Query

- Simple LogQL simulator
- Query best practices
- Matching IP addresses
- Metric queries
- Query acceleration
- Query examples
- LogQL Reference
- LogQL template functions
- Troubleshoot log queries (READ)
- LogCLI getting started
- LogCLI tutorial

Simple LogQL simulator

Simple LogQL simulator

The LogQL simulator is an online tool that you can use to experiment with writing simple LogQL queries and seeing the results, without needing to run an instance of Loki.

A set of example log lines are included for each of the primary log parsers supported by Loki:

- Logfmt
- JSON
- Unstructured text, which can be parsed with the Loki pattern or regex parsers

The log stream selector `{job="analyze"}` is shown as an example, and it remains fixed for all possible example queries in the simulator. A log stream is a set of logs which share the same labels. In LogQL, you use a log stream selector to determine which log streams to include in a query's results.

{{< admonition type="note" >}} This is a very limited simulator, primarily for evaluating filters and parsers. If you want to practice writing more complex queries, such as metric queries, you can use the Explore feature in Grafana. {{< /admonition >}}

To use the LogQL simulator:

1. Select a log line format using the radio buttons.
2. You can use the provided example log lines, or copy and paste your own log lines into the example log lines box.
3. Use the provided example LogQL query, or enter your own query. The log stream selector remains fixed for all possible example queries. There are additional sample queries at the end of this topic.
4. Click the **Run query** button to run the entered query against the example log lines.

The results output simulates how Loki would return results for your query. You can also click each line in the results pane to expand the details, which give an explanation for why the log line is or is not included in the query result set.

Log line format: logfmt JSON Unstructured text
Link copied to clipboard.

{job="analyze"}
Query:
{job="analyze"}

Additional sample queries

These are some additional sample queries that you can use in the LogQL simulator.

Logfmt

```
| logfmt | level = "debug"
```

Parses logfmt-formatted logs and returns only log lines where the "level" field is equal to "debug".

```
| logfmt | msg="server listening on addresses"
```

Parses logfmt-formatted logs and returns only log lines with the message “server listening on address.”

JSON

```
| json | level="INFO" | file="SpringApplication.java" | line_format `{{.class}}`
```

Parses JSON-formatted logs, filtering for lines where the 'level' field is "INFO" and the 'file field is "SpringApplication.java", then formats the line to return only the 'class' field.

```
|~ `(T|t)omcat`
```

Performs a regular expression filter for the string 'tomcat' or 'Tomcat', without using a parser.

Unstructured text

```
| pattern "<_> - <_> <_> \"<method> <url> <protocol>\" <status> <_> <_> \"<_>\" <_>" |
method="GET"
```

Parses unstructured logs with the pattern parser, filtering for lines where the HTTP method is "GET".

```
| pattern "<_> - <user> <_> \"<method> <url> <protocol>\" <status> <_> <_> \"<_>\" <_>" |
user=~"kling.*"
```

Parses unstructured logs with the pattern parser, extracting the 'user' field, and filtering for lines where the user field starts with "kling".

Query best practices

The way you write queries in Loki affects how quickly you get results returned from those queries. Understanding the way Loki parses queries can help you write queries that are efficient and performant.

{{< admonition type="tip" >}} Before you start optimizing queries, read the labels best practices page to understand what makes a good label. Choosing the right labels is the first step towards writing efficient queries. {{< /admonition >}}

Loki evaluates a LogQL query from left to right, in the order that it is written. To get the best possible query performance, eliminate as many potential results as you can earlier in the query and then continue to progressively narrow your search as you continue writing the query. This page describes the recommended order for writing queries that efficiently filter out unwanted results.

Narrow down your time range first

Reduce the number of logs Loki needs to look through by specifying a period of time that you'd like to search through. Loki creates one index file per day, so queries that span over multiple days fetches multiple index files. The fewer files Loki has to search, the faster the query results are returned.

Time ranges are typically not part of the query, but you can set a time range through your visualization tool or through the Loki API.

In Grafana

If you're using Loki with Grafana, you can use the dropdown menu on the upper right hand corner of a dashboard to select a time range, either relative (last X hours) or absolute (a specific date and time).

Screenshot of time selector on Grafana

Through Loki API

If you're querying Loki through the Loki API, you can use the `query_range` endpoint to add `start` and `end` timestamps for your query as parameters to the HTTP call rather than as part of the query itself.

```
http://<loki-
instance>/loki/api/v1/query_range?query={job="app"}&start=1633017600000000000&end=1633104000000000000
```

Use precise label selectors

Next, write your label selectors. Identify the most specific label you can use within the log line and search based on that first. For example, if the logs contain the labels `namespace` and `app_name` and the latter is a smaller subset of data, start your query by selecting based on `app_name`:

```
{app_name="carnivorousgreenhouse"}
```

Using the most specific label selector has the added benefit of reducing the length of your query. Since `app_name` is more specific than `namespace`, you don't need to add a selector for `namespace`. Adding more general label selectors has no further effect on the query.

Use simple line filter expressions over regular expressions

When using line filter expressions, prefer the simpler filter operators such as:

- `|=` (contains string) and
- `!=` (does not contain string) over the regular expression filter operators:
- `|~` (matches the regular expression)
- `!~` (does not match the regular expression)

Loki evaluates the first two filter expressions faster than it can evaluate regular expressions, so always try to rewrite your query in terms of whether a log line contains or does not contain a certain string. Use regular expressions only as a last resort.

Line filter expressions are more efficient than parser expressions.

Avoid using complex text parsers

Use parser expressions only after line filter expressions. Parser expressions are ways to look through the log line and extract labels in different formats, which can be useful but are also more intensive for Loki to do than line filter expressions. Using them after line filter expressions means that Loki only needs to evaluate parser expressions for log lines that match the line filter expression, reducing the amount of logs that Loki needs to search through.

Parser expressions include JSON, logfmt, pattern, regexp, and unpack parsers.

Use recording rules

Some queries are sufficiently complex, or some datasets sufficiently large, that there is a limit as to how much query performance can be optimized. If you're following the tips on this page and are still experiencing slow query times, consider creating a recording rule for them. A recording rule runs a query at a predetermined time and also precomputes the results of that query, saving those results for faster retrieval later.

Further resources

- Watch: 5 tips for improving Grafana Loki query performance
- Grafana Loki Design Basics with Ed Welch (Grafana Office Hours #27)
- Labels best practices

Matching IP addresses

LogQL supports matching IP addresses.

With logs such as

```
3.180.71.3 - - [17/May/2015:08:05:32 +0000] "GET /downloads/product_1 HTTP/1.1" 304 0 "-"
"Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.21)"
80.91.33.133 - - [17/May/2015:08:05:14 +0000] "GET /downloads/product_1 HTTP/1.1" 304 0 "-"
"Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.16)"
46.4.66.76 - - [17/May/2015:08:05:45 +0000] "GET /downloads/product_1 HTTP/1.1" 404 318 "-"
"Debian APT-HTTP/1.3 (1.0.1ubuntu2)"
93.180.71.3 - - [17/May/2015:08:05:26 +0000] "GET /downloads/product_1 HTTP/1.1" 404 324 "-"
"Debian APT-HTTP/1.3 (0.8.16~exp12ubuntu10.21)"
```

the LogQL line filter is not sufficient. A line filter such as

```
{job_name="myapp"} |= "3.180.71.3"
```

also matches example IP addresses such as 93.180.71.3. A better choice uses a regexp: `|~"^3.180.71.3"`. This regexp does not handle IPv6 addresses, and it does not match a range of IP addresses.

The LogQL support for matching IP addresses handles both IPv4 and IPv6 single addresses, as well as ranges within IP addresses and CIDR patterns.

Match IP addresses with the syntax: `ip("<pattern>")`. The `<pattern>` can be:

- A single IP address. Examples: `ip("192.0.2.0")`, `ip("::1")`
- A range within the IP address. Examples: `ip("192.168.0.1-192.189.10.12")`, `ip("2001:db8::1-2001:db8::8")`
- A CIDR specification. Examples: `ip("192.51.100.0/24")`, `ip("2001:db8::/32")`

The IP matching can be used in both line filter and label filter expressions. When specifying line filter expressions, only the `|=` and `!=` operations are allowed. When specifying label filter expressions, only the `=` and `!=` operations are allowed.

- Line filter examples

  ```
  {job_name="myapp"} |= ip("192.168.4.5/16")
  ```

Return log lines that do not match with an IPv4 range:

```
{job_name="myapp"} != ip("192.168.4.5-192.168.4.20")
```

- Label filter examples

```
{job_name="myapp"}
  | logfmt
  | remote_addr = ip("2001:db8::1-2001:db8::8")
  | level = "error"
```

Filters can also be chained. This example matches log lines with all IPv4 subnet values `192.168.4.5/16` except IP address `192.168.4.2`:

```
{job_name="myapp"}
  | logfmt
  | addr = ip("192.168.4.5/16")
  | addr != ip("192.168.4.2")
```

This example use the conditional `or` and matches log lines with either, all IPv4 subnet values `192.168.4.0/24` OR all IPv4 subnet values `10.10.15.0/24`:

```
{job_name="myapp"}
  | logfmt
  | addr = ip("192.168.4.0/24") or addr = ip("10.10.15.0/24")
```

Metric queries

{{< shared id="metric-queries" >}}

Metric queries extend log queries by applying a function to log query results. This powerful feature creates metrics from logs.

Metric queries can be used to calculate the rate of error messages or the top N log sources with the greatest quantity of logs over the last 3 hours.

Combined with parsers, metric queries can also be used to calculate metrics from a sample value within the log line, such as latency or request size. All labels, including extracted ones, will be available for aggregations and generation of new series.

{{< /shared >}}

Range Vector aggregation

LogQL shares the range vector concept of Prometheus. In Grafana Loki, the selected range of samples is a range of selected log or label values.

The aggregation is applied over a time duration. Loki defines Time Durations with the same syntax as Prometheus.

Loki supports two types of range vector aggregations: log range aggregations and unwrapped range aggregations.

Log range aggregations

A log range aggregation is a query followed by a duration. A function is applied to aggregate the query over the duration. The duration can be placed after the log stream selector or at end of the log pipeline.

The functions:

- `rate(log-range)`: calculates the number of entries per second
- `count_over_time(log-range)`: counts the entries for each log stream within the given range.
- `bytes_rate(log-range)`: calculates the number of bytes per second for each stream.

- `bytes_over_time(log-range)`: counts the amount of bytes used by each log stream for a given range.
- `absent_over_time(log-range)`: returns an empty vector if the range vector passed to it has any elements and a 1-element vector with the value 1 if the range vector passed to it has no elements. (`absent_over_time` is useful for alerting on when no time series and logs stream exist for label combination for a certain amount of time.)

Examples:

- Count all the log lines within the last five minutes for the MySQL job.

  ```
  count_over_time({job="mysql"}[5m])
  ```

- This aggregation includes filters and parsers. It returns the per-second rate of all non-timeout errors within the last minutes per host for the MySQL job and only includes errors whose duration is above ten seconds.

  ```
  sum by (host) (rate({job="mysql"} |= "error" != "timeout" | json | duration > 10s [1m]))
  ```

Offset modifier

The offset modifier allows changing the time offset for individual range vectors in a query.

For example, the following expression counts all the logs within the last ten minutes to five minutes rather than last five minutes for the MySQL job. Note that the `offset` modifier always needs to follow the range vector selector immediately.

```
count_over_time({job="mysql"}[5m] offset 5m) // GOOD
count_over_time({job="mysql"}[5m]) offset 5m // INVALID
```

Unwrapped range aggregations

Unwrapped ranges uses extracted labels as sample values instead of log lines. However to select which label will be used within the aggregation, the log query must end with an unwrap expression and optionally a label filter expression to discard errors.

The unwrap expression is noted `| unwrap label_identifier` where the label identifier is the label name to use for extracting sample values.

Since label values are string, by default a conversion into a float (64bits) will be attempted, in case of failure the `__error__` label is added to the sample. Optionally the label identifier can be wrapped by a conversion function `| unwrap <function>(label_identifier)`, which will attempt to convert the label value from a specific format.

We currently support the functions:

- `duration_seconds(label_identifier)` (or its short equivalent `duration`) which will convert the label value in seconds from the go duration format (e.g `5m`, `24s30ms`).
- `bytes(label_identifier)` which will convert the label value to raw bytes applying the bytes unit (e.g. `5 MiB`, `3k`, `1G`).

Supported function for operating over unwrapped ranges are:

- `rate(unwrapped-range)`: calculates per second rate of the sum of all values in the specified interval.
- `rate_counter(unwrapped-range)`: calculates per second rate of the values in the specified interval and treating them as "counter metric"
- `sum_over_time(unwrapped-range)`: the sum of all values in the specified interval.
- `avg_over_time(unwrapped-range)`: the average value of all points in the specified interval.
- `max_over_time(unwrapped-range)`: the maximum value of all points in the specified interval.
- `min_over_time(unwrapped-range)`: the minimum value of all points in the specified interval
- `first_over_time(unwrapped-range)`: the first value of all points in the specified interval
- `last_over_time(unwrapped-range)`: the last value of all points in the specified interval
- `stdvar_over_time(unwrapped-range)`: the population standard variance of the values in the specified interval.
- `stddev_over_time(unwrapped-range)`: the population standard deviation of the values in the specified interval.
- `quantile_over_time(scalar,unwrapped-range)`: the φ-quantile ($0 \leq \varphi \leq 1$) of the values in the specified interval.
- `absent_over_time(unwrapped-range)`: returns an empty vector if the range vector passed to it has any elements and a 1-element vector with the value 1 if the range vector passed to it has no elements. (`absent_over_time` is useful for alerting on when no time series and logs stream exist for label combination for a certain amount of time.)

Except for `sum_over_time`,`absent_over_time`, `rate` and `rate_counter`, unwrapped range aggregations support grouping. When a pipeline stage fails on a line, the resulting sample keeps the same labels a successful line would report for that grouping, plus the `__error__` and `__error_details__` labels, rather than the full label set. For the full set of rules, see Pipeline Errors in Metric Queries.

```
<aggr-op>([parameter,] <unwrapped-range>) [without|by (<label list>)]
```

Which can be used to aggregate over distinct labels dimensions by including a `without` or `by` clause.

`without` removes the listed labels from the result vector, while all other labels are preserved the output. `by` does the opposite and drops labels that are not listed in the `by` clause, even if their label values are identical between all elements of the vector.

See Unwrap examples for query examples that use the unwrap expression.

Built-in aggregation operators

Like PromQL, LogQL supports a subset of built-in aggregation operators that can be used to aggregate the element of a single vector, resulting in a new vector of fewer elements but with aggregated values:

- `sum`: Calculate sum over labels
- `avg`: Calculate the average over labels
- `min`: Select minimum over labels
- `max`: Select maximum over labels
- `stddev`: Calculate the population standard deviation over labels
- `stdvar`: Calculate the population standard variance over labels
- `count`: Count number of elements in the vector
- `topk`: Select largest k elements by sample value
- `bottomk`: Select smallest k elements by sample value
- `sort`: returns vector elements sorted by their sample values, in ascending order.
- `sort_desc`: Same as sort, but sorts in descending order.

The aggregation operators can either be used to aggregate over all label values or a set of distinct label values by including a `without` or a `by` clause:

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

`parameter` is required when using `topk` and `bottomk`. `topk` and `bottomk` are different from other aggregators in that a subset of the input samples, including the original labels, are returned in the result vector.

`by` and `without` are only used to group the input vector. The `without` clause removes the listed labels from the resulting vector, keeping all others. The `by` clause does the opposite, dropping labels that are not listed in the clause, even if their label values are identical between all elements of the vector.

See vector aggregation examples for query examples that use vector aggregation expressions.

Functions

LogQL supports a set of built-in functions.

- `vector(s scalar)`: returns the scalar s as a vector with no labels. This behaves identically to the Prometheus `vector()` function. `vector` is mainly used to return a value for a series that would otherwise return nothing; this can be useful when using LogQL to define an alert.

Examples:

- Count all the log lines within the last five minutes for the traefik namespace.

```
sum(count_over_time({namespace="traefik"}[5m])) # will return nothing
  or
vector(0) # will return 0
```

Probabilistic aggregation

`approx_count_distinct`

LogQL's `approx_count_distinct` function approximates the number of distinct values of a label or extracted field without creating one series per distinct value. Use it when an exact `count by` would explode series cardinality.

To use `approx_count_distinct`, set `limits_config.shard_aggregations` to include `approx_count_distinct` in your Loki configuration. This operator also requires `frontend.encoding: protobuf`.

The function is of the form:

```
approx_count_distinct(
  <counted field>,
  <log expression> [<duration>]
) [by (<grouping fields>)]
```

Example:

```
approx_count_distinct(
  mac_address,
  {job="devices"} | json [1d]
) by (version)
```

`approx_count_distinct` supports instant and range queries. A range duration is required. Grouping is optional: omit `by` to keep leftover stream labels, or use `by ()` for one unlabeled series. Don't group by the counted field.

Under the hood, Loki builds one HyperLogLog sketch per output group at precision 14 (sparse at low cardinality, about 16 KiB when dense). Index sharding merges sketches before estimating so overlapping values across shards are counted once.

`approx_topk`

{{< admonition type="note" >}} `approx_topk` is an experimental feature. Engineering and on-call support is not available. Documentation is either limited or not provided outside of code comments. No SLA is provided. To use this feature, set `limits_config.shard_aggregations` to include `approx_topk` in your Loki configuration. To enable this feature in Grafana Cloud, contact Grafana Support. {{< /admonition >}}

LogQL's `approx_topk` function provides a probabilistic approximation of `topk`. It is a drop-in replacement for `topk` that is great for when `topk` queries time out or hit the maximum series limit. This tends to happen when the list of values that you're sorting through in order to find the most frequent values is very large. `approx_topk` is also great in cases where a faster, approximate answer is preferred to a slower, more accurate one.

The function is of the form:

```
approx_topk(k, <vector expression>)
```

`approx_topk` is only supported for instant queries. Grouping is also not supported and should be handled by an inner `sum by` or `sum without` even though this might not be the same behavior as `topk by`.

Under the hood, `approx_topk` is implemented using sharding. The count-min sketch algorithm and a heap are used to approximate the counts for each shard. The accuracy of the approximation depends on the size of the heap, which is defined by Loki's `max_count_min_sketch_heap_size` parameter. Accuracy decreases as `k` approaches the size of the heap (which has a default size of `10,000`).

The expression `approx_topk(k,inner)` becomes

```
topk(
  k,
  eval_cms(
    __count_min_sketch__(inner, shard=1) ++ __count_min_sketch__(inner, shard=2)...
  )
```

```
)
```

`__count_min_sketch__` is calculated for each shard and merged on the frontend. Then `eval_cms` iterates through the labels list and determines the count for each. Then `topk` selects the top items.

Result ordering

Metric query results are not guaranteed to be returned in any particular order, unless the query uses `sort` or `sort_desc`, which order the elements by their sample value. Don't rely on the ordering of results in any other case.

`sort` and `sort_desc` only affect the results of instant queries. The ordering of range query results is unspecified, even when the query uses `sort` or `sort_desc`.

Further resources

- Watch: How to turn logs into metrics with Grafana Loki (Loki Community Call July 2025)

Query acceleration

{{< admonition type="warning" >}} In Loki and Grafana Enterprise Logs (GEL), Query acceleration using blooms is deprecated and will be removed in a future release. Engineering and on-call support is not available. No SLA is provided. {{< /admonition >}}

If bloom filters are enabled, you can write LogQL queries using structured metadata to benefit from query acceleration.

