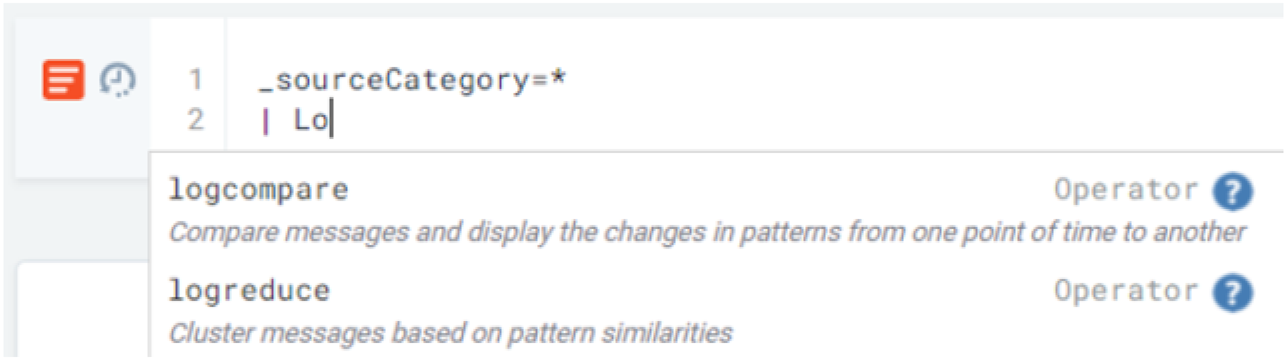
What are LogReduce and LogCompare?

dev.classmethod.jp/articles/sumologic-logreduce-logcompare-202303

佐久間昇吾

March 16, 2023



```
_sourceCategory=*
| Lo
```

logcompare — Operator
Compare messages and display the changes in patterns from one point of time to another

logreduce — Operator
Cluster messages based on pattern similarities

conclusion

Sumo Logic enables log clustering using search functions that use patented technologies LogReduce and LogCompare.
In this article, we will explain the differences between the two and efficient search methods, using examples based on anticipated use cases.

First, let's explain what LogReduce and LogCompare are and what effects they have.

LogReduce

This is an AI-based feature that groups logs by similarity based on the string patterns output in log query results,
allowing you to investigate logs that are grouped into small groups.

LogReduce | Sumo Logic

LogCompare

This function uses LogReduce to compare data with past data. It
compares and groups each grouped log data with the current log data. LogCompare also allows you to see the release of new logs and trends in increases and decreases.

LogCompare | Sumo Logic

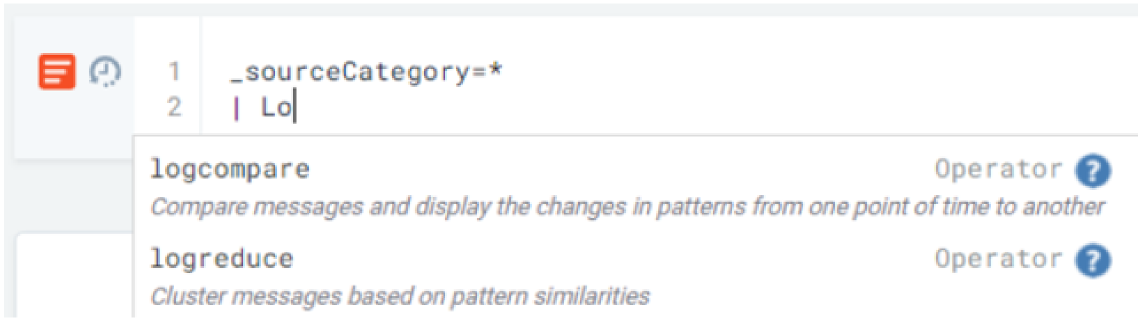
Common uses of LogReduce and LogCompare

- Both can be used before or after a query in LogSearch.

