location on ip=client_ip
   | count by latitude, longitude
   | sort _count
   ```
3. Map your results.
4. Bonus: Edit your map to only show results for the US (country_code="US"). This will require additional fields other than just latitude and longitude. Check out the documentation for other fields available.

### Optional: Test Your Knowledge

Using the extracted fields from the Apache Access Field Extraction Rule, build the following queries:

1. **Count Number of Messages by Method**
   a. Scope: _sourceCategory=Labs/Apache/Access
   b. Time Range: Last 45 Minutes (-45m)
   c. Time slice by 1 minute
   d. Count by method and timeslice

2. **Count Number of 404 Error Messages by Method**
   a. Scope: _sourceCategory=Labs/Apache/Access and only 404 messages
   b. Time Range: 15 minutes, but starting 30 Minutes ago (-45m -30m)
   c. Time slice by 1 minute
   d. Count by method and timeslice

3. **Count Successes versus Failures**
   a. Scope: _sourceCategory=Labs/Apache/Access
   b. Time Range: Nov 25 8:00 AM - Nov 25 8:45am
   c. Count 2* messages as Successes and 4* messages as Failures
   d. Sum Successes and Failures
Lab 7 - Format Results
In order to chart your data, sometimes you will need to format results to fit your desired chart needs.

1. Create a chart that displays error codes by minute, over the last 15 minutes. Transposing your original results will allow you to create a table with time in your rows and status code in your columns.

   _sourceCategory=Labs/Apache/Access
   | parse "HTTP/1.1\" * " as status_code
   | timeslice by lm
   | count by _timeslice, status_code
   | transpose row _timeslice column status_code

Lab 8 - Calculating Changes and Moving Averages
In this lab, learn about diff and smooth to calculate changes in your data.

1. Calculate difference in number of requests over the last 15 minutes:
   _sourceCategory=Labs/Apache/Access
   | timeslice 1m
   | count by _timeslice
   | sort by _timeslice asc
   | diff _count

2. Calculate moving average of 404 occurrences over the last 15 minutes.
   _sourceCategory=Labs/Apache/Access and status_code=404
   | timeslice 1m
   | count as error_count by _timeslice
   | sort by _timeslice asc
   | smooth error_count as rolling_avg

Bonus: Plot them on a line bar combo chart.
Advanced Analytics

Lab 9 - Find the "needle in the haystack"
Explore the functionality of LogReduce, which allows you to distill unique messages from the noise by identifying recurring Signatures in your data.

1. Search for all messages that have the word "error" across all your data. Then click on LogReduce to get a summarized view of all messages.
   error | logreduce

2. In a new search, run LogReduce on your Snort security data to identify unusual activity (i.e. intrusions).
   _sourceCategory=labs/snort
   | logreduce

3. Sort your results by count to identify those that happen only once. Click on the count (1) to view the unusual message. Now click on the host to view surrounding messages to identify the context of the intrusion.

<table>
<thead>
<tr>
<th>#</th>
<th>Time</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Host: 54.224.66.85 Name: Http Input Category: Labs/security/snort</td>
</tr>
</tbody>
</table>

   +/- 1 Minute < Surrounding Messages
   +/- 5 Minutes
   +/- 10 Minutes

Lab 10 - Compare Activity from Different Periods
Explore the functionality of LogCompare, which allows you to compare log activity from two different time periods, providing you insight on how your current time compares to a baseline. In this case, use LogCompare to identify when signature messages deviate by more than 25% from the baseline.

1. First, review summarized signatures for messages with 404 status in the last 15 minutes (Use LogReduce)
   _sourceCategory=Labs/Apache/Access and status_code=404
   | logreduce

2. Now use LogCompare to run a summarized query for a baseline 24 hours ago (Click on LogCompare button)
   _sourceCategory=Labs/Apache/Access and status_code=404
   | logcompare timeshift -24h

3. To view only those results where Delta Percentage is more than 25%, add a where clause for _deltaPercentage.
   _sourceCategory=Labs/Apache/Access and status_code=404
   | logcompare timeshift -24h
   | where abs(_deltaPercentage) > 25

4. To view results where there is a new Signature in the current time period, add a where clause for _isNew:
   _sourceCategory=Labs/Apache/Access and status_code=404
   | logcompare timeshift -24h
   | where (_isNew)
Lab 11 - Identify "out of the ordinary" Events
Explore the functionality of the **outlier** operator, which allows you to identify events outside of a threshold.

1. Search your Labs/Apache/Access logs looking for status_code 404 for the last 60 minutes.
2. Slice your 60 minutes by 1-minute increments and count your 404 status codes by timeslice.
3. Identify outliers outside of the 3 standard deviations and plot results on a line graph.
4. Test the same with outliers outside of 1 standard deviation. Chart on a line graph.