Prerequisites

- Bloom filters must be enabled. OpenSource and Enterprise customers must enable bloom filters. Grafana Support enables Bloom Filters for Cloud Customers who are part of the public preview.
- Logs must be sending structured metadata. OpenSource and Enterprise customers must enable structured metadata. Structured metadata is enabled by default in Grafana Cloud.

Query blooms

Queries will be accelerated for any label filter expression that satisfies *all* of the following criteria:

- The label filter expression using **string equality**, such as `| key="value"`.
  - `or` and `and` operators can be used to match multiple values, such as `| detected_level="error" or detected_level="warn"`.
  - *Basic* regular expressions are automatically simplified into a supported expression:
    - `| key=~"value"` is converted to `| key="value"`.
    - `| key=~"value1|value2"` is converted to `| key="value1" or key="value2"`.
    - `| key=~".+"` checks for existence of `key`. `.*` is not supported.
- The label filter expression is querying for structured metadata and not a stream label.
- The label filter expression is placed before any parser expression, labels format expression, drop labels expression, or keep labels expression.

To take full advantage of query acceleration with blooms, ensure that filtering structured metadata is done before any parser expression:

In the following example, the query is not accelerated because the structured metadata filter, `detected_level="error"`, is after a parser stage, `json`.

```
{cluster="prod"} | logfmt | json | detected_level="error"
```

In the following example, the query is accelerated because the structured metadata filter is before any parser stage.

```
{cluster="prod"} | detected_level="error" | logfmt | json
```

Query examples

These LogQL query examples have explanations of what the queries accomplish.

Log query examples

Examples that filter on IP address

- Return log lines that are not within a range of IPv4 addresses:

```
{job_name="myapp"} != ip("192.168.4.5-192.168.4.20")
```

- This example matches log lines with all IPv4 subnet values `192.168.4.5/16` except IP address `192.168.4.2`:

```
{job_name="myapp"}
  | logfmt
  | addr = ip("192.168.4.5/16")
  | addr != ip("192.168.4.2")
```

Examples that aid in security evaluation

- Extract the user and IP address of failed logins from Linux `/var/log/secure`

```
{job="security"}
    |~ "Invalid user.*"
    | regexp "(^(?P<user>\\S+ {1,2}){8})"
    | regexp "(^(?P<ip>\\S+ {1,2}){10})"
    | line_format "IP = {{.ip}}\tUSER = {{.user}}"
```

- Get successful logins from Linux `/var/log/secure`

```
{job="security"}
    != "grafana_com"
    |= "session opened"
```

```
    != "sudo: "
    | regexp "(^(?P<user>\\S+ {1,2}){11})"
    | line_format "USER = {{.user}}"
```

Metrics query examples

- Return the per-second rate of all non-timeout errors within the last minutes per host for the MySQL job, and only include errors whose duration is above ten seconds.

```
sum by (host) (rate({job="mysql"}
    |= "error" != "timeout"
    | json
    | duration > 10s [1m]))
```

Multiple filtering stages examples

Query results are gathered by successive evaluation of parts of the query from left to right. To make querying efficient, order the filtering stages left to right:

1. stream selector
2. line filters
3. label filters

Consider the query:

```
{cluster="ops-tools1", namespace="loki-dev", job="loki-dev/query-frontend"} |= "metrics.go" != "out of order" | logfmt | duration > 30s or status_code != "200"
```

Within this query, the stream selector is

```
{cluster="ops-tools1", namespace="loki-dev", job="loki-dev/query-frontend"}
```

There are two line filters: `|= "metrics.go"` and `!="out of order"`. Of the log lines identified with the stream selector, the query results include only those log lines that contain the string "metrics.go" and do not contain the string "out of order".

The `logfmt` parser produces the `duration` and `status_code` labels, such that they can be used by a label filter.

The label filter `| duration > 30s or status_code!="200"` further filters out log lines. It includes those log lines that contain a `status_code` label with any value other than the value 200, as well as log lines that contain a `duration` label with a value greater than 30 seconds,

While every query will have a stream selector, not all queries will have line and label filters.

Examples that use multiple parsers

Consider this logfmt log line. To extract the method and the path of this logfmt log line,

```
level=debug ts=2020-10-02T10:10:42.092268913Z caller=logging.go:66 traceID=a9d4d8a928d8db1
msg="POST /api/prom/api/v1/query_range (200) 1.5s"
```

To extract the method and the path, use multiple parsers (logfmt and regexp):

```
{job="loki-ops/query-frontend"} | logfmt | line_format "{{.msg}}" | regexp "(?P<method>\\w+)
(?P<path>[\\w|/]+) \\((?P<status>\\d+?)\\) (?P<duration>.*)"
```

This is possible because the `| line_format` reformats the log line to become `POST /api/prom/api/v1/query_range (200) 1.5s` which can then be parsed with the `| regexp ...` parser.

Log line formatting examples

The following query shows how you can reformat a log line to make it easier to read on screen.

```
{cluster="ops-tools1", name="querier", namespace="loki-dev"}
  |= "metrics.go" != "loki-canary"
  | logfmt
  | query != ""
  | label_format query="{{ Replace .query \"\\n\" \"\" -1 }}"
  | line_format "{{ .ts}}\t{{.duration}}\ttraceID = {{.traceID}}\t{{ printf \"%-100.100s\"
.query }} "
```

Label formatting is used to sanitize the query while the line format reduce the amount of information and creates a tabular output.

For these given log lines:

```
level=info ts=2020-10-23T20:32:18.094668233Z caller=metrics.go:81 org_id=29
traceID=1980d41501b57b68 latency=fast query="{cluster=\"ops-tools1\", job=\"loki-ops/query-
frontend\"} |= \"query_range\"" query_type=filter range_type=range length=15m0s step=7s
duration=650.22401ms status=200 throughput_mb=1.529717 total_bytes_mb=0.994659
level=info ts=2020-10-23T20:32:18.068866235Z caller=metrics.go:81 org_id=29
traceID=1980d41501b57b68 latency=fast query="{cluster=\"ops-tools1\", job=\"loki-ops/query-
frontend\"} |= \"query_range\"" query_type=filter range_type=range length=15m0s step=7s
duration=624.008132ms status=200 throughput_mb=0.693449 total_bytes_mb=0.432718
```

The result would be:

```
2020-10-23T20:32:18.094668233Z 650.22401ms    traceID = 1980d41501b57b68 {cluster="ops-tools1"
job="loki-ops/query-frontend"} |= "query_range"
2020-10-23T20:32:18.068866235Z 624.008132ms traceID = 1980d41501b57b68 {cluster="ops-tools1",
job="loki-ops/query-frontend"} |= "query_range"
```

It's possible to strip ANSI sequences from the log line, making it easier to parse it further:

```
{job="example"} | decolorize
```

This way this log line:

```
[2022-11-04 22:17:57.811] \033[0;32http\033[0m: GET /_health (0 ms) 204
```

turns into:

```
[2022-11-04 22:17:57.811] http: GET /_health (0 ms) 204
```

Unwrap examples

- Calculate the p99 of the nginx-ingress latency by path:

  ```
  quantile_over_time(0.99,
    {cluster="ops-tools1",container="ingress-nginx"}
      | json
      | __error__ = ""
      | unwrap request_time [1m]) by (path)
  ```

- Calculate the quantity of bytes processed per organization ID:

```
sum by (org_id) (
  sum_over_time(
  {cluster="ops-tools1",container="loki-dev"}
      |= "metrics.go"
      | logfmt
      | unwrap bytes_processed [1m])
  )
```

Vector aggregation examples

Get the top 10 applications by the highest log throughput:

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

Get the count of log lines for the last five minutes for a specified job, grouping by level:

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

Get the rate of HTTP GET requests to the `/home` endpoint for NGINX logs by region:

```
avg(rate(({job="nginx"} |= "GET" | json | path="/home")[10s])) by (region)
```

LogQL Reference

Loki accepts a number of operators and functions within a LogQL query.

Binary operators

Arithmetic operators

The following binary arithmetic operators exist in Loki:

- `+` (addition)
- `-` (subtraction)
- `*` (multiplication)
- `/` (division)
- `%` (modulo)
- `^` (power/exponentiation)

Binary arithmetic operators are defined between two literals (scalars), a literal and a vector, and two vectors.

Between two literals, the behavior is obvious: They evaluate to another literal that is the result of the operator applied to both scalar operands (`1 + 1 = 2`).

Between a vector and a literal, the operator is applied to the value of every data sample in the vector, e.g. if a time series vector is multiplied by 2, the result is another vector in which every sample value of the original vector is multiplied by 2.

Between two vectors, a binary arithmetic operator is applied to each entry in the left-hand side vector and its matching element in the right-hand vector. The result is propagated into the result vector with the grouping labels becoming the output label set. Entries for which no matching entry in the right-hand vector can be found are not part of the result.

Pay special attention to operator order when chaining arithmetic operators.

Arithmetic Examples

Implement a health check with a simple query:

```
1 + 1
```

Double the rate of a log stream's entries:

```
sum(rate({app="foo"}[1m])) * 2
```

Get proportion of warning logs to error logs for the `foo` app

```
sum(rate({app="foo", level="warn"}[1m])) / sum(rate({app="foo", level="error"}[1m]))
```

Logical and set operators

These logical/set binary operators are only defined between two vectors:

- `and` (intersection)
- `or` (union)
- `unless` (complement)

`vector1 and vector2` results in a vector consisting of the elements of vector1 for which there are elements in vector2 with exactly matching label sets. Other elements are dropped.

`vector1 or vector2` results in a vector that contains all original elements (label sets + values) of vector1 and additionally all elements of vector2 which do not have matching label sets in vector1.

`vector1 unless vector2` results in a vector consisting of the elements of vector1 for which there are no elements in vector2 with exactly matching label sets. All matching elements in both vectors are dropped.

Binary operators examples

This contrived query will return the intersection of these queries, effectively `rate({app="bar"})`:

```
rate({app=~"foo|bar"}[1m]) and rate({app="bar"}[1m])
```

Comparison operators

- `==` (equality)
- `!=` (inequality)
- `>` (greater than)
- `>=` (greater than or equal to)
- `<` (less than)
- `<=` (less than or equal to)

Comparison operators are defined between scalar/scalar, vector/scalar, and vector/vector value pairs. By default they filter. Their behavior can be modified by providing `bool` after the operator, which will return 0 or 1 for the value rather than filtering.

Between two scalars, these operators result in another scalar that is either 0 (false) or 1 (true), depending on the comparison result. The `bool` modifier must **not** be provided.

`1 >= 1` is equivalent to `1`

Between a vector and a scalar, these operators are applied to the value of every data sample in the vector, and vector elements between which the comparison result is false get dropped from the result vector. If the `bool` modifier is provided, vector elements that would be dropped instead have the value 0 and vector elements that would be kept have the value 1.

Filters the streams which logged at least 10 lines in the last minute:

```
count_over_time({foo="bar"}[1m]) > 10
```

Attach the value(s) `0`/`1` to streams that logged less/more than 10 lines:

```
count_over_time({foo="bar"}[1m]) > bool 10
```

Between two vectors, these operators behave as a filter by default, applied to matching entries. Vector elements for which the expression is not true or which do not find a match on the other side of the expression get dropped from the result, while the others are propagated into a result vector. If the `bool` modifier is provided, vector elements that would have been dropped instead have the value 0 and vector elements that would be kept have the value 1, with the grouping labels again becoming the output label set.

Return the streams matching `app=foo` without app labels that have higher counts within the last minute than their counterparts matching `app=bar` without app labels:

```
sum without(app) (count_over_time({app="foo"}[1m])) > sum without(app)
(count_over_time({app="bar"}[1m]))
```

Same as above, but vectors have their values set to `1` if they pass the comparison or `0` if they fail/would otherwise have been filtered out:

```
sum without(app) (count_over_time({app="foo"}[1m])) > bool sum without(app)
(count_over_time({app="bar"}[1m]))
```

Pattern match filter operators

- `|>` (line match pattern)
- `!>` (line match not pattern)

Pattern Filter not only enhances efficiency but also simplifies the process of writing LogQL queries. By eliminating the need for complex regex patterns, users can create queries using a more intuitive syntax, reducing the cognitive load and potential for errors.

Within the pattern syntax the `<_>` serves as a wildcard, representing any arbitrary text. This allows the query to match log lines where the specified pattern occurs, such as log lines containing static content, with variable content in between.

Line match pattern example:

```
{service_name=`distributor`} |> `<_> caller=http.go:194 level=debug <_> msg="POST
/push.v1.PusherService/Push <_>`
```

Line match not pattern example:

```
{service_name=`distributor`} !> `<_> caller=http.go:194 level=debug <_> msg="POST
/push.v1.PusherService/Push <_>`
```

For example, the example queries above will respectively match and not match the following log line from the `distributor` service:

```
ts=2024-04-05T08:40:13.585911094Z caller=http.go:194 level=debug traceID=23e54a271db607cc
orgID=3648 msg="POST /push.v1.PusherService/Push (200) 12.684035ms"
ts=2024-04-05T08:41:06.551403339Z caller=http.go:194 level=debug traceID=54325a1a15b42e2d
orgID=1218 msg="POST /push.v1.PusherService/Push (200) 1.664285ms"
ts=2024-04-05T08:41:06.506524777Z caller=http.go:194 level=debug traceID=69d4271da1595bcb
orgID=1218 msg="POST /push.v1.PusherService/Push (200) 1.783818ms"
ts=2024-04-05T08:41:06.473740396Z caller=http.go:194 level=debug traceID=3b8ec973e6397814
orgID=3648 msg="POST /push.v1.PusherService/Push (200) 1.893987ms"
ts=2024-04-05T08:41:05.88999067Z caller=http.go:194 level=debug traceID=6892d7ef67b4d65c
orgID=3648 msg="POST /push.v1.PusherService/Push (200) 2.314337ms"
ts=2024-04-05T08:41:05.826266414Z caller=http.go:194 level=debug traceID=0bb76e910cfd008d
orgID=3648 msg="POST /push.v1.PusherService/Push (200) 3.625744ms"
```

Order of operations

When chaining or combining operators, you have to consider operator precedence: Generally, you can assume regular mathematical convention with operators on the same precedence level being left-associative.

More details can be found in the Golang language documentation.

`1 + 2 / 3` is equal to `1 + ( 2 / 3 )`.

`2 * 3 % 2` is evaluated as `(2 * 3) % 2`.

Keywords on and ignoring

The `ignoring` keyword causes specified labels to be ignored during matching. The syntax:

```
<vector expr> <bin-op> ignoring(<labels>) <vector expr>
```

This example will return the machines which total count within the last minutes exceed average value for app `foo`.

```
max by(machine) (count_over_time({app="foo"}[1m])) > bool ignoring(machine)
avg(count_over_time({app="foo"}[1m]))
```

The on keyword reduces the set of considered labels to a specified list. The syntax:

```
<vector expr> <bin-op> on(<labels>) <vector expr>
```

This example will return every machine total count within the last minutes ratio in app `foo`:

```
sum by(machine) (count_over_time({app="foo"}[1m])) / on()
sum(count_over_time({app="foo"}[1m]))
```

Many-to-one and one-to-many vector matches

Many-to-one and one-to-many matchings occur when each vector element on the "one"-side can match with multiple elements on the "many"-side. You must explicitly request matching by using the group_left or group_right modifier, where left or right determines which vector has the higher cardinality. The syntax:

```
<vector expr> <bin-op> ignoring(<labels>) group_left(<labels>) <vector expr>
<vector expr> <bin-op> ignoring(<labels>) group_right(<labels>) <vector expr>
<vector expr> <bin-op> on(<labels>) group_left(<labels>) <vector expr>
<vector expr> <bin-op> on(<labels>) group_right(<labels>) <vector expr>
```

The label list provided with the group modifier contains additional labels from the "one"-side that are included in the result metrics. And a label should only appear in one of the lists specified by `on` and `group_x`. Every time series of the result vector must be uniquely identifiable. Grouping modifiers can only be used for comparison and arithmetic. By default, the system matches `and`, `unless`, and `or` operations with all entries in the right vector.

The following example returns the rates requests partitioned by `app` and `status` as a percentage of total requests.

```
sum by (app, status) (
  rate(
    {job="http-server"}
      | json
      [5m]
  )
)
/ on (app) group_left
sum by (app) (
  rate(
    {job="http-server"}
      | json
      [5m]
  )
)

=>
[
  {app="foo", status="200"} => 0.8
  {app="foo", status="400"} => 0.1
  {app="foo", status="500"} => 0.1
]
```

This version uses `group_left(<labels>)` to include `<labels>` from the right hand side in the result and returns the cost of discarded events per user, organization, and namespace:

```
sum by (user, namespace) (
  rate(
    {job="events"}
      | logfmt
      | discarded="true"
      [5m]
```

```
  )
)
* on (user) group_left(organization)
max_over_time(
  {job="cost-calculator"}
    | logfmt
    | unwrap cost
    [5m]
) by (user, organization)

=>
[
  {user="foo", namespace="dev", organization="little-org"} => 10
  {user="foo", namespace="prod", organization="little-org"} => 50
  {user="bar", namespace="dev", organization="big-org"} => 70
  {user="bar", namespace="prod", organization="big-org"} => 200
]
```

Comments

LogQL queries can be commented using the `#` character:

```
{app="foo"} # anything that comes after will not be interpreted in your query
```

With multi-line LogQL queries, the query parser can exclude whole or partial lines using `#`:

```
{app="foo"}
    | json
    # this line will be ignored
    | bar="baz" # this checks if bar = "baz"
```

Pipeline Errors

There are multiple reasons which cause pipeline processing errors, such as:

- A numeric label filter may fail to turn a label value into a number
- A metric conversion for a label may fail.
- A log line is not a valid json document.

- etc...

When those failures happen, Loki won't filter out those log lines. Instead they are passed into the next stage of the pipeline with a new system label named `__error__`. The only way to filter out errors is by using a label filter expressions. The `__error__` label can't be renamed via the language.

For example to remove json errors:

```
{cluster="ops-tools1",container="ingress-nginx"}
  | json
  | __error__ != "JSONParserErr"
```

Alternatively you can remove all error using a catch all matcher such as `__error__ = ""` or even show only errors using `__error__ != ""`.

Pipeline Errors in Metric Queries

A metric query fails when a sample carries `__error__`, and returns an error with the appropriate status code. This stops an aggregation from silently miscounting the lines that failed. A label filter on `__error__` says what to do with those lines instead:

- `| __error__=""` drops them, so the aggregation covers only the lines that succeeded.
- `| __error__!=""` keeps only them, which is how you count or alert on parse failures.
- `| __error__=~".*"` keeps every line, errored or not.

The `__error__` filter should be placed after the stage that generated this error. This means if you need to remove errors from an unwrap expression it needs to be placed after the unwrap. For example:

```
quantile_over_time(
  0.99,
  {container="ingress-nginx",service="hosted-grafana"}
  | json
  | unwrap response_latency_seconds
  | __error__=""[1m]
  ) by (cluster)
```

A range aggregation's `by (...)` (e.g. `count_over_time()`) keeps the error labels next to the labels it names, so an errored sample still reports `__error__`. `without (...)` behaves the same way: it keeps the error labels alongside whatever labels it doesn't exclude. A vector aggregation's `by (...)` (e.g. `count()`) drops them, like any other label it does not name.