- When searching

  ```
  _sourceCategory=* // 任意のメタデータ、カテゴリデータを指定可能
  | LogCompare
  ```

  If you enter part of the text as shown below, suggestions will appear.

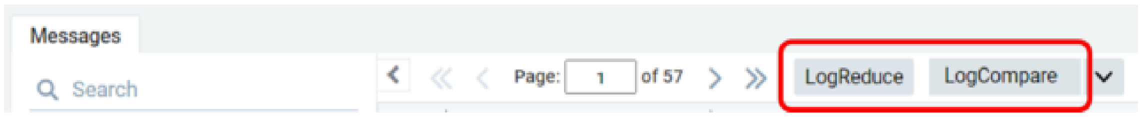
  

  ```
  _sourceCategory=*
  | Lo
  ```

  logcompare Operator
  Compare messages and display the changes in patterns from one point of time to another

  logreduce Operator
  Cluster messages based on pattern similarities

- After searching
  Simply select one of the buttons below.

Now that we have a basic overview, we will move on to the specific differences and use cases of each.

LogReduce

Let's take a closer look at the grouping mentioned at the beginning. Using LogReduce, similar logs are grouped together under a field called Signature. Sumo Logic then displays the number of logs in similar groups under **Count** , allowing users to view detailed log messages for each **Signature** . Sometimes, no logs are displayed under Signature, and "Others" is displayed. This means that the logs cannot be grouped because they contain simple messages of low importance.

Now, let's break down the LogReduce syntax, take some precautions into account, and explain some use cases.

syntax

```
| logreduce [field] [by] [limit] [, criteria]
```

By examining the syntax, we can see that we can search using four elements: [field] [by] [limit] [, criteria]. The elements are explained below.

- **[field]**

    - Specifies the field to group by, if no field is specified the raw message will be used.
      Specifically, you can specify the fields of the logs parsed with purse or json operators. However, there is not much benefit to using | logreduce [field] alone. If you do specify [field], try using it in combination with by, as described below.

- **[by]**
    - Use this when you want to group by some other element for the groups created by LogReduce.
      Specifically, Field, _timeslice, etc. However, in this case, it only aggregates the data like the count operator, and you cannot check the Signature.

      ```
      // Field
      _sourceCategory=Labs/Apache/*
      | parse "GET * " as Get_Object
      | logreduce by Get_Object

      // _timeslice
      _sourceCategory=Labs/Apache/*
      | parse "GET * " as Get_Object
      | timeslice 1m
      | logreduce(Get_Object) by _timeslice
      | sort by _timeslice asc
      ```

- **[limit]**
    - Limits the number of signatures returned.
      LogReduce basically processes a large number of logs, which means the number of signatures will also be large, but if that makes it difficult to investigate, it limits the number of signatures.

      ```
      _sourceCategory= "Labs/AWS/GuardDuty_V8"
      | logreduce by _sourceHost limit=5
      ```

- **[, criteria]**
    - By default, LogReduce will find the most unusual signatures. Override this with criteria.

      ```
      criteria=mostcommon // 最も頻繁に出現するカウント数の多い Signature（デフォルト）
      criteria=leastcommon // 出現頻度とカウント数が低い Signature

      _sourceCategory= "Labs/AWS/GuardDuty_V8"
      | logreduce by _sourceHost limit=5,criteria=mostcommon
      ```

Use Cases

As mentioned at the beginning, this feature can be used to narrow the scope of an investigation by consolidating large volumes of logs. We'll explain specifically when, how, and to what extent this should be narrowed down. Simply saying that this is a feature that can group large volumes of logs together and performing an extremely abstract search such as _sourceCategory=Labs/AWS/GuardDuty_V3 | logreduce will only result in a huge number of signatures. Therefore, narrowing the scope at the query stage using parse and operators such as where and if will improve the efficiency of your investigation.

An example is shown below. Depending on the volume of logs being output, increase the conditions to narrow the scope when querying.