```plaintext
_sourceCategory=Labs/Apache/Access status_code=404
| timeslice 1m
| count(status_code) as error_count by _timeslice
| outlier error_count window=10, consecutive=1, threshold=3, direction=+-
```

5. Bonus: Edit settings for window, consecutive, threshold, and direction to see the change in behaviour

Lab 12 - Identify Future Trends
Use the **predict** operator to understand future trends based on your existing data.

1. Search your Labs/Apache/Access logs looking for status_code 404 for the last 60 minutes.
2. Slice your 60 minutes by 1-minute increments and count your 404 status codes by timeslice.
3. Predict your future trend of 404s in 1 minute increments and plot results on a line graph.

```plaintext
_sourceCategory=Labs/Apache/Access status_code=404
| timeslice 1m
| count(status_code) as error_count by _timeslice
| predict error_count by 1m
```

Lab 13 - Analyze Related Log Messages
The **transaction** operator, allows you to analyze related sequences of messages based on a unique transaction identifier such as a SessionID or IP Address. Transaction uses the unique identifier you specify to group related messages together and arrange them based on states which you define. This lab uses transaction to track the states a user hits within an e-commerce website called ecommark. This will allow you to conduct analysis on how users are interacting with the e-commerce website.

1. Run a search for all ecommark log messages (_sourceCategory=Labs/ecommark) for the last 24 hours.
2. Notice that each message contains details indicating the IP address and the state that was triggered. For example, "Order Shipped" and "GET /checkout/confirmation" are two possible states.
3. Below is an example search using the transaction operator to capture the some possible states using IP address as the unique identifier. Copy this query into your log search window and use a timeframe of "Last 1 hour":

```plaintext
```

Last Updated: October 4 2018
4. Now let’s understand how often each state was hit by using the count operator. Add a count by fromstate, tostate.

5. To chart these aggregate results, select the flowchart to see a Sanke Diagram of the transaction flow.

Bonus:
Remove or comment out the last 2 lines of your query. This will tell the transaction operator to ignore the order in which events happened and simply count the times each state was triggered for a given IP Address without regard for the order in which the states were triggered.
Analyzing your Metrics

Lab 14 - Metric Basics
Metrics track KPIs over time and help you gain visibility into your application's performance. In this lab, we'll plot data from an online travel website to determine successful versus unsuccessful bookings.

1. In a new Metrics tab, add a query to search for all your successful bookings for the last 60 minutes.
   \[\text{type=bookings metric=success.count}\]

2. In a second query underneath the first one, search for all failed bookings.
   \[\text{type=bookings metric=fail.count}\]

3. Click on the Settings tab to see what options are available to you. For example, you can change the chart type (line or area), the color palette used, the line width and the axes labels and scales.

4. Explore the Legend tab, which allows you to view all time series detail for each metric charted.

5. Back in the query tab, toggle off the success.count by clicking on the orange icon (\(\text{\small \ding{41}}\)). This will now only chart the fail.count metric.

6. Any pink dots in your chart are identifying outliers in your data. You can edit settings for your outliers by clicking the pink dot on the top right and editing the Outliers settings: how many to outliers to show (Top) and how many standard deviations to use when considering outliers (Threshold).

7. Lastly, click on the 3 grey dots in the top right corner to view the query info, refresh the query, or add this chart to a Dashboard.
Lab 15 - Compare your KPIs to a Different Time Period

The `timeshift` operator shifts time series of your query. It's very useful to compare across multiple time periods.

1. In a new Metrics tab, add a query to search for all your mean latency for the last 60 minutes.
   ```
   metric=latency.mean
   ```

2. Compare that with your latency from 1 day ago.
   ```
   metric=latency.mean | timeshift 1d
   ```

Lab 16 - Identify Rate of Change

Similar to logs, metrics have the usual operators (min, max, sum, count, avg). However, oftentimes, what you want to measure is change.

1. In a new Metrics tab, add a query to search for a count of packets received in the last 60 minutes.
   ```
   type=packets_received metric=count
   ```

2. To find the difference between one data point and the next, edit your query to show the delta.
   ```
   type=packets_received metric=count | delta
   ```

3. However, to find the rate of change, in this case, packets received per second, edit your query to:
   ```
   type=packets_received metric=count | rate
   ```

   With this last query, you're able to determine if the rate at which packets are being received is increasing gradually or spiking quickly. **Identifying an outlier on a rate of change is a better indicator of an impending problem.**
Lab 17 - Relating your metrics and logs
Metrics allow you to identify symptoms in your environment (WHAT is going on?). Relevant logs help you identify the cause (WHY is this happening?). Looking at our failed bookings from Lab 14, let’s take a look at the relevant logs to identify why we have failed bookings.

1. Follow steps 1 and 2 for Lab 14 to identify counts of successful booking and failed bookings for your travel website.

2. To overlay your metrics with the relevant logs, enter this log query as depicted below:

   ```
   _sourceCategory=training/travel/checkout error
   ```

3. Notice the orange bars at the top. The darker the bar, the larger the number of logs containing the word ERROR. Click the bar to view the relevant logs in this same screen. Shift+click to view logs in a Log Search screen.
Visualizing your Data

Lab 18 - Create and Publish a Dashboard
Dashboards allow you to group your queries and charts and share with others.

4. Run Lab 2.3 to obtain a pie chart.
5. In the chart area, click Add to Dashboard.
6. You can name your new Panel: *Top 10 by Bandwidth*.
7. You can name your Interactive Dashboard: *<Your Name> Training Dashboard*
8. Once created, you are able to resize your Panel, toggle themes, add filters, or share your newly created Dashboard.
9. Note that by clicking the "Show in Search" button, you can go back to the query behind the map chart, which allows you to make any changes needed to the query and re-save by clicking "Save Back to Panel".

Lab 19 - Add Panels to your Dashboard
1. Re-run lab 11 and create a line graph.
2. Save this as a Panel to your existing dashboard.
3. Add a Text panel to your dashboard to simulate the screenshot below.
4. Lastly, share your dashboard with other users in your organization.

**Bonus**: Re-run Lab 15 and add the resulting Metrics chart to this Dashboard. A Dashboard can contain both log and metrics data Panels.

Last Updated: October 4 2018
Lab 20 - Create Meaningful Alerts

In this lab, rather than alerting on simple counts, learn to create an alert that will notify you when your 404s increase at a higher rate than your overall traffic (measured by 200s). See this blog post for full explanation.

1. Search only for messages with status code 200 or 404
2. Count 2** messages as Successes and 4** messages as Fails
3. Sum Successes and Fails to get a count
4. Create a ratio of fails to successes
5. Use outlier operator to identify anomalies in the ratio