In a `__error__` filter joined by `and` or `or`, only a part that asks to keep the errored lines keeps them:

- `| __error__="" or pod="p2"` asks to drop them, so an errored line that the `pod="p2"` part re-admits fails the query rather than being counted
- `| __error__="" and pod="p2"` asks to drop them too, and `and` applies both parts, so the errored lines never reach the aggregation and the query runs

The `__error__` filter applies to every error a pipeline can raise, not only a parser error. A line whose `| unwrap` conversion failed has no number to report, so its sample reports `0` next to the error labels; an outer aggregation reads that `0` like any other value, so check the `__error__` label before aggregating further.

Functions

Loki supports functions to operate on data.

label_replace()

For each time series in `v`,

```
label_replace(v instant-vector,
    dst_label string,
    replacement string,
    src_label string,
    regex string)
```

matches the regular expression `regex` against the label `src_label`. If it matches, then the time series is returned with the label `dst_label` replaced by the expansion of `replacement`.

`$1` is replaced with the first matching subgroup, `$2` with the second etc. If the regular expression doesn't match, then the time series is returned unchanged.

This example will return a vector with each time series having a `foo` label with the value `a` added to it:

```
label_replace(rate({job="api-server",service="a:c"} |= "err" [1m]), "foo", "$1",
  "service", "(.*):.*")
```

LogQL template functions

The Go templating language is embedded in the Loki query language, LogQL. The text template format used in `| line_format` and `| label_format` support the usage of functions.

{{< admonition type="note" >}} In the examples below we use backticks to quote the template strings. This is because some template strings contain double quotes, and using backticks lets us avoid escaping the double quotes. If you are using a different quoting style, you may need to escape the double quotes. {{< /admonition >}}

For more information, refer to the Go template documentation.

Additionally you can also access the log line using the `__line__` function and the timestamp using the `__timestamp__` function.

Template pipeline syntax

A pipeline is a possibly chained sequence of "commands". A command is a simple value (argument) or a function or method call, possibly with multiple arguments. A pipeline may be "chained" by separating a sequence of commands with pipeline characters '|'. In a chained pipeline, the result of each command is passed as the last argument of the following command. The output of the final command in the pipeline is the value of the pipeline.

You can take advantage of pipeline to join together multiple functions.

Example:

```
`{{ .path | replace " " "_" | trunc 5 | upper }}`
```

For function that returns a `bool` such as `contains`, `eq`, `hasPrefix` and `hasSuffix`, you can apply `AND` / `OR` and nested `if` logic.

Example:

```
`{{ if and (contains "he" "hello") (contains "llo" "hello") }} yes {{end}}`
`{{ if or (contains "he" "hello") (contains("llo" "hello") }} yes {{end}}`
`{{ if contains "ErrTimeout" .err }} timeout {{else if contains "he" "hello"}} yes {{else}} no
{{end}}`
```

Built-in variables for log line properties

These variables provide a way of referencing something from the log line when writing a template expression.

.label_name

All labels from the Log line are added as variables in the template engine. They can be referenced using the label name prefixed by a `.`(for example,`.label_name`). For example the following template will output the value of the `path` label:

```
`{{ .path }}`
```

line

The `__line__` function returns the original log line without any modifications.

Signature: `line() string`

Examples:

```
`{{ __line__ | lower }}`
`{{ __line__ }}`
```

timestamp

The `__timestamp__` function returns the current log line's timestamp.

Signature: `timestamp() time.Time`

```
`{{ __timestamp__ }}`
`{{ __timestamp__ | date "2006-01-02T15:04:05.00Z-07:00" }}`
`{{ __timestamp__ | unixEpoch }}`
```

For more information, refer to the blog Parsing and formatting date/time in Go.

Date and time

You can use the following functions to manipulate dates and times when building LogQL queries.

date

Returns a textual representation of the time value formatted according to the provided golang datetime layout.

Signature: `date(fmt string, date interface{}) string`

Example:

```
`{{ date "2006-01-02" now }}`
```

duration

An alias for `duration_seconds`

Examples:

```
`{{ .foo | duration }}`
`{{ duration .foo }}`
```

duration_seconds

Convert a humanized time duration to seconds using time.ParseDuration

Signature: `duration_seconds(string) float64`

Examples:

```
`{{ .foo | duration_seconds }}`
`{{ duration_seconds .foo }}`
```

now

Returns the current time in the local timezone of the Loki server.

Signature: `Now() time.Time`

Example:

```
`{{ now }}`
```

toDate

Parses a formatted string and returns the time value it represents using the local timezone of the server running Loki.

For more consistency between Loki installations, it's recommended to use `toDateInZone`

The format string must use the exact date as defined in the golang datetime layout

Signature: `toDate(fmt, str string) time.Time`

Examples:

```
`{{ toDate "2006-01-02" "2021-11-02" }}`
`{{ .foo | toDate "2006-01-02T15:04:05.999999999Z" }}`
```

toDateInZone

Parses a formatted string and returns the time value it represents in the provided timezone.

The format string must use the exact date as defined in the golang datetime layout

The timezone value can be `Local`, `UTC`, or any of the IANA Time Zone database values

Signature: `toDateInZone(fmt, zone, str string) time.Time`

Examples:

```
`{{ toDateInZone "2006-01-02" "UTC" "2021-11-02" }}`
`{{ .foo | toDateInZone "2006-01-02T15:04:05.999999999Z" "UTC" }}`
```

unixEpoch

Returns the number of seconds elapsed since January 1, 1970 UTC.

Signature: `unixEpoch(date time.Time) string`

Examples:

```
`{{ unixEpoch now }}`
`{{ .foo | toDateInZone "2006-01-02T15:04:05.999999999Z" "UTC" | unixEpoch }}`
```

Example of a query to filter Loki querier jobs which create time is 1 day before:

```
{job="loki/querier"} | label_format nowEpoch=`{{(unixEpoch
now)}}`,createDateEpoch=`{{unixEpoch (toDate "2006-01-02" .createDate)}}` | label_format
dateTimeDiff=`{{sub .nowEpoch .createDateEpoch}}` | dateTimeDiff > 86400
```

unixEpochMillis

Returns the number of milliseconds elapsed since January 1, 1970 UTC.

Signature: `unixEpochMillis(date time.Time) string`

Examples:

```
`{{ unixEpochMillis now }}`
`{{ .foo | toDateInZone "2006-01-02T15:04:05.999999999Z" "UTC" | unixEpochMillis }}`
```

unixEpochNanos

Returns the number of nanoseconds elapsed since January 1, 1970 UTC.

Signature: `unixEpochNanos(date time.Time) string`

Examples:

```
`{{ unixEpochNanos now }}`
`{{ .foo | toDateInZone "2006-01-02T15:04:05.999999999Z" "UTC" | unixEpochNanos }}`
```

unixToTime

Converts the string epoch to the time value it represents. Epoch times in days, seconds, milliseconds, microseconds and nanoseconds are supported.

Signature: `unixToTime(epoch string) time.Time`

Examples:

Consider the following log line `{"from": "1679577215","to":"1679587215","message":"some message"}`. To print the `from` field as human readable add the following at the end of the LogQL query:

```
... | json | line_format `from="{{date "2006-01-02" (unixToTime .from)}}"`
```

String manipulation

You can use the following templates to manipulate strings when building LogQL Queries.

alignLeft

Use this function to format a string to a fixed with, aligning the content to the left.

Signature: `alignLeft(count int, src string) string`

Examples:

```
`{{ alignLeft 5 "hello world"}}` // output: "hello"
`{{ alignLeft 5 "hi"}}`          // output: "hi   "
```

alignRight

Use this function to format a string to a fixed with, aligning the content to the right.

Signature: `alignRight(count int, src string) string`

Examples:

```
`{{ alignRight 5 "hello world"}}` // output: "world"
`{{ alignRight 5 "hi"}}`          // output: "   hi"
```

b64enc

Base64 encode a string.

Signature: `b64enc(string) string`

Examples:

```
`{{ .foo | b64enc }}`
`{{ b64enc  .foo }}`
```

b64dec

Base64 decode a string. Inputs that omit trailing `=` padding are padded before decoding.

Signature: `b64dec(string) string`

Examples:

```
`{{ .foo | b64dec }}`
`{{ b64dec  .foo }}`
```

bytes

Convert a humanized byte string to bytes using go-humanize. Durations can be turned into strings such as "3 days ago", numbers representing sizes like 82854982 into useful strings like, "83 MB" or "79 MiB"

Signature: `bytes(string) string`

Examples:

```
`{{ .foo | bytes }}`
`{{ bytes .foo }}`
```

default

Enables outputting a default value if the source string is otherwise empty. If the 'src' parameter is not empty, this function returns the value of 'src'. Useful for JSON fields that can be missing, like HTTP headers in a log line that aren't required, as in the following example:

```
{job="access_log"} | json | line_format `{{.http_request_headers_x_forwarded_for | default "-"}}`
```

Signature: `default(d string, src string) string`

Examples:

```
`{{ default "-" "" }}` // output: -
`{{ default "-" "foo" }}` // output: foo
```

Example of a query to print a `-` if the `http_request_headers_x_forwarded_for` label is empty:

```
{job="access_log"} | json | line_format `{{.http_request_headers_x_forwarded_for | default "-"}}`
```

fromJson

Decodes a JSON document into a structure. If the input cannot be decoded as JSON the function will return an empty string.

Signature: `fromJson(v string) interface{}`

Example:

```
`{{fromJson "{\"foo\": 55}"}}`
```

Example of a query to print a newline per queries stored as a json array in the log line:

```
{job="loki/querier"} |= "finish in prometheus" | logfmt | line_format `{{ range $q := fromJson .queries }} {{ $q.query }} {{ end }}`
```

lower

Use this function to convert to lower case.

Signature: `lower(string) string`

Examples:

```
`{{ .request_method | lower }}`
`{{ lower  "HELLO"}}`
```

The last example will return `hello`.

indent

The indent function indents every line in a given string to the specified indent width. This is useful when aligning multi-line strings.

Signature: `indent(spaces int,src string) string`

Example:

```
`{{ indent 4 .query }}`
```

This indents each line contained in the `.query` by four (4) spaces.

nindent

The nindent function is the same as the indent function, but prepends a new line to the beginning of the string.

Signature: `nindent(spaces int,src string) string`

Example:

```
`{{ nindent 4 .query }}`
```

This will indent every line of text by 4 space characters and add a new line to the beginning.

repeat

Use this function to repeat a string multiple times.

Signature: `repeat(c int,value string) string`

Example:

```
`{{ repeat 3 "hello" }}` // output: hellohellohello
```

printf

Use this function to format a string in a custom way. For more information about the syntax, refer to the Go documentation.

Signature: `printf(format string, a ...interface{})`

Examples:

```
`{{ printf "The IP address was %s" .remote_addr }}` // output: The IP address was 129.168.1.1


`{{ printf "%-40.40s" .request_uri}} {{printf "%-5.5s" .request_method}}`
// output:
// /a/509965767/alternative-to-my-mtg.html  GET
// /id/609259548/hpr.html                   GET
```

```
line_format "\"|\" {{printf \"%15.15s\" .ClientHost}} \"|\""
```

replace

This function performs simple string replacement.

Signature: `replace(old string, new string, src string) string`

It takes three arguments:

- `old` string to replace
- `new` string to replace with
- `src` source string

Examples:

```
`{{ .cluster | replace "-cluster" "" }}`
`{{ replace "hello" "world" "hello world" }}`
```

The last example will return `world world`.

substr

Get a substring from a string.

Signature: `substr(start int,end int,value string) string`

If start is < 0, this calls value[:end]. If start is >= 0 and end < 0 or end bigger than s length, this calls value[start:] Otherwise, this calls value[start, end].

Examples:

```
`{{ .path | substr 2 5 }}`
`{{ substr 0 5 "hello world"}}`  // output: hello
`{{ substr 6 11 "hello world"}}` // output: world
```

title

Convert to title case.

Signature: `title(string) string`

Examples:

```
`{{.request_method | title}}`
`{{ title "hello world"}}`
```

The last example will return `Hello World`.

trim

The trim function removes space from either side of a string.

Signature: `trim(string) string`

Examples:

```
`{{ .ip | trim }}`
`{{ trim "   hello    " }}` // output: hello
```

trimAll

Use this function to remove given characters from the front or back of a string.

Signature: `trimAll(chars string,src string) string`

Examples:

```
`{{ .path | trimAll "/" }}`
`{{ trimAll "$" "$5.00" }}` // output: 5.00
```

trimPrefix

Use this function to trim just the prefix from a string.

Signature: `trimPrefix(prefix string, src string) string`

Examples:

```
`{{  .path | trimPrefix "/" }}`
`{{ trimPrefix "-" "-hello" }}` // output: hello
```

trimSuffix

Use this function to trim just the suffix from a string.

Signature: `trimSuffix(suffix string, src string) string`

Examples:

```
`{{  .path | trimSuffix "/" }}`
`{{ trimSuffix "-" "hello-" }}` // output: hello
```

trunc

Truncate a string and add no suffix.

Signature: `trunc(count int,value string) string`

Examples:

```
`{{ .path | trunc 2 }}`
`{{ trunc 5 "hello world"}}`   // output: hello
`{{ trunc -5 "hello world"}}`  // output: world
```

upper

Use this function to convert to upper case.

Signature: `upper(string) string`

Examples:

```
`{ .request_method | upper }}`
`{{ upper  "hello"}}`
```

This results in `HELLO`.

urlencode

Use this function to urlencode a string.

Signature: `urlencode(string) string`

Examples:

```
`{{ .request_url | urlencode }}`
`{{ urlencode  .request_url}}`
```

urldecode

Use this function to urldecode a string.

Signature: `urldecode(string) string`

Examples:

```
`{{ .request_url | urldecode }}`
`{{ urldecode  .request_url}}`
```

Logical functions

You can use the following logical functions to compare strings when building a template expression.

contains

Use this function to test to see if one string is contained inside of another.

Signature: `contains(s string, src string,) bool`

Examples:

```
`{{ if contains "ErrTimeout" .err }} timeout {{end}}`
`{{ if contains "he" "hello" }} yes {{end}}`
```

eq

Use this function to test to see if one string has exact matching inside of another.

Signature: `eq(s string, src string) bool`

Examples:

```
`{{ if eq "ErrTimeout" .err }} timeout {{end}}`
`{{ if eq "hello" "hello" }} yes {{end}}`
```

hasPrefix and hasSuffix

The `hasPrefix` and `hasSuffix` functions test whether a string has a given prefix or suffix.

Signatures:

- `hasPrefix(prefix string, src string) bool`
- `hasSuffix(suffix string, src string) bool`

Examples:

```
`{{ if hasSuffix .err "Timeout" }} timeout {{end}}`
`{{ if hasPrefix "he" "hello" }} yes {{end}}`
```

Mathematical functions

You can use the following mathematical functions when writing template expressions.

add

Sum numbers. Supports multiple numbers

Signature: `func(i ...interface{}) int64`

Example:

```
`{{ add 3 2 5 }}` // output: 10
```

addf

Sum floating point numbers. Supports multiple numbers.

Signature: `func(i ...interface{}) float64`

Example:

```
`{{ addf 3.5 2 5 }}` // output: 10.5
```

ceil

Returns the greatest float value greater than or equal to input value

Signature: `ceil(a interface{}) float64`

Example:

```
`{{ ceil 123.001 }}` //output 124.0
```

div

Divide two integers.

Signature: `func(a, b interface{}) int64`

Example:

```
`{{ div 10 2}}` // output: 5
```

divf

Divide floating point numbers. Supports multiple numbers.

Signature: `func(a interface{}, v ...interface{}) float64`

Example:

```
`{{ divf 10 2 4}}` // output: 1.25
```

float64

Convert a string to a float64.

Signature: `toFloat64(v interface{}) float64`

Example:

```
`{{ "3.5" | float64 }}` //output 3.5
```

floor

Returns the greatest float value less than or equal to input value.

Signature: `floor(a interface{}) float64`

Example:

```
`{{ floor 123.9999 }}` //output 123.0
```

int

Convert value to an integer.

Signature: `toInt(v interface{}) int`

Example:

```
`{{ "3" | int }}` //output 3
```

max

Return the largest of a series of integers:

Signature: `max(a interface{}, i ...interface{}) int64`

Example:

```
`{{ max 1 2 3 }}` //output 3
```

maxf

Return the largest of a series of floats:

Signature: `maxf(a interface{}, i ...interface{}) float64`

Example:

```
`{{ maxf 1 2.5 3 }}` //output 3
```

min

Return the smallest of a series of integers.

Signature: `min(a interface{}, i ...interface{}) int64`

Example:

```
`{{ min 1 2 3 }}`//output 1
```

minf

Return the smallest of a series of floats.

Signature: `minf(a interface{}, i ...interface{}) float64`

Example:

```
`{{ minf 1 2.5 3 }}` //output 1.5
```

mul

Multiply numbers. Supports multiple numbers.

Signature: `mul(a interface{}, v ...interface{}) int64`

Example:

```
`{{ mul 5 2 3}}` // output: 30
```

mulf

Multiply floating numbers. Supports multiple numbers

Signature: `mulf(a interface{}, v ...interface{}) float64`

Example:

```
`{{ mulf 5.5 2 2.5 }}` // output: 27.5
```

mod

Returns the remainder when number 'a' is divided by number 'b'.

Signature: `mod(a, b interface{}) int64`

Example:

```
`{{ mod 10 3}}` // output: 1
```

round

Returns a float value with the remainder rounded to the given number of digits after the decimal point.

Signature: `round(a interface{}, p int, rOpt ...float64) float64`

Example:

```
`{{ round 123.555555 3 }}` //output 123.556
```

We can also provide a `roundOn` number as third parameter

Example:

```
`{{ round 123.88571428571 5 .2 }}` //output 123.88572
```

With default `roundOn` of `.5` the above value would be `123.88571`

sub

Subtract one number from another.

Signature: `func(a, b interface{}) int64`

Example:

```
`{{ sub 5 2 }}` // output: 3
```

subf

Subtract floating numbers. Supports multiple numbers.

Signature: `func(a interface{}, v ...interface{}) float64`

Example:

```
`{{ subf  5.5 2 1.5 }}` // output: 2
```

Regular expression functions

You can use the following functions to perform regular expressions in a template expression.

count

Counts occurrences of the regex (`regex`) in (`src`).

Signature: `count(regex string, src string) int`

Examples:

```
`{{ count "a|b" "abab" }}` // output: 4
`{{ count "o" "foo" }}`    // output: 2
```

Example of a query to print how many times XYZ occurs in a line:

```
{job="xyzlog"} | line_format `{{ __line__ | count "XYZ"}}`
```

regexReplaceAll and regexReplaceAllLiteral

`regexReplaceAll` returns a copy of the input string, replacing matches of the Regexp with the replacement string replacement. Inside string replacement, $ signs are interpreted as in Expand, so for instance $1 represents the text of the first sub-match. See the golang Regexp.replaceAll documentation for more examples.

Signature: regexReplaceAll(regex string, src string, replacement string) (source)

Example:

```
`{{ regexReplaceAll "(a*)bc" .some_label "${1}a" }}`
```

`regexReplaceAllLiteral` function returns a copy of the input string and replaces matches of the Regexp with the replacement string replacement. The replacement string is substituted directly, without using Expand.