- **Example 1) Narrow down by AWS region and account level (environment level)**

  ```
  _sourceCategory=Labs/AWS/GuardDuty_V3
  | parse "accountId\":\"*\"" as accountId, "region\":\"*\"" as region
  | where !isNull(accountId AND region)
  | where accountId = "prod" AND region = "us-east-1"
  | logreduce
  ```

  Here, the second line parses the accountId and region, and the third line checks for null in the fields to reject them. Furthermore, the fourth line specifies that it is a production environment and that the region is Northern Virginia. When looking at logs that you want to investigate by environment or region, especially when multi-region or micro-component environments, multiple signatures will be output in large quantities, so narrowing down the query will improve the efficiency of your investigation.

- **Example 2) Using keyword search**

  ```
  _sourceCategory=Labs/Apache/* error
  | parse "mod_log_sql: * connection error" as db_connection_error
  | where !isNull(db_connection_error)
  | where db_connection_error = "database"
  | logreduce
  ```

  In this example, the first line narrows down the logs to the keyword "error," then the second and third lines parse and check for nulls, and the fourth line narrows down the logs to those containing the string "database." This allows you to search for logs in which a connection to the database could not be established successfully, even among error logs.

LogCompare

This allows comparison with past data, which can help discover logs that did not occur before or logs that have increased abnormally compared to the past. Specifically, TimeShift, which will be described later, compares the baseline (past log data group) with the target (current log data group) and outputs a signature including the occurrence rate of the log group. This makes it possible to discover new logs that are output in logs before the failure occurred, enabling troubleshooting of the root cause.

syntax

```
| logcompare timeshift <time> [baseline (<baseline query>)]
```

By reading the syntax, we can see that we can search using two elements: timeshift [baseline ()].

- **timeshift**
  - Determine the baseline (the starting point of the historical log data set).

    ```
    _sourceCategory=Labs/Apache/* error
    | logcompare timeshift -24h
    ```

- **[baseline()]**
  - You can compare the target with the baseline query.
    Specifically, | logcompare timeshift -N compares past and current logs of previous query results and the query results written in baseline().

    ```
    _sourceHost=cluster-1
    | logcompare timeshift -0s baseline(_sourceHost=cluster-2)
    ```

Use Cases

Because this is an expanded version of LogReduce, it still uses operators such as parse, where, and if. Basically, you can use the methods explained in the LogReduce use case, but combine them with LogCompare, which we will introduce here as a feature of comparing with past data.

- Example 1) Display only newly appeared logs.

  ```
  _sourceCategory=Labs/AWS/GuardDuty_V3
  | parse "accountId\":\"*\"" as accountId, "region\":\"*\"" as region
  | where !isNull(accountId AND region)
  | where accountId = "prod" AND region = "us-east-1"
  | logcompare timeshift -24h
  | where (_isNew)
  ```

  The third line, where (_isNew), specifies a condition to display only newly appeared logs in the output results of LogCompare.
- Example 2) Displaying important information such as error logs

  ```
  _sourceCategory=Labs/Apache/* error
  | parse "GET * " as GET_Object
  | where !isNull(GET_Object)
  | logcompare timeshift -15m
  ```

  An increasing number of logs, such as errors and high-severity log messages, can also be investigated using this keyword search.

summary

LogReduce and LogCompare are AI-based search functions, but their search results don't always provide a clear answer. Therefore, it's important to clarify the query and identify the logs you want to view. In some cases, aggregating data using the count or transpose operators can be easier to understand. For example, while writing this article, I considered using LogCompare to output the web page cache rate using HTTP status codes. A more accurate answer would be to combine the count, transpose, and _timeslice operators. LogReduce is a useful tool for obtaining limited information, allowing you to investigate important information within many logs or for periodic analysis. LogCompare can be used to compare logs with past data to troubleshoot the root cause of alerts. This is a feature I'd like to explore in more depth, so I plan to continue updating this article.