```
| _sourceCategory=Labs/Apache/Access (status_code=200 or status_code=404)
| timeslice 1m
| if (status_code matches "2\*", 1, 0) as successes
| if (status_code matches "4\*", 1, 0) as fails
| sum(successes) as success_cnt, sum(fails) as fail_cnt by _timeslice
| fail_cnt/success_cnt as failure_rate
| sort _timeslice desc
| outlier failure_rate window=5, threshold=3, consecutive=1, direction=+
```

6. Adding this line allows you to filter out only outliers (where ration increase is higher than normal) . You can now create a Scheduled Search to Alert when this query has results.

   | where failure_rate_indicator > 0

Lab 21 - Alerting when Data Ingest Reaches 50% Threshold

Create an alert that will notify you when your data usage has reached a certain threshold.

1. Note that as a prerequisite, this solution requires enabling the Data Volume Index to capture more detail information about your daily data ingest.
2. Note that the query below will only return results if it meets the threshold of 0.5. You can comment out the where clause to test query for all your Collectors.
3. Ideally, you can schedule this query to run every 4 hours, with a time range of Today.

```
| _index=sumologic_volume
| where _sourceCategory="collector_volume"
| parse regex
   "(?<collector>"[^\"]*"):\{"sizeInBytes":(?<bytes>\d+),"count":(?<count>\d+)}" multi
| bytes/1024/1024/1024 as gbytes
| sum(gbytes) as gbytes by collector
| total gbytes as todays_volume
| "100" as plan_size //replace with your daily plan limit
| gbytes / todays_volume as collector_pct_of_todaysvolume
| todays_volume / plan_size as todaysvolume_against_plan
| where todaysvolume_against_plan > .5 //replace with desired threshold
| sort gbytes
| fields collector, gbytes, collector_pct_of_todaysvolume, todays_volume, plan_size, todaysvolume_against_plan
```
Lab 22 - Installing the Apache App

Sumo Logic Apps allow you to take advantage of out-of-the-box-content, providing you with popular queries and Dashboards for common data Sources.

1. In the App Catalog, search for Apache
2. Select Apache and click Add to Library
3. For _sourceCategory, select Labs/Apache/Access.
4. You will notice your Personal folder has a new folder named Apache. This new folder includes all pre-built queries and Dashboards.
5. Review the new content, both queries and dashboards.
6. Notice that you can click on any Dashboard panel to view the query behind it. If you make changes to the query, you can always click Update Dashboard to save your changes to the original dashboard.