Signature: regexReplaceAllLiteral(regex string, src string, replacement string)

Example:

```
`{{ regexReplaceAllLiteral "(ts=)" .timestamp "timestamp=" }}`
```

Troubleshoot log queries (READ)

{{< docs/shared lookup="troubleshoot-query.md" source="loki" version="<LOKI_VERSION>" >}}

LogCLI getting started

logcli is a command-line client for Loki that lets you run LogQL queries against your Loki instance. The `query` command will output extra information about the query and its results, such as the API URL, set of common labels, and set of excluded labels.

This is useful, for example, if you want to download a range of logs from Loki. Or want to perform analytical administration tasks, for example, discover the number of log streams to understand your label cardinality, or find out the estimated volume of data that a query will search over. You can also use logcli as part of shell scripts.

If you are a Grafana Cloud user, you can also use logcli to query logs that you have exported to long-term storage with Cloud Logs Export, or any other Loki formatted log data.

{{< admonition type="note" >}} Note that logcli is a querying tool, it cannot be used to ingest logs. {{< /admonition >}}

Install logcli

As a best practice, you should download the version of logcli that matches your Loki version. And upgrade your logcli when you upgrade your version of Loki.

Binary (Recommended)

Download the `logcli` binary from the Loki releases page.

Builds are available for Linux, Mac, and Windows.

Build LogCLI from source

Clone the Loki repository and build `logcli` from source:

```
git clone https://github.com/grafana/loki.git
cd loki
make logcli
```

Optionally, move the binary into a directory that is part of your `$PATH`.

```
cp cmd/logcli/logcli /usr/local/bin/logcli
```

Set up command completion

You can set up tab-completion for `logcli` with one of the two options, depending on your shell:

- For bash, add this to your `~/.bashrc` file:

```
eval "$(logcli --completion-script-bash)"
```

- For zsh, add this to your `~/.zshrc` file:

```
eval "$(logcli --completion-script-zsh)"
```

LogCLI usage

Once you have installed logcli, you can run it in the following way:

`logcli <command> [<flags>, <args> ...]`

`<command>` points to one of the commands, detailed in the command reference below.

`<flags>` is one of the subcommands available for each command.

`<args>` is a list of space separated arguments. Arguments can optionally be overridden using environment variables. Environment variables will always take precedence over command line arguments.

Authenticate to Loki

To connect to a Loki instance, set the following argument:

- `--addr=http://loki.example.com:3100` or the `LOKI_ADDR` environment variable

For example, to query a local Loki instance directly without needing a username and password:

```
export LOKI_ADDR=http://localhost:3100

logcli query '{service_name="website"}'
```

To connect to a Loki instance which requires authentication, you will need to additionally set the following arguments:

- `--username` or the `LOKI_USERNAME` environment variable
- `--password` or the `LOKI_PASSWORD` environment variable

For example, to query Grafana Cloud:

```
export LOKI_ADDR=https://logs-us-west1.grafana.net
export LOKI_USERNAME=<username>
export LOKI_PASSWORD=<password>

logcli query '{service_name="website"}'
```

To specify a particular tenant, set the following argument:

- `--org-id` or the `LOKI_ORG_ID` environment variable

{{< admonition type="note" >}} If you are running Loki behind a proxy server and you have authentication configured, you will also have to pass in LOKI_USERNAME and LOKI_PASSWORD, LOKI_BEARER_TOKEN or LOKI_BEARER_TOKEN_FILE accordingly. {{< /admonition >}}

LogCLI command reference

The output of `logcli help`:

```
usage: logcli [<flags>] <command> [<args> ...]

A command-line for loki.


Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
                              and --help-man).
      --[no-]version          Show application version.
  -q, --[no-]quiet            Suppress query metadata
```

```
    --[no-]stats            Show query statistics
-o, --output=default        Specify output mode [default, raw, jsonl].
                            raw suppresses log labels and timestamp.
-z, --timezone=Local        Specify the timezone to use when formatting output
                            timestamps [Local, UTC]
    --output-timestamp-format=rfc3339
                            Specify the format of timestamps in the default
                            output mode [rfc3339, rfc3339nano, rfc822z,
                            rfc1123z, stampmicro, stampmilli, stampnano,
                            unixdate]
    --cpuprofile=""         Specify the location for writing a CPU profile.
    --memprofile=""         Specify the location for writing a memory profile.
    --[no-]stdin            Take input logs from stdin
    --addr="http://localhost:3100"
                            Server address. Can also be set using LOKI_ADDR
                            env var. ($LOKI_ADDR)
    --username=""           Username for HTTP basic auth. Can also be set
                            using LOKI_USERNAME env var. ($LOKI_USERNAME)
    --password=""           Password for HTTP basic auth. Can also be set
                            using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
    --ca-cert=""            Path to the server Certificate Authority.
                            Can also be set using LOKI_CA_CERT_PATH env var.
                            ($LOKI_CA_CERT_PATH)
    --[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
                            be set using LOKI_TLS_SKIP_VERIFY env var.
                            ($LOKI_TLS_SKIP_VERIFY)
    --cert=""               Path to the client certificate. Can also
                            be set using LOKI_CLIENT_CERT_PATH env var.
                            ($LOKI_CLIENT_CERT_PATH)
    --key=""                Path to the client certificate key. Can also
                            be set using LOKI_CLIENT_KEY_PATH env var.
                            ($LOKI_CLIENT_KEY_PATH)
    --org-id=""             adds X-Scope-OrgID to API requests for
                            representing tenant ID. Useful for requesting
                            tenant data when bypassing an auth gateway.
                            Can also be set using LOKI_ORG_ID env var.
                            ($LOKI_ORG_ID)
    --query-tags=""         adds X-Query-Tags http header to API requests.
                            This header value will be part of `metrics.go`
```

```
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
  --[no-]nocache        adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
  --bearer-token=""     adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
  --bearer-token-file=""  adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
  --retries=0           How many times to retry each query when
                        getting an error response from Loki. Can also
                        be set using LOKI_CLIENT_RETRIES env var.
                        ($LOKI_CLIENT_RETRIES)
  --min-backoff=0       Minimum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MIN_BACKOFF env var.
                        ($LOKI_CLIENT_MIN_BACKOFF)
  --max-backoff=0       Maximum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MAX_BACKOFF env var.
                        ($LOKI_CLIENT_MAX_BACKOFF)
  --auth-header="Authorization"
                        The authorization header used. Can also
                        be set using LOKI_AUTH_HEADER env var.
                        ($LOKI_AUTH_HEADER)
  --proxy-url=""        The http or https proxy to use when
                        making requests. Can also be set
                        using LOKI_HTTP_PROXY_URL env var.
                        ($LOKI_HTTP_PROXY_URL)
  --[no-]compress       Request that Loki compress returned
                        data in transit. Can also be set
                        using LOKI_HTTP_COMPRESSION env var.
                        ($LOKI_HTTP_COMPRESSION)
  --[no-]envproxy       Use ProxyFromEnvironment to use net/http
                        ProxyFromEnvironment configuration, eg HTTP_PROXY
                        ($LOKI_ENV_PROXY)
  --log.level           Only log messages with the given severity or above. Valid
```

```
levels: [debug, info, warn, error]

Commands:
help [<command>...]
    Show help.

query [<flags>] <query>
    Run a LogQL query.

    The "query" command is useful for querying for logs. Logs can be returned in
    a few output modes:

      raw: log line
      default: log timestamp + log labels + log line
      jsonl: JSON response from Loki API of log line

    The output of the log can be specified with the "-o" flag, for example,
    "-o raw" for the raw output format.

    The "query" command will output extra information about the query and its
    results, such as the API URL, set of common labels, and set of excluded
    labels. This extra information can be suppressed with the --quiet flag.

    By default we look over the last hour of data; use --since to modify or
    provide specific start and end times with --from and --to respectively.

    Notice that when using --from and --to then ensure to use RFC3339Nano time
    format, but without timezone at the end. The local timezone will be added
    automatically or if using --timezone flag. In default output mode the
    --output-timestamp-format flag can be used to modify the output timestamp.

    Example:

      logcli query
         --timezone=UTC
         --from="2021-01-19T10:00:00Z"
         --to="2021-01-19T20:00:00Z"
         --output=jsonl
         'my-query'
```

```
Example with --output-timestamp-format:

  logcli query
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
     --output-timestamp-format=rfc3339nano
     'my-query'

The output is limited to 30 entries by default; use --limit to increase.

While "query" does support metrics queries, its output contains multiple
data points between the start and end query time. This output is used to
build graphs, similar to what is seen in the Grafana Explore graph view.
If you are querying metrics and just want the most recent data point (like
what is seen in the Grafana Explore table view), then you should use the
"instant-query" command instead.

Parallelization:

You can download an unlimited number of logs in parallel, there are a few
flags which control this behaviour:

  --parallel-duration
  --parallel-max-workers
  --part-path-prefix
  --overwrite-completed-parts
  --merge-parts
  --keep-parts

Refer to the help for each flag for details about what each of them do.

Example:

  logcli query
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
```

```
    --output=jsonl
    --parallel-duration="15m"
    --parallel-max-workers="4"
    --part-path-prefix="/tmp/my_query"
    --merge-parts
    'my-query'
```

This example will create a queue of jobs to execute, each being 15 minutes in duration. In this case, that means, for the 10-hour total duration, there will be forty 15-minute jobs. The --limit flag is ignored.

It will start four workers, and they will each take a job to work on from the queue until all the jobs have been completed.

Each job will save a "part" file to the location specified by the --part-path-prefix. Different prefixes can be used to run multiple queries at the same time. The timestamp of the start and end of the part is in the file name. While the part is being downloaded, the filename will end in ".part", when it is complete, the file will be renamed to remove this ".part" extension. By default, if a completed part file is found, that part will not be downloaded again. This can be overridden with the --overwrite-completed-parts flag.

Part file example using the previous command, adding --keep-parts so they are not deleted:

Since we don't have the --forward flag, the parts will be downloaded in reverse. Two of the workers have finished their jobs (last two files), and have picked up the next jobs in the queue. Running ls, this is what we should expect to see.

```
$ ls -1 /tmp/my_query*
/tmp/my_query_20210119T183000_20210119T184500.part.tmp
/tmp/my_query_20210119T184500_20210119T190000.part.tmp
/tmp/my_query_20210119T190000_20210119T191500.part.tmp
/tmp/my_query_20210119T191500_20210119T193000.part.tmp
/tmp/my_query_20210119T193000_20210119T194500.part
/tmp/my_query_20210119T194500_20210119T200000.part
```

```
    If you do not specify the --merge-parts flag, the part files will be
    downloaded, and logcli will exit, and you can process the files as you wish.
    With the flag specified, the part files will be read in order, and the
    output printed to the terminal. The lines will be printed as soon as the
    next part is complete, you don't have to wait for all the parts to download
    before getting output. The --merge-parts flag will remove the part files
    when it is done reading each of them. To change this, you can use the
    --keep-parts flag, and the part files will not be removed.

instant-query [<flags>] <query>
    Run an instant LogQL query.

    The "instant-query" command is useful for evaluating a metric query for a
    single point in time. This is equivalent to the Grafana Explore table view;
    if you want a metrics query that is used to build a Grafana graph, you
    should use the "query" command instead.

    This command does not produce useful output when querying for log lines;
    you should always use the "query" command when you are running log queries.

    For more information about log queries and metric queries, refer to the
    LogQL documentation:

    https://grafana.com/docs/loki/latest/logql/

labels [<flags>] [<label>]
    Find values for a given label.

series [<flags>] <matcher>
    Run series query.

    The "series" command will take the provided label matcher and return all the
    log streams found in the time window.

    It is possible to send an empty label matcher '{}' to return all streams.

    Use the --analyze-labels flag to get a summary of the labels found in all
    streams. This is helpful to find high cardinality labels.
```

```
fmt
    Formats a LogQL query.

stats [<flags>] <query>
    Run a stats query.

    The "stats" command will take the provided query and return statistics
    from the index on how much data is contained in the matching stream(s).
    This only works against Loki instances using the TSDB index format.

    By default we look over the last hour of data; use --since to modify or
    provide specific start and end times with --from and --to respectively.

    Notice that when using --from and --to then ensure to use RFC3339Nano time
    format, but without timezone at the end. The local timezone will be added
    automatically or if using --timezone flag.

    Example:

      logcli stats
         --timezone=UTC
         --from="2021-01-19T10:00:00Z"
         --to="2021-01-19T20:00:00Z"
         'my-query'

volume [<flags>] <query>
    Run a volume query.

    The "volume" command will take the provided label selector(s) and return
    aggregate volumes for series matching those volumes. This only works against
    Loki instances using the TSDB index format.

    By default we look over the last hour of data; use --since to modify or
    provide specific start and end times with --from and --to respectively.

    Notice that when using --from and --to then ensure to use RFC3339Nano time
    format, but without timezone at the end. The local timezone will be added
    automatically or if using --timezone flag.
```

```
    Example:

      logcli volume
         --timezone=UTC
         --from="2021-01-19T10:00:00Z"
         --to="2021-01-19T20:00:00Z"
         'my-query'

volume_range [<flags>] <query>
    Run a volume query and return timeseries data.

    The "volume_range" command will take the provided label selector(s) and
    return aggregate volumes for series matching those volumes, aggregated into
    buckets according to the step value. This only works against Loki instances
    using the TSDB index format.

    By default we look over the last hour of data; use --since to modify or
    provide specific start and end times with --from and --to respectively.

    Notice that when using --from and --to then ensure to use RFC3339Nano time
    format, but without timezone at the end. The local timezone will be added
    automatically or if using --timezone flag.

    Example:

      logcli volume_range
         --timezone=UTC
         --from="2021-01-19T10:00:00Z"
         --to="2021-01-19T20:00:00Z"
          --step=1h
         'my-query'

detected-fields [<flags>] <query> [<field>]
    Run a query for detected fields..

delete <command> [<args> ...]
    Manage log deletion requests.

    The "detected-fields" command will return information about fields detected
```

```
    using either the "logfmt" or "json" parser against the log lines returned by
    the provided query for the provided time range.

    The "detected-fields" command will output extra information about the query
    and its results, such as the API URL, set of common labels, and set of
    excluded labels. This extra information can be suppressed with the --quiet
    flag.

    By default we look over the last hour of data; use --since to modify or
    provide specific start and end times with --from and --to respectively.

    Notice that when using --from and --to then ensure to use RFC3339Nano time
    format, but without timezone at the end. The local timezone will be added
    automatically or if using --timezone flag.

    Example:

      logcli detected-fields
         --timezone=UTC
         --from="2021-01-19T10:00:00Z"
         --to="2021-01-19T20:00:00Z"
         --output=jsonl
         'my-query'

    The output is limited to 100 fields by default; use --field-limit to
    increase. The query is limited to processing 1000 lines per subquery;
    use --line-limit to increase.
```

