Fusion-Reactor.com

LogQL Cheat Sheet

This cheat sheet highlight different pattern for querying FusionReactor LogQL.

Introducing LogQL log query Language

FusionReactor queries log with its unique log query language called LogQL, often distributed as grep besides labels for filtering. There are two parts of a basic LogQL: the filter expression and the log stream selector.

All LogQL queries contain a log streaming selector. The log stream selector does most of the work. It filters the logs and reduces the log volume. This stream collector impacts the performance of the query's execution, depending on the number of labels used to filter down the log streams.

Log Stream Selector

The primary function of the log stream selector is to select and organize which log selector will query streams. At least one or more key-value pairs are found in the stream selector, which makes up the log label's value. This key-value pairs are closely enclosed in a pair of curly braces:

{app="mysql",name="mysql-backup"}

In the example above, we can see that the log streams contain a label of the app whose value is Mysql and a name whose value is in the query results. The = represents the label matching operator. This does not limit the following label matching operators when querying logs:

- =~: regex matches.
- !~: regex does not match.
- =: exactly equal.
- !=: not equal.

Another example:

{name=~"mysql.+"}{name!~"mysql.+"}

FusionReactor and Prometheus apply the same rules when selecting labels.

Filter expressions

After filtering the logs through the log stream selector in FusionReactor, the resulting set of logs can be further filtered with a search expression
The search expression can be just text or regex:

- {job="mysql"} |= "error"
- {name="kafka"} |~ "tsdb-ops.*io:2003"
- {instance=~"kafka-[23]",name="kafka"} != kafka.server:type=ReplicaManager

The following filter operators are supported when querying logs:

- |=: Log line contains a string.
- !=: Log line does not contain a string.
- |~: Log line matches the regular expression.
- !~: Log line does not match the regular expression.

Note. that all log lines should match every filter expression, regardless of how to filter operators are channeled for search expression:

{job="mysql"} |= "error" != "timeout"

When using |~ and !~, Go RE2 syntax regex may be used. By default, the matching is case-sensitive and can be switched to case-insensitive, prefixing the regex with (?i).

Counting Logs

Another exciting feature of LogQL is that it can envelop a query and allow the log stream selector to count entries per stream.

Range Vector Aggregation

LogQL range vector is unique from other log language queries because the selected range of samples usually includes a value of 1 for each entry. Hence, the aggregate can envelop a selected range into an instance vector. The range vector supported functions are:

- rate: calculate the number of entries per second
- count_over_time: counts the entries for each log stream within the given range.
- count_over_time({job="MySQL"}[5m])

For instance, the range vector counts all the log lines among the first 5 minutes for the MySQL query. This example showcases the per-second rate of all non timeout errors
rate:

(({job="mysql"} |= "error" != "timeout)[10s]))

Aggregation operators

LogQL aggregation operators act similarly to PromQL, which supports a subset that can aggregate the elements of a single vector. This action results in a new vector of fewer elements with aggregated values.

- sum: Calculate the sum over labels
- min: Select minimum over labels
- max: Select maximum over labels
- avg: Calculate the average over labels
- stddev: Calculate the population standard deviation over labels
- stdvar: Calculate the population standard variance over labels
- count: Count the number of elements in the vector
- bottomk: Select the smallest k elements by sample value
- topk: Select largest k elements by sample value

Also, by using a *without* or *by* clause, whether a without or by clause, the log operators can use the aggregation operators to aggregate over all label values or a set of distinct values.
For instance

<aggr-op>([parameter,] <vector expression>) [without|by (<label list>)]

Note. Using *parameters* is only needed when using topk, bottomk., and topk, considered group vectors. Aggregators such as bottomk are different from other aggregators within the sample subset.
The without cause removes listed labels from results, while the by clause does the opposite, unlisting labels that are not part of the clause, regardless of label identity.
For instance: filter logs for the top 10 applications by highest log throughput;

topk(10,sum(rate({region="us-east1"}[5m])) by (name))

Get the count of logs during the last five minutes, grouping by level:

sum(count_over_time({job="mysql"}[5m])) by (level)

Get the rate of HTTP GET requests from NGINX logs:

avg(rate(({job="nginx"} |= "GET")[10s])) by (region)