`query` command reference

The output of `logcli help query`:

```
usage: logcli query [<flags>] <query>

Run a LogQL query.

The "query" command is useful for querying for logs. Logs can be returned in a
few output modes:
```

```
  raw: log line
  default: log timestamp + log labels + log line
  jsonl: JSON response from Loki API of log line

The output of the log can be specified with the "-o" flag, for example, "-o raw"
for the raw output format.

The "query" command will output extra information about the query and its
results, such as the API URL, set of common labels, and set of excluded labels.
This extra information can be suppressed with the --quiet flag.

By default we look over the last hour of data; use --since to modify or provide
specific start and end times with --from and --to respectively.

Notice that when using --from and --to then ensure to use RFC3339Nano time
format, but without timezone at the end. The local timezone will be added
automatically or if using --timezone flag. In default output mode the
--output-timestamp-format flag can be used to modify the output timestamp.

Example:

  logcli query
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
     --output=jsonl
     'my-query'

Example with --output-timestamp-format:

  logcli query
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
     --output-timestamp-format=rfc3339nano
     'my-query'

The output is limited to 30 entries by default; use --limit to increase.
```

```
While "query" does support metrics queries, its output contains multiple data
points between the start and end query time. This output is used to build
graphs, similar to what is seen in the Grafana Explore graph view. If you are
querying metrics and just want the most recent data point (like what is seen in
the Grafana Explore table view), then you should use the "instant-query" command
instead.

Parallelization:

You can download an unlimited number of logs in parallel, there are a few flags
which control this behaviour:

  --parallel-duration
  --parallel-max-workers
  --part-path-prefix
  --overwrite-completed-parts
  --merge-parts
  --keep-parts

Refer to the help for each flag for details about what each of them do.

Example:

  logcli query
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
     --output=jsonl
     --parallel-duration="15m"
     --parallel-max-workers="4"
     --part-path-prefix="/tmp/my_query"
     --merge-parts
     'my-query'

This example will create a queue of jobs to execute, each being 15 minutes in
duration. In this case, that means, for the 10-hour total duration, there will
be forty 15-minute jobs. The --limit flag is ignored.

It will start four workers, and they will each take a job to work on from the
```

```
queue until all the jobs have been completed.

Each job will save a "part" file to the location specified by the
--part-path-prefix. Different prefixes can be used to run multiple queries at
the same time. The timestamp of the start and end of the part is in the file
name. While the part is being downloaded, the filename will end in ".part",
when it is complete, the file will be renamed to remove this ".part" extension.
By default, if a completed part file is found, that part will not be downloaded
again. This can be overridden with the --overwrite-completed-parts flag.

Part file example using the previous command, adding --keep-parts so they are
not deleted:

Since we don't have the --forward flag, the parts will be downloaded in reverse.
Two of the workers have finished their jobs (last two files), and have picked up
the next jobs in the queue. Running ls, this is what we should expect to see.

$ ls -1 /tmp/my_query* /tmp/my_query_20210119T183000_20210119T184500.part.tmp
/tmp/my_query_20210119T184500_20210119T190000.part.tmp
/tmp/my_query_20210119T190000_20210119T191500.part.tmp
/tmp/my_query_20210119T191500_20210119T193000.part.tmp
/tmp/my_query_20210119T193000_20210119T194500.part
/tmp/my_query_20210119T194500_20210119T200000.part

If you do not specify the --merge-parts flag, the part files will be downloaded,
and logcli will exit, and you can process the files as you wish. With the flag
specified, the part files will be read in order, and the output printed to
the terminal. The lines will be printed as soon as the next part is complete,
you don't have to wait for all the parts to download before getting output.
The --merge-parts flag will remove the part files when it is done reading each
of them. To change this, you can use the --keep-parts flag, and the part files
will not be removed.


Flags:
      --[no-]help               Show context-sensitive help (also try
                                --help-long and --help-man).
      --[no-]version            Show application version.
  -q, --[no-]quiet              Suppress query metadata
```

```
    --[no-]stats              Show query statistics
-o, --output=default          Specify output mode [default, raw, jsonl].
                              raw suppresses log labels and timestamp.
-z, --timezone=Local          Specify the timezone to use when formatting
                              output timestamps [Local, UTC]
    --output-timestamp-format=rfc3339
                              Specify the format of timestamps in the default
                              output mode [rfc3339, rfc3339nano, rfc822z,
                              rfc1123z, stampmicro, stampmilli, stampnano,
                              unixdate]
    --cpuprofile=""           Specify the location for writing a CPU profile.
    --memprofile=""           Specify the location for writing a memory
                              profile.
    --[no-]stdin              Take input logs from stdin
    --addr="http://localhost:3100"
                              Server address. Can also be set using LOKI_ADDR
                              env var. ($LOKI_ADDR)
    --username=""             Username for HTTP basic auth. Can also be set
                              using LOKI_USERNAME env var. ($LOKI_USERNAME)
    --password=""             Password for HTTP basic auth. Can also be set
                              using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
    --ca-cert=""              Path to the server Certificate Authority.
                              Can also be set using LOKI_CA_CERT_PATH env var.
                              ($LOKI_CA_CERT_PATH)
    --[no-]tls-skip-verify    Server certificate TLS skip verify. Can also
                              be set using LOKI_TLS_SKIP_VERIFY env var.
                              ($LOKI_TLS_SKIP_VERIFY)
    --cert=""                 Path to the client certificate. Can also
                              be set using LOKI_CLIENT_CERT_PATH env var.
                              ($LOKI_CLIENT_CERT_PATH)
    --key=""                  Path to the client certificate key. Can also
                              be set using LOKI_CLIENT_KEY_PATH env var.
                              ($LOKI_CLIENT_KEY_PATH)
    --org-id=""               adds X-Scope-OrgID to API requests for
                              representing tenant ID. Useful for requesting
                              tenant data when bypassing an auth gateway.
                              Can also be set using LOKI_ORG_ID env var.
                              ($LOKI_ORG_ID)
    --query-tags=""           adds X-Query-Tags http header to API requests.
```

```
                          This header value will be part of `metrics.go`
                          statistics. Useful for tracking the query.
                          Can also be set using LOKI_QUERY_TAGS env var.
                          ($LOKI_QUERY_TAGS)
    --[no-]nocache        adds Cache-Control: no-cache http header to API
                          requests. Can also be set using LOKI_NO_CACHE
                          env var. ($LOKI_NO_CACHE)
    --bearer-token=""     adds the Authorization header to API
                          requests for authentication purposes.
                          Can also be set using LOKI_BEARER_TOKEN env var.
                          ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""    adds the Authorization header to API requests
                          for authentication purposes. Can also be
                          set using LOKI_BEARER_TOKEN_FILE env var.
                          ($LOKI_BEARER_TOKEN_FILE)
    --retries=0           How many times to retry each query when
                          getting an error response from Loki. Can also
                          be set using LOKI_CLIENT_RETRIES env var.
                          ($LOKI_CLIENT_RETRIES)
    --min-backoff=0       Minimum backoff time between retries. Can also
                          be set using LOKI_CLIENT_MIN_BACKOFF env var.
                          ($LOKI_CLIENT_MIN_BACKOFF)
    --max-backoff=0       Maximum backoff time between retries. Can also
                          be set using LOKI_CLIENT_MAX_BACKOFF env var.
                          ($LOKI_CLIENT_MAX_BACKOFF)
    --auth-header="Authorization"
                          The authorization header used. Can also
                          be set using LOKI_AUTH_HEADER env var.
                          ($LOKI_AUTH_HEADER)
    --proxy-url=""        The http or https proxy to use when
                          making requests. Can also be set
                          using LOKI_HTTP_PROXY_URL env var.
                          ($LOKI_HTTP_PROXY_URL)
    --[no-]compress       Request that Loki compress returned
                          data in transit. Can also be set
                          using LOKI_HTTP_COMPRESSION env var.
                          ($LOKI_HTTP_COMPRESSION)
    --[no-]envproxy       Use ProxyFromEnvironment to use net/http
                          ProxyFromEnvironment configuration, eg
```

```
                          HTTP_PROXY ($LOKI_ENV_PROXY)
--limit=30                Limit on number of entries to print. Setting it
                          to 0 will fetch all entries.
--since=1h                Lookback window.
--from=FROM               Start looking for logs at this absolute time
                          (inclusive)
--to=TO                   Stop looking for logs at this absolute time
                          (exclusive)
--step=STEP               Query resolution step width, for metric queries.
                          Evaluate the query at the specified step over
                          the time range.
--interval=INTERVAL       Query interval, for log queries. Return entries
                          at the specified interval, ignoring those
                          between. **This parameter is experimental,
                          please see Issue 1779**
--batch=1000              Query batch size to use until 'limit' is reached
--parallel-duration=1h    Split the range into jobs of this length to
                          download the logs in parallel. This will
                          result in the logs being out of order.
                          Use --part-path-prefix to create a file per job
                          to maintain ordering.
--parallel-max-workers=1  Max number of workers to start up for parallel
                          jobs. A value of 1 will not create any parallel
                          workers. When using parallel workers, limit is
                          ignored.
--part-path-prefix=PART-PATH-PREFIX
                          When set, each server response will be saved
                          to a file with this prefix. Creates files in
                          the format: 'prefix-utc_start-utc_end.part'.
                          Intended to be used with the parallel-* flags
                          so that you can combine the files to maintain
                          ordering based on the filename. Default is to
                          write to stdout.
--[no-]overwrite-completed-parts
                          Overwrites completed part files. This will
                          download the range again, and replace the
                          original completed part file. Default will skip
                          a range if it's part file is already downloaded.
--[no-]merge-parts        Reads the part files in order and writes the
```

```
                              output to stdout. Original part files will be
                              deleted with this option.
      --[no-]keep-parts       Overrides the default behaviour of --merge-parts
                              which will delete the part files once all the
                              files have been read. This option will keep the
                              part files.
      --[no-]forward          Scan forwards through logs.
      --[no-]no-labels        Do not print any labels
      --exclude-label=EXCLUDE-LABEL ...
                              Exclude labels given the provided key during
                              output.
      --include-label=INCLUDE-LABEL ...
                              Include labels given the provided key during
                              output.
      --[no-]include-common-labels
                              Include common labels in output for each log
                              line.
      --labels-length=0       Set a fixed padding to labels
      --store-config=""       Execute the current query using a configured
                              storage from a given Loki configuration file.
      --[no-]remote-schema    Execute the current query using a remote schema
                              retrieved from the configured -schema-store.
      --schema-store=""       Store used for retrieving remote schema.
      --[no-]colored-output   Show output with colored labels
  -t, --[no-]tail             Tail the logs
  -f, --[no-]follow           Alias for --tail
      --delay-for=0           Delay in tailing by number of seconds to
                              accumulate logs for re-ordering

Args:
  <query>  eg '{foo="bar",baz=~".*blip"} |~ ".*error.*"'
```

`instant-query` command reference

The output of `logcli help instant-query`:

```
usage: logcli instant-query [<flags>] <query>

Run an instant LogQL query.
```

```
The "instant-query" command is useful for evaluating a metric query for a
single point in time. This is equivalent to the Grafana Explore table view;
if you want a metrics query that is used to build a Grafana graph, you should
use the "query" command instead.

This command does not produce useful output when querying for log lines;
you should always use the "query" command when you are running log queries.

For more information about log queries and metric queries, refer to the LogQL
documentation:

https://grafana.com/docs/loki/latest/logql/


Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
                              and --help-man).
      --[no-]version          Show application version.
  -q, --[no-]quiet            Suppress query metadata
      --[no-]stats            Show query statistics
  -o, --output=default        Specify output mode [default, raw, jsonl].
                              raw suppresses log labels and timestamp.
  -z, --timezone=Local        Specify the timezone to use when formatting output
                              timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                              Specify the format of timestamps in the default
                              output mode [rfc3339, rfc3339nano, rfc822z,
                              rfc1123z, stampmicro, stampmilli, stampnano,
                              unixdate]
      --cpuprofile=""         Specify the location for writing a CPU profile.
      --memprofile=""         Specify the location for writing a memory profile.
      --[no-]stdin            Take input logs from stdin
      --addr="http://localhost:3100"
                              Server address. Can also be set using LOKI_ADDR
                              env var. ($LOKI_ADDR)
      --username=""           Username for HTTP basic auth. Can also be set
                              using LOKI_USERNAME env var. ($LOKI_USERNAME)
      --password=""           Password for HTTP basic auth. Can also be set
```

```
                        using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
--ca-cert=""            Path to the server Certificate Authority.
                        Can also be set using LOKI_CA_CERT_PATH env var.
                        ($LOKI_CA_CERT_PATH)
--[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
                        be set using LOKI_TLS_SKIP_VERIFY env var.
                        ($LOKI_TLS_SKIP_VERIFY)
--cert=""               Path to the client certificate. Can also
                        be set using LOKI_CLIENT_CERT_PATH env var.
                        ($LOKI_CLIENT_CERT_PATH)
--key=""                Path to the client certificate key. Can also
                        be set using LOKI_CLIENT_KEY_PATH env var.
                        ($LOKI_CLIENT_KEY_PATH)
--org-id=""             adds X-Scope-OrgID to API requests for
                        representing tenant ID. Useful for requesting
                        tenant data when bypassing an auth gateway.
                        Can also be set using LOKI_ORG_ID env var.
                        ($LOKI_ORG_ID)
--query-tags=""         adds X-Query-Tags http header to API requests.
                        This header value will be part of `metrics.go`
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
--[no-]nocache          adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
--bearer-token=""       adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
--bearer-token-file=""  adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
--retries=0             How many times to retry each query when
                        getting an error response from Loki. Can also
                        be set using LOKI_CLIENT_RETRIES env var.
                        ($LOKI_CLIENT_RETRIES)
--min-backoff=0         Minimum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MIN_BACKOFF env var.
```

```
                        ($LOKI_CLIENT_MIN_BACKOFF)
--max-backoff=0         Maximum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MAX_BACKOFF env var.
                        ($LOKI_CLIENT_MAX_BACKOFF)
--auth-header="Authorization"
                        The authorization header used. Can also
                        be set using LOKI_AUTH_HEADER env var.
                        ($LOKI_AUTH_HEADER)
--proxy-url=""          The http or https proxy to use when
                        making requests. Can also be set
                        using LOKI_HTTP_PROXY_URL env var.
                        ($LOKI_HTTP_PROXY_URL)
--[no-]compress         Request that Loki compress returned
                        data in transit. Can also be set
                        using LOKI_HTTP_COMPRESSION env var.
                        ($LOKI_HTTP_COMPRESSION)
--[no-]envproxy         Use ProxyFromEnvironment to use net/http
                        ProxyFromEnvironment configuration, eg HTTP_PROXY
                        ($LOKI_ENV_PROXY)
--limit=30              Limit on number of entries to print. Setting it to
                        0 will fetch all entries.
--now=NOW               Time at which to execute the instant query.
--[no-]forward          Scan forwards through logs.
--[no-]no-labels        Do not print any labels
--exclude-label=EXCLUDE-LABEL ...
                        Exclude labels given the provided key during
                        output.
--include-label=INCLUDE-LABEL ...
                        Include labels given the provided key during
                        output.
--[no-]include-common-labels
                        Include common labels in output for each log line.
--labels-length=0       Set a fixed padding to labels
--store-config=""       Execute the current query using a configured
                        storage from a given Loki configuration file.
--[no-]remote-schema    Execute the current query using a remote schema
                        retrieved from the configured -schema-store.
--schema-store=""       Store used for retrieving remote schema.
--[no-]colored-output   Show output with colored labels
```

```
Args:
  <query>  eg 'rate({foo="bar"} |~ ".*error.*" [5m])'
```

`labels` command reference

The output of `logcli help labels`:

```
usage: logcli labels [<flags>] [<label>]

Find values for a given label.


Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
                              and --help-man).
      --[no-]version          Show application version.
  -q, --[no-]quiet            Suppress query metadata
      --[no-]stats            Show query statistics
  -o, --output=default        Specify output mode [default, raw, jsonl].
                              raw suppresses log labels and timestamp.
  -z, --timezone=Local        Specify the timezone to use when formatting output
                              timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                              Specify the format of timestamps in the default
                              output mode [rfc3339, rfc3339nano, rfc822z,
                              rfc1123z, stampmicro, stampmilli, stampnano,
                              unixdate]
      --cpuprofile=""         Specify the location for writing a CPU profile.
      --memprofile=""         Specify the location for writing a memory profile.
      --[no-]stdin            Take input logs from stdin
      --addr="http://localhost:3100"
                              Server address. Can also be set using LOKI_ADDR
                              env var. ($LOKI_ADDR)
      --username=""           Username for HTTP basic auth. Can also be set
                              using LOKI_USERNAME env var. ($LOKI_USERNAME)
      --password=""           Password for HTTP basic auth. Can also be set
                              using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
      --ca-cert=""            Path to the server Certificate Authority.
```

```
                        Can also be set using LOKI_CA_CERT_PATH env var.
                        ($LOKI_CA_CERT_PATH)
    --[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
                        be set using LOKI_TLS_SKIP_VERIFY env var.
                        ($LOKI_TLS_SKIP_VERIFY)
    --cert=""           Path to the client certificate. Can also
                        be set using LOKI_CLIENT_CERT_PATH env var.
                        ($LOKI_CLIENT_CERT_PATH)
    --key=""            Path to the client certificate key. Can also
                        be set using LOKI_CLIENT_KEY_PATH env var.
                        ($LOKI_CLIENT_KEY_PATH)
    --org-id=""         adds X-Scope-OrgID to API requests for
                        representing tenant ID. Useful for requesting
                        tenant data when bypassing an auth gateway.
                        Can also be set using LOKI_ORG_ID env var.
                        ($LOKI_ORG_ID)
    --query-tags=""     adds X-Query-Tags http header to API requests.
                        This header value will be part of `metrics.go`
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
    --[no-]nocache      adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
    --bearer-token=""   adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""  adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
    --retries=0         How many times to retry each query when
                        getting an error response from Loki. Can also
                        be set using LOKI_CLIENT_RETRIES env var.
                        ($LOKI_CLIENT_RETRIES)
    --min-backoff=0     Minimum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MIN_BACKOFF env var.
                        ($LOKI_CLIENT_MIN_BACKOFF)
    --max-backoff=0     Maximum backoff time between retries. Can also
```

```
                           be set using LOKI_CLIENT_MAX_BACKOFF env var.
                           ($LOKI_CLIENT_MAX_BACKOFF)
      --auth-header="Authorization"
                           The authorization header used. Can also
                           be set using LOKI_AUTH_HEADER env var.
                           ($LOKI_AUTH_HEADER)
      --proxy-url=""       The http or https proxy to use when
                           making requests. Can also be set
                           using LOKI_HTTP_PROXY_URL env var.
                           ($LOKI_HTTP_PROXY_URL)
      --[no-]compress      Request that Loki compress returned
                           data in transit. Can also be set
                           using LOKI_HTTP_COMPRESSION env var.
                           ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy      Use ProxyFromEnvironment to use net/http
                           ProxyFromEnvironment configuration, eg HTTP_PROXY
                           ($LOKI_ENV_PROXY)
      --since=1h           Lookback window.
      --from=FROM          Start looking for labels at this absolute time
                           (inclusive)
      --to=TO              Stop looking for labels at this absolute time
                           (exclusive)

Args:
  [<label>]  The name of the label.
```

`series` command reference

The output of `logcli help series`:

```
usage: logcli series [<flags>] <matcher>

Run series query.

The "series" command will take the provided label matcher and return all the log
streams found in the time window.

It is possible to send an empty label matcher '{}' to return all streams.
```

```
Use the --analyze-labels flag to get a summary of the labels found in all
streams. This is helpful to find high cardinality labels.


Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
                              and --help-man).
      --[no-]version          Show application version.
  -q, --[no-]quiet            Suppress query metadata
      --[no-]stats            Show query statistics
  -o, --output=default        Specify output mode [default, raw, jsonl].
                              raw suppresses log labels and timestamp.
  -z, --timezone=Local        Specify the timezone to use when formatting output
                              timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                              Specify the format of timestamps in the default
                              output mode [rfc3339, rfc3339nano, rfc822z,
                              rfc1123z, stampmicro, stampmilli, stampnano,
                              unixdate]
      --cpuprofile=""         Specify the location for writing a CPU profile.
      --memprofile=""         Specify the location for writing a memory profile.
      --[no-]stdin            Take input logs from stdin
      --addr="http://localhost:3100"
                              Server address. Can also be set using LOKI_ADDR
                              env var. ($LOKI_ADDR)
      --username=""           Username for HTTP basic auth. Can also be set
                              using LOKI_USERNAME env var. ($LOKI_USERNAME)
      --password=""           Password for HTTP basic auth. Can also be set
                              using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
      --ca-cert=""            Path to the server Certificate Authority.
                              Can also be set using LOKI_CA_CERT_PATH env var.
                              ($LOKI_CA_CERT_PATH)
      --[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
                              be set using LOKI_TLS_SKIP_VERIFY env var.
                              ($LOKI_TLS_SKIP_VERIFY)
      --cert=""               Path to the client certificate. Can also
                              be set using LOKI_CLIENT_CERT_PATH env var.
                              ($LOKI_CLIENT_CERT_PATH)
      --key=""                Path to the client certificate key. Can also
```

```
                        be set using LOKI_CLIENT_KEY_PATH env var.
                        ($LOKI_CLIENT_KEY_PATH)
    --org-id=""         adds X-Scope-OrgID to API requests for
                        representing tenant ID. Useful for requesting
                        tenant data when bypassing an auth gateway.
                        Can also be set using LOKI_ORG_ID env var.
                        ($LOKI_ORG_ID)
    --query-tags=""     adds X-Query-Tags http header to API requests.
                        This header value will be part of `metrics.go`
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
    --[no-]nocache      adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
    --bearer-token=""   adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""  adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
    --retries=0         How many times to retry each query when
                        getting an error response from Loki. Can also
                        be set using LOKI_CLIENT_RETRIES env var.
                        ($LOKI_CLIENT_RETRIES)
    --min-backoff=0     Minimum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MIN_BACKOFF env var.
                        ($LOKI_CLIENT_MIN_BACKOFF)
    --max-backoff=0     Maximum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MAX_BACKOFF env var.
                        ($LOKI_CLIENT_MAX_BACKOFF)
    --auth-header="Authorization"
                        The authorization header used. Can also
                        be set using LOKI_AUTH_HEADER env var.
                        ($LOKI_AUTH_HEADER)
    --proxy-url=""      The http or https proxy to use when
                        making requests. Can also be set
                        using LOKI_HTTP_PROXY_URL env var.
```

```
                           ($LOKI_HTTP_PROXY_URL)
      --[no-]compress      Request that Loki compress returned
                           data in transit. Can also be set
                           using LOKI_HTTP_COMPRESSION env var.
                           ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy      Use ProxyFromEnvironment to use net/http
                           ProxyFromEnvironment configuration, eg HTTP_PROXY
                           ($LOKI_ENV_PROXY)
      --since=1h           Lookback window.
      --from=FROM          Start looking for logs at this absolute time
                           (inclusive)
      --to=TO              Stop looking for logs at this absolute time
                           (exclusive)
      --[no-]analyze-labels   Printout a summary of labels including count of
                           label value combinations, useful for debugging
                           high cardinality series

Args:
  <matcher>  eg '{foo="bar",baz=~".*blip"}'
```

`stats` command reference

The output of `logcli help stats`:

```
usage: logcli stats [<flags>] <query>

Run a stats query.

The "stats" command will take the provided query and return statistics from the
index on how much data is contained in the matching stream(s). This only works
against Loki instances using the TSDB index format.

By default we look over the last hour of data; use --since to modify or provide
specific start and end times with --from and --to respectively.

Notice that when using --from and --to then ensure to use RFC3339Nano time
format, but without timezone at the end. The local timezone will be added
automatically or if using --timezone flag.
```

```
Example:

  logcli stats
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
     'my-query'


Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
                              and --help-man).
      --[no-]version          Show application version.
  -q, --[no-]quiet            Suppress query metadata
      --[no-]stats            Show query statistics
  -o, --output=default        Specify output mode [default, raw, jsonl].
                              raw suppresses log labels and timestamp.
  -z, --timezone=Local        Specify the timezone to use when formatting output
                              timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                              Specify the format of timestamps in the default
                              output mode [rfc3339, rfc3339nano, rfc822z,
                              rfc1123z, stampmicro, stampmilli, stampnano,
                              unixdate]
      --cpuprofile=""         Specify the location for writing a CPU profile.
      --memprofile=""         Specify the location for writing a memory profile.
      --[no-]stdin            Take input logs from stdin
      --addr="http://localhost:3100"
                              Server address. Can also be set using LOKI_ADDR
                              env var. ($LOKI_ADDR)
      --username=""           Username for HTTP basic auth. Can also be set
                              using LOKI_USERNAME env var. ($LOKI_USERNAME)
      --password=""           Password for HTTP basic auth. Can also be set
                              using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
      --ca-cert=""            Path to the server Certificate Authority.
                              Can also be set using LOKI_CA_CERT_PATH env var.
                              ($LOKI_CA_CERT_PATH)
      --[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
```

```
                        be set using LOKI_TLS_SKIP_VERIFY env var.
                        ($LOKI_TLS_SKIP_VERIFY)
    --cert=""           Path to the client certificate. Can also
                        be set using LOKI_CLIENT_CERT_PATH env var.
                        ($LOKI_CLIENT_CERT_PATH)
    --key=""            Path to the client certificate key. Can also
                        be set using LOKI_CLIENT_KEY_PATH env var.
                        ($LOKI_CLIENT_KEY_PATH)
    --org-id=""         adds X-Scope-OrgID to API requests for
                        representing tenant ID. Useful for requesting
                        tenant data when bypassing an auth gateway.
                        Can also be set using LOKI_ORG_ID env var.
                        ($LOKI_ORG_ID)
    --query-tags=""     adds X-Query-Tags http header to API requests.
                        This header value will be part of `metrics.go`
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
    --[no-]nocache      adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
    --bearer-token=""   adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""  adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
    --retries=0         How many times to retry each query when
                        getting an error response from Loki. Can also
                        be set using LOKI_CLIENT_RETRIES env var.
                        ($LOKI_CLIENT_RETRIES)
    --min-backoff=0     Minimum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MIN_BACKOFF env var.
                        ($LOKI_CLIENT_MIN_BACKOFF)
    --max-backoff=0     Maximum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MAX_BACKOFF env var.
                        ($LOKI_CLIENT_MAX_BACKOFF)
    --auth-header="Authorization"
```

```
                           The authorization header used. Can also
                           be set using LOKI_AUTH_HEADER env var.
                           ($LOKI_AUTH_HEADER)
      --proxy-url=""       The http or https proxy to use when
                           making requests. Can also be set
                           using LOKI_HTTP_PROXY_URL env var.
                           ($LOKI_HTTP_PROXY_URL)
      --[no-]compress      Request that Loki compress returned
                           data in transit. Can also be set
                           using LOKI_HTTP_COMPRESSION env var.
                           ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy      Use ProxyFromEnvironment to use net/http
                           ProxyFromEnvironment configuration, eg HTTP_PROXY
                           ($LOKI_ENV_PROXY)
      --since=1h           Lookback window.
      --from=FROM          Start looking for logs at this absolute time
                           (inclusive)
      --to=TO              Stop looking for logs at this absolute time
                           (exclusive)

Args:
  <query>  eg '{foo="bar",baz=~".*blip"} |~ ".*error.*"'
```

`volume` command reference

The output of `logcli help volume`:

```
usage: logcli volume [<flags>] <query>

Run a volume query.

The "volume" command will take the provided label selector(s) and return
aggregate volumes for series matching those volumes. This only works against
Loki instances using the TSDB index format.

By default we look over the last hour of data; use --since to modify or provide
specific start and end times with --from and --to respectively.

Notice that when using --from and --to then ensure to use RFC3339Nano time
```

```
format, but without timezone at the end. The local timezone will be added
automatically or if using --timezone flag.

Example:

  logcli volume
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
     'my-query'


Flags:
      --[no-]help               Show context-sensitive help (also try
                                --help-long and --help-man).
      --[no-]version            Show application version.
  -q, --[no-]quiet              Suppress query metadata
      --[no-]stats              Show query statistics
  -o, --output=default          Specify output mode [default, raw, jsonl].
                                raw suppresses log labels and timestamp.
  -z, --timezone=Local          Specify the timezone to use when formatting
                                output timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                                Specify the format of timestamps in the default
                                output mode [rfc3339, rfc3339nano, rfc822z,
                                rfc1123z, stampmicro, stampmilli, stampnano,
                                unixdate]
      --cpuprofile=""           Specify the location for writing a CPU profile.
      --memprofile=""           Specify the location for writing a memory
                                profile.
      --[no-]stdin              Take input logs from stdin
      --addr="http://localhost:3100"
                                Server address. Can also be set using LOKI_ADDR
                                env var. ($LOKI_ADDR)
      --username=""             Username for HTTP basic auth. Can also be set
                                using LOKI_USERNAME env var. ($LOKI_USERNAME)
      --password=""             Password for HTTP basic auth. Can also be set
                                using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
      --ca-cert=""              Path to the server Certificate Authority.
```

```
                          Can also be set using LOKI_CA_CERT_PATH env var.
                          ($LOKI_CA_CERT_PATH)
    --[no-]tls-skip-verify    Server certificate TLS skip verify. Can also
                          be set using LOKI_TLS_SKIP_VERIFY env var.
                          ($LOKI_TLS_SKIP_VERIFY)
    --cert=""             Path to the client certificate. Can also
                          be set using LOKI_CLIENT_CERT_PATH env var.
                          ($LOKI_CLIENT_CERT_PATH)
    --key=""              Path to the client certificate key. Can also
                          be set using LOKI_CLIENT_KEY_PATH env var.
                          ($LOKI_CLIENT_KEY_PATH)
    --org-id=""           adds X-Scope-OrgID to API requests for
                          representing tenant ID. Useful for requesting
                          tenant data when bypassing an auth gateway.
                          Can also be set using LOKI_ORG_ID env var.
                          ($LOKI_ORG_ID)
    --query-tags=""       adds X-Query-Tags http header to API requests.
                          This header value will be part of `metrics.go`
                          statistics. Useful for tracking the query.
                          Can also be set using LOKI_QUERY_TAGS env var.
                          ($LOKI_QUERY_TAGS)
    --[no-]nocache        adds Cache-Control: no-cache http header to API
                          requests. Can also be set using LOKI_NO_CACHE
                          env var. ($LOKI_NO_CACHE)
    --bearer-token=""     adds the Authorization header to API
                          requests for authentication purposes.
                          Can also be set using LOKI_BEARER_TOKEN env var.
                          ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""    adds the Authorization header to API requests
                          for authentication purposes. Can also be
                          set using LOKI_BEARER_TOKEN_FILE env var.
                          ($LOKI_BEARER_TOKEN_FILE)
    --retries=0           How many times to retry each query when
                          getting an error response from Loki. Can also
                          be set using LOKI_CLIENT_RETRIES env var.
                          ($LOKI_CLIENT_RETRIES)
    --min-backoff=0       Minimum backoff time between retries. Can also
                          be set using LOKI_CLIENT_MIN_BACKOFF env var.
                          ($LOKI_CLIENT_MIN_BACKOFF)
```

```
      --max-backoff=0           Maximum backoff time between retries. Can also
                                be set using LOKI_CLIENT_MAX_BACKOFF env var.
                                ($LOKI_CLIENT_MAX_BACKOFF)
      --auth-header="Authorization"
                                The authorization header used. Can also
                                be set using LOKI_AUTH_HEADER env var.
                                ($LOKI_AUTH_HEADER)
      --proxy-url=""            The http or https proxy to use when
                                making requests. Can also be set
                                using LOKI_HTTP_PROXY_URL env var.
                                ($LOKI_HTTP_PROXY_URL)
      --[no-]compress           Request that Loki compress returned
                                data in transit. Can also be set
                                using LOKI_HTTP_COMPRESSION env var.
                                ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy           Use ProxyFromEnvironment to use net/http
                                ProxyFromEnvironment configuration, eg
                                HTTP_PROXY ($LOKI_ENV_PROXY)
      --since=1h                Lookback window.
      --from=FROM               Start looking for logs at this absolute time
                                (inclusive)
      --to=TO                   Stop looking for logs at this absolute time
                                (exclusive)
      --limit=30                Limit on number of series to return volumes for.
      --targetLabels=TARGETLABELS ...
                                List of labels to aggregate results into.
      --[no-]aggregateByLabels  Whether to aggregate results by label name only.

Args:
  <query>  eg '{foo="bar",baz=~".*blip"}
```

`volume_range` command reference

The output of `logcli help volume_range`:

```
usage: logcli volume_range [<flags>] <query>

Run a volume query and return timeseries data.
```

```
The "volume_range" command will take the provided label selector(s) and return
aggregate volumes for series matching those volumes, aggregated into buckets
according to the step value. This only works against Loki instances using the
TSDB index format.

By default we look over the last hour of data; use --since to modify or provide
specific start and end times with --from and --to respectively.

Notice that when using --from and --to then ensure to use RFC3339Nano time
format, but without timezone at the end. The local timezone will be added
automatically or if using --timezone flag.

Example:

  logcli volume_range
    --timezone=UTC
    --from="2021-01-19T10:00:00Z"
    --to="2021-01-19T20:00:00Z"
     --step=1h
    'my-query'


Flags:
      --[no-]help               Show context-sensitive help (also try
                                --help-long and --help-man).
      --[no-]version            Show application version.
  -q, --[no-]quiet              Suppress query metadata
      --[no-]stats              Show query statistics
  -o, --output=default          Specify output mode [default, raw, jsonl].
                                raw suppresses log labels and timestamp.
  -z, --timezone=Local          Specify the timezone to use when formatting
                                output timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                                Specify the format of timestamps in the default
                                output mode [rfc3339, rfc3339nano, rfc822z,
                                rfc1123z, stampmicro, stampmilli, stampnano,
                                unixdate]
      --cpuprofile=""           Specify the location for writing a CPU profile.
      --memprofile=""           Specify the location for writing a memory
```

```
                          profile.
--[no-]stdin              Take input logs from stdin
--addr="http://localhost:3100"
                          Server address. Can also be set using LOKI_ADDR
                          env var. ($LOKI_ADDR)
--username=""             Username for HTTP basic auth. Can also be set
                          using LOKI_USERNAME env var. ($LOKI_USERNAME)
--password=""             Password for HTTP basic auth. Can also be set
                          using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
--ca-cert=""              Path to the server Certificate Authority.
                          Can also be set using LOKI_CA_CERT_PATH env var.
                          ($LOKI_CA_CERT_PATH)
--[no-]tls-skip-verify    Server certificate TLS skip verify. Can also
                          be set using LOKI_TLS_SKIP_VERIFY env var.
                          ($LOKI_TLS_SKIP_VERIFY)
--cert=""                 Path to the client certificate. Can also
                          be set using LOKI_CLIENT_CERT_PATH env var.
                          ($LOKI_CLIENT_CERT_PATH)
--key=""                  Path to the client certificate key. Can also
                          be set using LOKI_CLIENT_KEY_PATH env var.
                          ($LOKI_CLIENT_KEY_PATH)
--org-id=""               adds X-Scope-OrgID to API requests for
                          representing tenant ID. Useful for requesting
                          tenant data when bypassing an auth gateway.
                          Can also be set using LOKI_ORG_ID env var.
                          ($LOKI_ORG_ID)
--query-tags=""           adds X-Query-Tags http header to API requests.
                          This header value will be part of `metrics.go`
                          statistics. Useful for tracking the query.
                          Can also be set using LOKI_QUERY_TAGS env var.
                          ($LOKI_QUERY_TAGS)
--[no-]nocache            adds Cache-Control: no-cache http header to API
                          requests. Can also be set using LOKI_NO_CACHE
                          env var. ($LOKI_NO_CACHE)
--bearer-token=""         adds the Authorization header to API
                          requests for authentication purposes.
                          Can also be set using LOKI_BEARER_TOKEN env var.
                          ($LOKI_BEARER_TOKEN)
--bearer-token-file=""    adds the Authorization header to API requests
```

```
                          for authentication purposes. Can also be
                          set using LOKI_BEARER_TOKEN_FILE env var.
                          ($LOKI_BEARER_TOKEN_FILE)
  --retries=0             How many times to retry each query when
                          getting an error response from Loki. Can also
                          be set using LOKI_CLIENT_RETRIES env var.
                          ($LOKI_CLIENT_RETRIES)
  --min-backoff=0         Minimum backoff time between retries. Can also
                          be set using LOKI_CLIENT_MIN_BACKOFF env var.
                          ($LOKI_CLIENT_MIN_BACKOFF)
  --max-backoff=0         Maximum backoff time between retries. Can also
                          be set using LOKI_CLIENT_MAX_BACKOFF env var.
                          ($LOKI_CLIENT_MAX_BACKOFF)
  --auth-header="Authorization"
                          The authorization header used. Can also
                          be set using LOKI_AUTH_HEADER env var.
                          ($LOKI_AUTH_HEADER)
  --proxy-url=""          The http or https proxy to use when
                          making requests. Can also be set
                          using LOKI_HTTP_PROXY_URL env var.
                          ($LOKI_HTTP_PROXY_URL)
  --[no-]compress         Request that Loki compress returned
                          data in transit. Can also be set
                          using LOKI_HTTP_COMPRESSION env var.
                          ($LOKI_HTTP_COMPRESSION)
  --[no-]envproxy         Use ProxyFromEnvironment to use net/http
                          ProxyFromEnvironment configuration, eg
                          HTTP_PROXY ($LOKI_ENV_PROXY)
  --since=1h              Lookback window.
  --from=FROM             Start looking for logs at this absolute time
                          (inclusive)
  --to=TO                 Stop looking for logs at this absolute time
                          (exclusive)
  --limit=30              Limit on number of series to return volumes for.
  --targetLabels=TARGETLABELS ...
                          List of labels to aggregate results into.
  --[no-]aggregateByLabels  Whether to aggregate results by label name only.
  --step=1h               Query resolution step width, roll up volumes
                          into buckets cover step time each.
```

```
Args:
  <query>  eg '{foo="bar",baz=~".*blip"}
```

`detected-fields` command reference

The output of `logcli help detected-fields`:

```
usage: logcli detected-fields [<flags>] <query> [<field>]

Run a query for detected fields..

The "detected-fields" command will return information about fields detected
using either the "logfmt" or "json" parser against the log lines returned by the
provided query for the provided time range.

The "detected-fields" command will output extra information about the query
and its results, such as the API URL, set of common labels, and set of excluded
labels. This extra information can be suppressed with the --quiet flag.

By default we look over the last hour of data; use --since to modify or provide
specific start and end times with --from and --to respectively.

Notice that when using --from and --to then ensure to use RFC3339Nano time
format, but without timezone at the end. The local timezone will be added
automatically or if using --timezone flag.

Example:

  logcli detected-fields
     --timezone=UTC
     --from="2021-01-19T10:00:00Z"
     --to="2021-01-19T20:00:00Z"
     --output=jsonl
     'my-query'

The output is limited to 100 fields by default; use --field-limit to increase.
The query is limited to processing 1000 lines per subquery; use --line-limit to
increase.
```

```
Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
                              and --help-man).
      --[no-]version          Show application version.
  -q, --[no-]quiet            Suppress query metadata
      --[no-]stats            Show query statistics
  -o, --output=default        Specify output mode [default, raw, jsonl].
                              raw suppresses log labels and timestamp.
  -z, --timezone=Local        Specify the timezone to use when formatting output
                              timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                              Specify the format of timestamps in the default
                              output mode [rfc3339, rfc3339nano, rfc822z,
                              rfc1123z, stampmicro, stampmilli, stampnano,
                              unixdate]
      --cpuprofile=""         Specify the location for writing a CPU profile.
      --memprofile=""         Specify the location for writing a memory profile.
      --[no-]stdin            Take input logs from stdin
      --addr="http://localhost:3100"
                              Server address. Can also be set using LOKI_ADDR
                              env var. ($LOKI_ADDR)
      --username=""           Username for HTTP basic auth. Can also be set
                              using LOKI_USERNAME env var. ($LOKI_USERNAME)
      --password=""           Password for HTTP basic auth. Can also be set
                              using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
      --ca-cert=""            Path to the server Certificate Authority.
                              Can also be set using LOKI_CA_CERT_PATH env var.
                              ($LOKI_CA_CERT_PATH)
      --[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
                              be set using LOKI_TLS_SKIP_VERIFY env var.
                              ($LOKI_TLS_SKIP_VERIFY)
      --cert=""               Path to the client certificate. Can also
                              be set using LOKI_CLIENT_CERT_PATH env var.
                              ($LOKI_CLIENT_CERT_PATH)
      --key=""                Path to the client certificate key. Can also
                              be set using LOKI_CLIENT_KEY_PATH env var.
                              ($LOKI_CLIENT_KEY_PATH)
```

```
--org-id=""             adds X-Scope-OrgID to API requests for
                        representing tenant ID. Useful for requesting
                        tenant data when bypassing an auth gateway.
                        Can also be set using LOKI_ORG_ID env var.
                        ($LOKI_ORG_ID)
--query-tags=""         adds X-Query-Tags http header to API requests.
                        This header value will be part of `metrics.go`
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
--[no-]nocache          adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
--bearer-token=""       adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
--bearer-token-file=""  adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
--retries=0             How many times to retry each query when
                        getting an error response from Loki. Can also
                        be set using LOKI_CLIENT_RETRIES env var.
                        ($LOKI_CLIENT_RETRIES)
--min-backoff=0         Minimum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MIN_BACKOFF env var.
                        ($LOKI_CLIENT_MIN_BACKOFF)
--max-backoff=0         Maximum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MAX_BACKOFF env var.
                        ($LOKI_CLIENT_MAX_BACKOFF)
--auth-header="Authorization"
                        The authorization header used. Can also
                        be set using LOKI_AUTH_HEADER env var.
                        ($LOKI_AUTH_HEADER)
--proxy-url=""          The http or https proxy to use when
                        making requests. Can also be set
                        using LOKI_HTTP_PROXY_URL env var.
                        ($LOKI_HTTP_PROXY_URL)
--[no-]compress         Request that Loki compress returned
```

```
                           data in transit. Can also be set
                           using LOKI_HTTP_COMPRESSION env var.
                           ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy      Use ProxyFromEnvironment to use net/http
                           ProxyFromEnvironment configuration, eg HTTP_PROXY
                           ($LOKI_ENV_PROXY)
      --limit=100          Limit on number of fields or values to return.
      --line-limit=1000    Limit the number of lines each subquery is allowed
                           to process.
      --since=1h           Lookback window.
      --from=FROM          Start looking for logs at this absolute time
                           (inclusive)
      --to=TO              Stop looking for logs at this absolute time
                           (exclusive)
      --step=10s           Query resolution step width, for metric queries.
                           Evaluate the query at the specified step over the
                           time range.

Args:
  <query>    eg '{foo="bar",baz=~".*blip"} |~ ".*error.*"'
  [<field>]  The name of the field.
```

`delete` command reference

The output of `logcli help delete`:

```
usage: logcli delete <command> [<args> ...]

Manage log deletion requests.


Flags:
      --[no-]help          Show context-sensitive help (also try --help-long
                           and --help-man).
      --[no-]version       Show application version.
  -q, --[no-]quiet         Suppress query metadata
      --[no-]stats         Show query statistics
  -o, --output=default     Specify output mode [default, raw, jsonl].
                           raw suppresses log labels and timestamp.
```

```
-z, --timezone=Local        Specify the timezone to use when formatting output
                            timestamps [Local, UTC]
    --output-timestamp-format=rfc3339
                            Specify the format of timestamps in the default
                            output mode [rfc3339, rfc3339nano, rfc822z,
                            rfc1123z, stampmicro, stampmilli, stampnano,
                            unixdate]
    --cpuprofile=""         Specify the location for writing a CPU profile.
    --memprofile=""         Specify the location for writing a memory profile.
    --[no-]stdin            Take input logs from stdin
    --addr="http://localhost:3100"
                            Server address. Can also be set using LOKI_ADDR
                            env var. ($LOKI_ADDR)
    --username=""           Username for HTTP basic auth. Can also be set
                            using LOKI_USERNAME env var. ($LOKI_USERNAME)
    --password=""           Password for HTTP basic auth. Can also be set
                            using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
    --ca-cert=""            Path to the server Certificate Authority.
                            Can also be set using LOKI_CA_CERT_PATH env var.
                            ($LOKI_CA_CERT_PATH)
    --[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
                            be set using LOKI_TLS_SKIP_VERIFY env var.
                            ($LOKI_TLS_SKIP_VERIFY)
    --cert=""               Path to the client certificate. Can also
                            be set using LOKI_CLIENT_CERT_PATH env var.
                            ($LOKI_CLIENT_CERT_PATH)
    --key=""                Path to the client certificate key. Can also
                            be set using LOKI_CLIENT_KEY_PATH env var.
                            ($LOKI_CLIENT_KEY_PATH)
    --org-id=""             adds X-Scope-OrgID to API requests for
                            representing tenant ID. Useful for requesting
                            tenant data when bypassing an auth gateway.
                            Can also be set using LOKI_ORG_ID env var.
                            ($LOKI_ORG_ID)
    --query-tags=""         adds X-Query-Tags http header to API requests.
                            This header value will be part of `metrics.go`
                            statistics. Useful for tracking the query.
                            Can also be set using LOKI_QUERY_TAGS env var.
                            ($LOKI_QUERY_TAGS)
```

```
      --[no-]nocache          adds Cache-Control: no-cache http header to API
                              requests. Can also be set using LOKI_NO_CACHE env
                              var. ($LOKI_NO_CACHE)
      --bearer-token=""       adds the Authorization header to API requests for
                              authentication purposes. Can also be set using
                              LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
      --bearer-token-file=""  adds the Authorization header to API requests
                              for authentication purposes. Can also be
                              set using LOKI_BEARER_TOKEN_FILE env var.
                              ($LOKI_BEARER_TOKEN_FILE)
      --retries=0             How many times to retry each query when
                              getting an error response from Loki. Can also
                              be set using LOKI_CLIENT_RETRIES env var.
                              ($LOKI_CLIENT_RETRIES)
      --min-backoff=0         Minimum backoff time between retries. Can also
                              be set using LOKI_CLIENT_MIN_BACKOFF env var.
                              ($LOKI_CLIENT_MIN_BACKOFF)
      --max-backoff=0         Maximum backoff time between retries. Can also
                              be set using LOKI_CLIENT_MAX_BACKOFF env var.
                              ($LOKI_CLIENT_MAX_BACKOFF)
      --auth-header="Authorization"
                              The authorization header used. Can also
                              be set using LOKI_AUTH_HEADER env var.
                              ($LOKI_AUTH_HEADER)
      --proxy-url=""          The http or https proxy to use when
                              making requests. Can also be set
                              using LOKI_HTTP_PROXY_URL env var.
                              ($LOKI_HTTP_PROXY_URL)
      --[no-]compress         Request that Loki compress returned
                              data in transit. Can also be set
                              using LOKI_HTTP_COMPRESSION env var.
                              ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy         Use ProxyFromEnvironment to use net/http
                              ProxyFromEnvironment configuration, eg HTTP_PROXY
                              ($LOKI_ENV_PROXY)

Subcommands:
delete create [<flags>] <query>
    Create a new log deletion request.
```

```
    The "delete create" command creates a new log deletion request for the
    specified query and time range.

    Example:

      logcli delete create '{job="app"}' --from="2023-01-01T00:00:00Z" --to="2023-01-
02T00:00:00Z"

delete list
    List existing log deletion requests.

    The "delete list" command lists all existing log deletion requests for the
    tenant.

    Example:

      logcli delete list
      logcli delete list --output=json

delete cancel --request-id=REQUEST-ID [<flags>]
    Cancel a log deletion request.

    The "delete cancel" command cancels an existing log deletion request by ID.

    Example:

      logcli delete cancel --request-id="abc123"
      logcli delete cancel --request-id="abc123" --force
```

`delete create` command reference

The output of `logcli help delete create`:

```
usage: logcli delete create [<flags>] <query>

Create a new log deletion request.

The "delete create" command creates a new log deletion request for the specified
```

```
query and time range.

Example:

  logcli delete create '{job="app"}' --from="2023-01-01T00:00:00Z" --to="2023-01-02T00:00:00Z"


Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
                              and --help-man).
      --[no-]version          Show application version.
  -q, --[no-]quiet            Suppress query metadata
      --[no-]stats            Show query statistics
  -o, --output=default        Specify output mode [default, raw, jsonl].
                              raw suppresses log labels and timestamp.
  -z, --timezone=Local        Specify the timezone to use when formatting output
                              timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                              Specify the format of timestamps in the default
                              output mode [rfc3339, rfc3339nano, rfc822z,
                              rfc1123z, stampmicro, stampmilli, stampnano,
                              unixdate]
      --cpuprofile=""         Specify the location for writing a CPU profile.
      --memprofile=""         Specify the location for writing a memory profile.
      --[no-]stdin            Take input logs from stdin
      --addr="http://localhost:3100"
                              Server address. Can also be set using LOKI_ADDR
                              env var. ($LOKI_ADDR)
      --username=""           Username for HTTP basic auth. Can also be set
                              using LOKI_USERNAME env var. ($LOKI_USERNAME)
      --password=""           Password for HTTP basic auth. Can also be set
                              using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
      --ca-cert=""            Path to the server Certificate Authority.
                              Can also be set using LOKI_CA_CERT_PATH env var.
                              ($LOKI_CA_CERT_PATH)
      --[no-]tls-skip-verify  Server certificate TLS skip verify. Can also
                              be set using LOKI_TLS_SKIP_VERIFY env var.
                              ($LOKI_TLS_SKIP_VERIFY)
      --cert=""               Path to the client certificate. Can also
```

```
                        be set using LOKI_CLIENT_CERT_PATH env var.
                        ($LOKI_CLIENT_CERT_PATH)
    --key=""            Path to the client certificate key. Can also
                        be set using LOKI_CLIENT_KEY_PATH env var.
                        ($LOKI_CLIENT_KEY_PATH)
    --org-id=""         adds X-Scope-OrgID to API requests for
                        representing tenant ID. Useful for requesting
                        tenant data when bypassing an auth gateway.
                        Can also be set using LOKI_ORG_ID env var.
                        ($LOKI_ORG_ID)
    --query-tags=""     adds X-Query-Tags http header to API requests.
                        This header value will be part of `metrics.go`
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
    --[no-]nocache      adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
    --bearer-token=""   adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""  adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
    --retries=0         How many times to retry each query when
                        getting an error response from Loki. Can also
                        be set using LOKI_CLIENT_RETRIES env var.
                        ($LOKI_CLIENT_RETRIES)
    --min-backoff=0     Minimum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MIN_BACKOFF env var.
                        ($LOKI_CLIENT_MIN_BACKOFF)
    --max-backoff=0     Maximum backoff time between retries. Can also
                        be set using LOKI_CLIENT_MAX_BACKOFF env var.
                        ($LOKI_CLIENT_MAX_BACKOFF)
    --auth-header="Authorization"
                        The authorization header used. Can also
                        be set using LOKI_AUTH_HEADER env var.
                        ($LOKI_AUTH_HEADER)
```

```
      --proxy-url=""          The http or https proxy to use when
                              making requests. Can also be set
                              using LOKI_HTTP_PROXY_URL env var.
                              ($LOKI_HTTP_PROXY_URL)
      --[no-]compress         Request that Loki compress returned
                              data in transit. Can also be set
                              using LOKI_HTTP_COMPRESSION env var.
                              ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy         Use ProxyFromEnvironment to use net/http
                              ProxyFromEnvironment configuration, eg HTTP_PROXY
                              ($LOKI_ENV_PROXY)
      --since=1h              Lookback window.
      --from=FROM             Start time for deletion (inclusive)
      --to=TO                 End time for deletion (exclusive)
      --max-interval=MAX-INTERVAL
                              Maximum time interval for delete request

Args:
  <query>  LogQL query to match log lines for deletion (e.g. '{job="app"}')
```

`delete list` command reference

The output of `logcli help delete list`:

```
usage: logcli delete list

List existing log deletion requests.

The "delete list" command lists all existing log deletion requests for the
tenant.

Example:

  logcli delete list
  logcli delete list --output=json



Flags:
      --[no-]help             Show context-sensitive help (also try --help-long
```

```
                           and --help-man).
    --[no-]version         Show application version.
-q, --[no-]quiet           Suppress query metadata
    --[no-]stats           Show query statistics
-o, --output=default       Specify output mode [default, raw, jsonl].
                           raw suppresses log labels and timestamp.
-z, --timezone=Local       Specify the timezone to use when formatting output
                           timestamps [Local, UTC]
    --output-timestamp-format=rfc3339
                           Specify the format of timestamps in the default
                           output mode [rfc3339, rfc3339nano, rfc822z,
                           rfc1123z, stampmicro, stampmilli, stampnano,
                           unixdate]
    --cpuprofile=""        Specify the location for writing a CPU profile.
    --memprofile=""        Specify the location for writing a memory profile.
    --[no-]stdin           Take input logs from stdin
    --addr="http://localhost:3100"
                           Server address. Can also be set using LOKI_ADDR
                           env var. ($LOKI_ADDR)
    --username=""          Username for HTTP basic auth. Can also be set
                           using LOKI_USERNAME env var. ($LOKI_USERNAME)
    --password=""          Password for HTTP basic auth. Can also be set
                           using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
    --ca-cert=""           Path to the server Certificate Authority.
                           Can also be set using LOKI_CA_CERT_PATH env var.
                           ($LOKI_CA_CERT_PATH)
    --[no-]tls-skip-verify Server certificate TLS skip verify. Can also
                           be set using LOKI_TLS_SKIP_VERIFY env var.
                           ($LOKI_TLS_SKIP_VERIFY)
    --cert=""              Path to the client certificate. Can also
                           be set using LOKI_CLIENT_CERT_PATH env var.
                           ($LOKI_CLIENT_CERT_PATH)
    --key=""               Path to the client certificate key. Can also
                           be set using LOKI_CLIENT_KEY_PATH env var.
                           ($LOKI_CLIENT_KEY_PATH)
    --org-id=""            adds X-Scope-OrgID to API requests for
                           representing tenant ID. Useful for requesting
                           tenant data when bypassing an auth gateway.
                           Can also be set using LOKI_ORG_ID env var.
```

```
                        ($LOKI_ORG_ID)
    --query-tags=""         adds X-Query-Tags http header to API requests.
                            This header value will be part of `metrics.go`
                            statistics. Useful for tracking the query.
                            Can also be set using LOKI_QUERY_TAGS env var.
                            ($LOKI_QUERY_TAGS)
    --[no-]nocache          adds Cache-Control: no-cache http header to API
                            requests. Can also be set using LOKI_NO_CACHE env
                            var. ($LOKI_NO_CACHE)
    --bearer-token=""       adds the Authorization header to API requests for
                            authentication purposes. Can also be set using
                            LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""  adds the Authorization header to API requests
                            for authentication purposes. Can also be
                            set using LOKI_BEARER_TOKEN_FILE env var.
                            ($LOKI_BEARER_TOKEN_FILE)
    --retries=0             How many times to retry each query when
                            getting an error response from Loki. Can also
                            be set using LOKI_CLIENT_RETRIES env var.
                            ($LOKI_CLIENT_RETRIES)
    --min-backoff=0         Minimum backoff time between retries. Can also
                            be set using LOKI_CLIENT_MIN_BACKOFF env var.
                            ($LOKI_CLIENT_MIN_BACKOFF)
    --max-backoff=0         Maximum backoff time between retries. Can also
                            be set using LOKI_CLIENT_MAX_BACKOFF env var.
                            ($LOKI_CLIENT_MAX_BACKOFF)
    --auth-header="Authorization"
                            The authorization header used. Can also
                            be set using LOKI_AUTH_HEADER env var.
                            ($LOKI_AUTH_HEADER)
    --proxy-url=""          The http or https proxy to use when
                            making requests. Can also be set
                            using LOKI_HTTP_PROXY_URL env var.
                            ($LOKI_HTTP_PROXY_URL)
    --[no-]compress         Request that Loki compress returned
                            data in transit. Can also be set
                            using LOKI_HTTP_COMPRESSION env var.
                            ($LOKI_HTTP_COMPRESSION)
    --[no-]envproxy         Use ProxyFromEnvironment to use net/http
```

```
                              ProxyFromEnvironment configuration, eg HTTP_PROXY
                              ($LOKI_ENV_PROXY)
```

`delete cancel` command reference

The output of `logcli help delete cancel`:

```
usage: logcli delete cancel --request-id=REQUEST-ID [<flags>]

Cancel a log deletion request.

The "delete cancel" command cancels an existing log deletion request by ID.

Example:

  logcli delete cancel --request-id="abc123"
  logcli delete cancel --request-id="abc123" --force


Flags:
      --[no-]help              Show context-sensitive help (also try --help-long
                               and --help-man).
      --[no-]version           Show application version.
  -q, --[no-]quiet             Suppress query metadata
      --[no-]stats             Show query statistics
  -o, --output=default         Specify output mode [default, raw, jsonl].
                               raw suppresses log labels and timestamp.
  -z, --timezone=Local         Specify the timezone to use when formatting
                               output timestamps [Local, UTC]
      --output-timestamp-format=rfc3339
                               Specify the format of timestamps in the default
                               output mode [rfc3339, rfc3339nano, rfc822z,
                               rfc1123z, stampmicro, stampmilli, stampnano,
                               unixdate]
      --cpuprofile=""          Specify the location for writing a CPU profile.
      --memprofile=""          Specify the location for writing a memory
                               profile.
      --[no-]stdin             Take input logs from stdin
      --addr="http://localhost:3100"
```

```
                        Server address. Can also be set using LOKI_ADDR
                        env var. ($LOKI_ADDR)
    --username=""       Username for HTTP basic auth. Can also be set
                        using LOKI_USERNAME env var. ($LOKI_USERNAME)
    --password=""       Password for HTTP basic auth. Can also be set
                        using LOKI_PASSWORD env var. ($LOKI_PASSWORD)
    --ca-cert=""        Path to the server Certificate Authority.
                        Can also be set using LOKI_CA_CERT_PATH env var.
                        ($LOKI_CA_CERT_PATH)
    --[no-]tls-skip-verify   Server certificate TLS skip verify. Can also
                        be set using LOKI_TLS_SKIP_VERIFY env var.
                        ($LOKI_TLS_SKIP_VERIFY)
    --cert=""           Path to the client certificate. Can also
                        be set using LOKI_CLIENT_CERT_PATH env var.
                        ($LOKI_CLIENT_CERT_PATH)
    --key=""            Path to the client certificate key. Can also
                        be set using LOKI_CLIENT_KEY_PATH env var.
                        ($LOKI_CLIENT_KEY_PATH)
    --org-id=""         adds X-Scope-OrgID to API requests for
                        representing tenant ID. Useful for requesting
                        tenant data when bypassing an auth gateway.
                        Can also be set using LOKI_ORG_ID env var.
                        ($LOKI_ORG_ID)
    --query-tags=""     adds X-Query-Tags http header to API requests.
                        This header value will be part of `metrics.go`
                        statistics. Useful for tracking the query.
                        Can also be set using LOKI_QUERY_TAGS env var.
                        ($LOKI_QUERY_TAGS)
    --[no-]nocache      adds Cache-Control: no-cache http header to API
                        requests. Can also be set using LOKI_NO_CACHE env
                        var. ($LOKI_NO_CACHE)
    --bearer-token=""   adds the Authorization header to API requests for
                        authentication purposes. Can also be set using
                        LOKI_BEARER_TOKEN env var. ($LOKI_BEARER_TOKEN)
    --bearer-token-file=""   adds the Authorization header to API requests
                        for authentication purposes. Can also be
                        set using LOKI_BEARER_TOKEN_FILE env var.
                        ($LOKI_BEARER_TOKEN_FILE)
    --retries=0         How many times to retry each query when
```

```
                          getting an error response from Loki. Can also
                          be set using LOKI_CLIENT_RETRIES env var.
                          ($LOKI_CLIENT_RETRIES)
      --min-backoff=0     Minimum backoff time between retries. Can also
                          be set using LOKI_CLIENT_MIN_BACKOFF env var.
                          ($LOKI_CLIENT_MIN_BACKOFF)
      --max-backoff=0     Maximum backoff time between retries. Can also
                          be set using LOKI_CLIENT_MAX_BACKOFF env var.
                          ($LOKI_CLIENT_MAX_BACKOFF)
      --auth-header="Authorization"
                          The authorization header used. Can also
                          be set using LOKI_AUTH_HEADER env var.
                          ($LOKI_AUTH_HEADER)
      --proxy-url=""      The http or https proxy to use when
                          making requests. Can also be set
                          using LOKI_HTTP_PROXY_URL env var.
                          ($LOKI_HTTP_PROXY_URL)
      --[no-]compress     Request that Loki compress returned
                          data in transit. Can also be set
                          using LOKI_HTTP_COMPRESSION env var.
                          ($LOKI_HTTP_COMPRESSION)
      --[no-]envproxy     Use ProxyFromEnvironment to use net/http
                          ProxyFromEnvironment configuration, eg HTTP_PROXY
                          ($LOKI_ENV_PROXY)
      --request-id=REQUEST-ID  ID of the delete request to cancel
      --[no-]force        Force cancellation of partially completed request
```

Use `--stdin` to query locally

You can use the logcli `–stdin` argument to run a command against a log file on your local machine, instead of a Loki instance. This lets you use LogQL to query a local log file without having to load the file into Loki, for example if you have downloaded a log file and want to query it outside of Loki.

If you have log files in your local machine, and just want to run some LogQL queries against those log files, `--stdin` flag can help.

You may use `stdin` flag to do the following:

- Use as a quick way to test or validate a LogQL expression against some log data.
- Learn the basics of LogQL with just local log files and the `logcli` tool (without needing to set up Loki servers, Grafana, etc.).

- Enable troubleshooting by letting you run queries without accessing a Loki instance.
- Use LogQL to parse and extract data from a local log file without ingesting the data into Loki.
- Enable discussion on public forums, for example submitting questions and answers, and sharing LogQL expressions.

Notes on `stdin` usage

1. The `--limit` flag doesn't have any meaning when using `--stdin` (use pager like `less` for that).
2. Be aware there are no **labels** when using `--stdin`. So the stream selector in the query is optional, for example, just `|="timeout"|logfmt|level="error"` is same as `{foo="bar"}|="timeout|logfmt|level="error"`.

{{< admonition type="note" >}} Currently `stdin` doesn't support any type of metric queries. {{< /admonition >}}

`stdin` examples

- Line filter - `cat mylog.log | logcli --stdin query '|="too many open connections"'`
- Label matcher - `echo 'msg="timeout happened" level="warning"' | logcli --stdin query '|logfmt|level="warning"'`
- Different parsers (logfmt, json, pattern, regexp) - `cat mylog.log | logcli --stdin query '|pattern <ip> - - <_> "<method> <uri> <_>" <status> <size> <_> "<agent>" <_>'`
- Line formatters - `cat mylog.log | logcli --stdin query '|logfmt|line_format "{{.query}} {{.duration}}"'`

Batching

logcli sends queries to Loki in such a way that query results arrive in batches.

The `--limit` option for a `logcli query` command limits the total number of log lines that will be returned for a single query. When not set, `--limit` defaults to 30. The limit protects the user from overwhelming Loki in cases where the specified query would have returned a large number of log lines. The limit also protects the user from unexpectedly large responses.

Larger result sets can be batched for easier consumption. Use the `--batch` option to control the number of log line results that are returned in each batch. When not set, `--batch` defaults to 1000.

Setting a `--limit` value larger than the `--batch` value will cause the requests from logcli to Loki to be batched.

When you run a query in Loki, it will return up to a certain number of log lines. By default, this limit is 5000 lines. You can configure this server limit with the `limits_config.max_entries_limit_per_query`

in Loki's configuration.

Batching lets you query for a results set that is larger than this server-side limit, as long as the `--batch` value is less than the server limit.

Query metadata is output to `stderr` for each batch. To suppress the output of the query metadata, set the `--quiet` option on the `logcli query` command line.

logcli example queries

Here are some examples of logcli.

Find all values for a label.

```
logcli labels job
```

```
https://logs-dev-ops-tools1.grafana.net/api/prom/label/job/values
loki-ops/consul
loki-ops/loki-gw
```

Print all labels and their unique values. This command is especially useful for finding high-cardinality labels in the index.

```
logcli series '{cluster="vinson"}' --analyze-labels
```

```
2024/10/31 13:46:25 https://logs-prod-
008.grafana.net/loki/api/v1/series?end=1730382385746344416&match=%7Bcluster%3D%22vinson%22%7D&
start=1730378785746344416
Total Streams:  10
Unique Labels:  10

Label Name       Unique Values  Found In Streams
service_name         8          10
pod                  7          7
job                  6          10
app_kubernetes_io_name  6           6
container            5          7
namespace            3          10
stream               2          7
```

```
flags               1          7
instance            1          3
cluster             1          10
```

Get all logs for a given stream

```
logcli query '{job="loki-ops/consul"}'
```

```
https://logs-dev-ops-tools1.grafana.net/api/prom/query?query=%7Bjob%3D%22loki-
ops%2Fconsul%22%7D&limit=30&start=1529928228&end=1529931828&direction=backward&regexp=
Common labels: {job="loki-ops/consul", namespace="loki-ops"}
2018-06-25T12:52:09Z {instance="consul-8576459955-pl75w"} 2018/06/25 12:52:09 [INFO] raft:
Snapshot to 475409 complete
2018-06-25T12:52:09Z {instance="consul-8576459955-pl75w"} 2018/06/25 12:52:09 [INFO] raft:
Compacting logs from 456973 to 465169
```

Print all log streams for the given stream selector. This example shows all known label combinations that match your query.

```
logcli series -q --match='{namespace="loki",container_name="loki"}'
```

```
{app="loki", container_name="loki", controller_revision_hash="loki-57c9df47f4",
filename="/var/log/pods/loki_loki-0_8ed03ded-bacb-4b13-a6fe-53a445a15887/loki/0.log",
instance="loki-0", job="loki/loki", name="loki", namespace="loki", release="loki",
statefulset_kubernetes_io_pod_name="loki-0", stream="stderr"}
```

Delete all logs for a specific job within a time range:

```
logcli delete create '{job="app"}' --from="2023-01-01T00:00:00Z" --to="2023-01-02T00:00:00Z"
```

List all deletion requests:

```
logcli delete list
```

Cancel a specific deletion request:

```
logcli delete cancel --request-id="abc123"
```

Troubleshoot logcli

Make sure that the version of Logcli you are using matches your Loki version. You can check your logcli version with the following command:

```
logcli –version
```

If you experience timeouts, you can update the following setting in your `logcli-config.yaml` file.

```
limits_config:
  query_timeout: 10m
```

LogCLI tutorial

LogCLI tutorial

This LogCLI tutorial will walk you through the following concepts:

- Querying logs
- Meta Queries against your Loki instance
- Queries against static log files

Prerequisites

Before you begin, you need to have the following:

- Docker
- Docker-compose
- LogCLI installed on your machine (see LogCLI installation)

{{< admonition type="tip" >}} Alternatively, you can try out this example in our interactive learning environment: LogCLI Tutorial.

It's a fully configured environment with all the dependencies already installed.

Interactive

Provide feedback, report bugs, and raise issues in the Grafana Killercoda repository. {{< /admonition >}}

Scenario

You are a site manager for a new logistics company. The company uses structured logs to keep track of every shipment sent and received. The payload format looks like this:

{"timestamp": "2024-11-22T13:22:56.377884", "state": "New York", "city": "Buffalo", "package_id": "PKG34245", "package_type": "Documents", "package_size": "Medium", "package_status": "error", "note": "Out for delivery", "sender": {"name": "Sender27", "address": "144 Elm St, Buffalo, New York"}, "receiver": {"name": "Receiver4", "address": "260 Cedar Blvd, New York City, New York"}}

The logs are processed from Grafana Alloy to extract labels and structured metadata before they're stored in Loki. You have been tasked with monitoring the logs using the LogCLI and build a report on the overall health of the shipments.

Setup

To get started, we need to clone the Alloy Scenario repository and start the mail-house example:

1. Clone the repository:

   ```
   git clone https://github.com/grafana/alloy-scenarios.git
   ```

2. Start the mail-house example:

   ```
   docker compose -f alloy-scenarios/mail-house/docker-compose.yml up -d
   ```

This will start the mail-house example and expose the Loki instance at `http://localhost:3100`. We have also included a Grafana instance to verify the LogCLI results which can be accessed at http://localhost:3000.

Connecting LogCLI to Loki

To connect LogCLI to the Loki instance, you need to set the `LOKI_ADDR` environment variable:

{{< admonition type="tip" >}}

If you are running this example against your own Loki instance and have configured authentication, you will also need to set the `LOKI_USERNAME` and `LOKI_PASSWORD` environment variables.

{{< /admonition >}}

```
export LOKI_ADDR=http://localhost:3100
```

Now let's verify the connection by running the following command:

```
logcli labels
```

This should return an output similar to the following:

```
http://localhost:3100/loki/api/v1/labels?end=1732282703894072000&start=1732279103894072000
package_size
service_name
state
```

This confirms that LogCLI is connected to the Loki instance and we now know that the logs contain the following labels: `package_size`, `service_name`, and `state`. Let's run some queries against Loki to better understand our package logistics.

Querying logs

As part of our role within the logistics company, we need to build a report on the overall health of the shipments. Unfortunately, we only have access to a console and cannot use Grafana to visualize the data. We can use LogCLI to query the logs and build the report.

Find all critical packages

To find all critical packages in the last hour (the default lookback time), we can run the following query:

```
logcli query '{service_name="Delivery World"} | package_status="critical"'
```

This will return all logs where the `service_name` is `Delivery World` and the `package_status` is `critical`. The output will look similar to the following:

```
http://localhost:3100/loki/api/v1/query_range?direction=BACKWARD&end=1732617594381712000&limit=30&query=%7Bservice_name%3D%22Delivery+World%22%7D+%7C+package_status%3D%22critical%22&start=1732613994381712000
Common labels: {package_status="critical", service_name="Delivery World"}
2024-11-26T10:39:52Z {package_id="PKG79755", package_size="Small", state="Texas"}
{"timestamp": "2024-11-26T10:39:52.521602Z", "state": "Texas", "city": "Dallas", "package_id": "PKG79755", "package_type": "Clothing", "package_size": "Small", "package_status": "critical", "note": "In transit", "sender": {"name": "Sender38", "address": "906 Maple Ave, Dallas, Texas"}, "receiver": {"name": "Receiver41", "address": "455 Pine Rd, Dallas, Texas"}}
2024-11-26T10:39:50Z {package_id="PKG34018", package_size="Large", state="Illinois"}
```

{"timestamp": "2024-11-26T10:39:50.510841Z", "state": "Illinois", "city": "Chicago", "package_id": "PKG34018", "package_type": "Clothing", "package_size": "Large", "package_status": "critical", "note": "Delayed due to weather", "sender": {"name": "Sender22", "address": "758 Elm St, Chicago, Illinois"}, "receiver": {"name": "Receiver10", "address": "441 Cedar Blvd, Naperville, Illinois"}}

Lets suppose we want to look back for the last 24 hours, we can use the `--since` flag to specify the time range:

```
logcli query --since 24h '{service_name="Delivery World"} | package_status="critical"'
```

This will query all logs for the `package_status` `critical` in the last 24 hours. However it will not return all of the logs, but only the first 30 logs. We can use the `--limit` flag to specify the number of logs to return:

```
logcli query --since 24h --limit 100 '{service_name="Delivery World"} |
package_status="critical"'
```

Metric queries

We can also use LogCLI to query logs based on metrics. For instance as part of the site report we want to count the total number of packages sent from California in the last 24 hours in 1 hour intervals. We can use the following query:

```
logcli query --since 24h 'sum(count_over_time({state="California"}[1h]))'
```

This will return a JSON object containing a list of timestamps (Unix format) and the number of packages sent from California in 1 hour intervals. Since we summing the count of logs over time, we will see the total number of logs steadily increase over time. The output will look similar to the following:


```
[
  {
    "metric": {},
    "values": [
      [
        1733913765,
        "46"
      ],
      [
        1733914110,
```

```
        "114"
      ],
      [
        1733914455,
        "179"
      ],
      [
        1733914800,
        "250"
      ],
      [
        1733915145,
        "318"
      ],
      [
        1733915490,
        "392"
      ],
      [
        1733915835,
        "396"
      ]
    ]
  }
]
```

We can take this a step further and filter the logs based on the `package_type` label. For instance, we can count the number of documents sent from California in the last 24 hours in 1 hour intervals:

```
logcli query --since 24h  'sum(count_over_time({state="California"}| json | package_type=
"Documents" [1h]))'
```

This will return a similar JSON object above but will only show a trend of the number of documents sent from California in 1 hour intervals.

Instant metric queries

Instant metric queries are a subset of metric queries that return the value of the metric at a specific point in time. This can be useful for quickly understanding an aggregate state of the stored logs.

For instance, we can use the following query to get the number of packages sent from California in the last 5 minutes:

```
logcli instant-query 'sum(count_over_time({state="California"}[5m]))'
```

This will return a result similar to the following:

```
[
  {
    "metric": {},
    "value": [
      1732702998.725,
      "58"
    ]
  }
]
```

Writing query results to a file

Another useful feature of LogCLI is the ability to write the query results to a file. This can be useful for downloading the results of our inventory report:

First we need to create a directory to store the logs:

```
mkdir -p ./inventory
```

Next we can run the following query to write the logs to the `./inventory` directory:

```
  logcli query \
     --timezone=UTC \
     --output=jsonl \
     --parallel-duration="12h" \
     --parallel-max-workers="4" \
     --part-path-prefix="./inventory/inv" \
     --since=24h \
     '{service_name="Delivery World"}'
```

This will write all logs for the `service_name` `Delivery World` in the last 24 hours to the `./inventory` directory. The logs will be split into two files, each containing 12 hours of logs. Note that we do not need to specify `--limit` as this is overridden by the `--parallel-duration` flag.

Meta queries

As site managers, it's essential to maintain good data hygiene and ensure Loki operates efficiently. Understanding the labels and log volume in your logs plays a key role in this process. Beyond querying logs, LogCLI also supports meta queries on your Loki instance. Meta queries don't return log data but provide insights into the structure of your logs and the performance of your queries. The following examples demonstrate some of the core meta queries we run internally to better understand how a Loki instance is performing.

Checking series cardinality

One of the most important aspects of keeping Loki healthy is to monitor the series cardinality. This is the number of unique series in your logs. A high series cardinality can lead to performance issues and increased storage costs. We can use LogCLI to check the series cardinality of our logs.

To start let's print how many unique series we have in our logs:

```
logcli series '{}'
```

This will return a list of all the unique series in our logs. The output will look similar to the following:

```
{package_size="Small", service_name="Delivery World", state="Florida"}
{package_size="Medium", service_name="Delivery World", state="Florida"}
{package_size="Small", service_name="Delivery World", state="California"}
{package_size="Large", service_name="Delivery World", state="New York"}
{package_size="Small", service_name="Delivery World", state="Illinois"}
{package_size="Large", service_name="Delivery World", state="Florida"}
{package_size="Medium", service_name="Delivery World", state="Illinois"}
{package_size="Large", service_name="Delivery World", state="Texas"}
{package_size="Medium", service_name="Delivery World", state="California"}
{package_size="Medium", service_name="Delivery World", state="Texas"}
{package_size="Small", service_name="Delivery World", state="Texas"}
{package_size="Large", service_name="Delivery World", state="Illinois"}
{package_size="Small", service_name="Delivery World", state="New York"}
{package_size="Medium", service_name="Delivery World", state="New York"}
{package_size="Large", service_name="Delivery World", state="California"}
```

We can further improve this query by adding `--analyze-labels`:

```
logcli series '{}' --analyze-labels
```

This will return a summary of the unique values for each label in our logs. The output will look similar to the following:

```
Label Name    Unique Values  Found In Streams
state         5              15
package_size  3              15
service_name  1              15
```

Detected fields

Another useful feature of LogCLI is the ability to detect fields in your logs. This can be useful for understanding the structure of your logs and the keys that are present. This will let us detect keys which could be promoted to labels or to structured metadata.

```
logcli detected-fields --since 24h '{service_name="Delivery World"}'
```

This will return a list of all the keys detected in our logs. The output will look similar to the following:

```
label: city                   type: string  cardinality: 15
label: detected_level         type: string  cardinality: 3
label: note                   type: string  cardinality: 7
label: package_id             type: string  cardinality: 994
label: package_size_extracted type: string  cardinality: 3
label: package_status         type: string  cardinality: 4
label: package_type           type: string  cardinality: 5
label: receiver_address       type: string  cardinality: 991
label: receiver_name          type: string  cardinality: 100
label: sender_address         type: string  cardinality: 991
label: sender_name            type: string  cardinality: 100
label: state_extracted        type: string  cardinality: 5
label: timestamp              type: string  cardinality: 1000
```

You can now see why we opted to keep `package_id` in structured metadata and `package_size` as a label. Package ID has a high cardinality and is unique to each log entry, making it a good candidate for structured metadata since we potentially may need to query for it directly. Package size, on the other hand, has a low cardinality, making it a good candidate for a label.

Checking query performance

Another important aspect of keeping Loki healthy is to monitor the query performance. We can use LogCLI to check the query performance of our logs.

{{< admonition type="note" >}} The LogCLI can only return statistics for queries that touch object storage. In this example we force the Loki ingesters to flush chunks every 5 minutes which isn't recommended for production use. When running this demo if you don't see any statistics returned, try running the command again after a few minutes. {{< /admonition >}}

To start lets print the query performance of our logs:

```
logcli stats --since 24h '{service_name="Delivery World"}'
```

This will provide a JSON object containing statistics on the amount of data queried. The output will look similar to the following:

```
http://localhost:3100/loki/api/v1/index/stats?end=1732639430272850000&query=%7Bservice_name%3D
%22Delivery+World%22%7D&start=1732553030272850000
{
  bytes: 12MB
  chunks: 63
  streams: 15
  entries: 29529
}
```

This will return the total number of bytes queried, the number of chunks queried, the number of streams queried, and the number of entries queried. If we narrow down the query by specifying a secondary label, we can see the performance of the query:

```
logcli stats --since 24h '{service_name="Delivery World", package_size="Large"}'
```

This will return the statistics for the logs where the `service_name` is `Delivery World` and the `package_size` is `Large`. The output will look similar to the following:

```
{
  bytes: 4.2MB
  chunks: 22
  streams: 5
  entries: 10198
}
```

As you can see, we touched far fewer streams and entries by narrowing down the query.

Checking the log volume

We may also want to check the log volume in our logs. This can be useful for understanding the amount of data being ingested into Loki. We can use LogCLI to check the log volume in our logs.

```
logcli volume --since 24h '{service_name="Delivery World"}'
```

This returns the total number of logs ingested for the label `Delivery World` in the last 24 hours. The output will look similar to the following:

```
[
  {
    "metric": {
      "service_name": "Delivery World"
    },
    "value": [
      1732640292.354,
      "11669299"
    ]
  }
]
```

The result includes the timestamp and the total number of logs ingested.

We can also return the log volume over time by using `volume_range`:

```
logcli volume_range --since 24h --step=1h '{service_name="Delivery World"}'
```

This will provide a JSON object containing the log volume for the label `Delivery World` in the last 24 hours. `--step` will aggregate the log volume into 1 hour buckets. Note that if there are no logs for a specific hour, the log volume for that hour will not return a value.

We can even aggregate the log volume into buckets based on a specific labels value:

```
logcli volume_range --since 24h --step=1h --targetLabels='state' '{service_name="Delivery
World"}'
```

This will provide a similar JSON object but will aggregate the log volume into buckets based on the `state` label value.

Queries against static log files

In addition to querying logs from Loki, LogCLI also supports querying static log files. This can be useful for querying logs that are not stored in Loki. Earlier in the tutorial we stored the logs in the `./inventory` directory. Lets run a similar query but pipe it into a log file:

```
  logcli query \
     --timezone=UTC \
     --parallel-duration="12h" \
     --parallel-max-workers="4" \
     --part-path-prefix="./inventory/inv" \
     --since=24h \
     --merge-parts \
     --output=raw \
     '{service_name="Delivery World"}' > ./inventory/complete.log
```

Next lets run a query against the static log file:

```
cat ./inventory/complete.log |  logcli --stdin query '{service_name="Delivery World"} | json | package_status="critical"'
```

Note that since we are querying a static log file, labels are not automatically detected:

- `{service_name="Delivery World"}` is optional in this case but is recommended for clarity.
- `json` is required to parse the log file as JSON. This lets us extract the `package_status` field.

For example, suppose we try to query the log file without the `json` filter:

```
cat ./inventory/complete.log | logcli --stdin query '{service_name="Delivery World"} | package_status="critical"'
```

This will return no results as the `package_status` field is not detected.

Conclusion

In this tutorial as site manager for a logistics company, we have successfully used LogCLI to query logs and build a report on the overall health of the shipments. We have also used meta queries to better understand our data cleanliness and query performance. The LogCLI is a powerful tool for understanding your logs and how they are stored in Loki, as you continue to scale your solution remember to keep LogCLI in mind to monitor cardinality and query performance.