Logging Cheat Sheet

chronosphere

This reference covers core language features of Chronosphere logging. See the documentation for more information.

Mental model

Data flows through logging as a single table, which can be **filtered**, **transformed, aggregated,** and **post** processed**. All steps are optional** and independent, but aggregations must be used with a transformation.
All **steps are separated by a pipe** | to indicate data flow.

Query structure

```
filter | transformation aggregations | post
```

Steps are separated by a pipe. Every query can be read as, starting with all the logs: apply my filterExpression, then aggregate and transform to the desired shape.

filter Syntax

```
FIELD =|!=|=~|!~|: VALUE AND|OR FIELD
=|!=|=~|!~|: VALUE AND|OR "full-text
search value" AND| OR NOT KEY EXISTS
# this is a comment
```

Regular Expression

=~	Matches regex
!~	Does not match regex
:	Contains literal string only

Logging uses RE2 regular expression syntax.

Filter Notes

All values must be quoted, except boolean.
Filter operators are case insensitive (and = AND). Where no operator is specified between clauses, AND is assumed. See docs for array access syntax.

filter Example

```
# find test generated errors
severity = "Error" service = "payment"
and ("testUsr" or admin = true)
```

transformation Syntax

Transformations reshape your data based on ***aggregation***s and support different output shapes. **ALL ARGS OPTIONAL.**

make-series	Formats data for a time-series
summarize	Formats to a table (also bar/pie chart, etc_
top-nested	Formats to a table for hierarchical grouping

make-series Syntax

```
f | make-series aggregations step size by F
```

aggregation	aggregation(s) to use. Default ***count()***
size	Prom. format time duration (e.g., 1h)
F	Fields to group by (e.g., service, severity)

Supports series visualization only.

summarize Syntax

```
f | summarize aggregations by F
```

aggregation	aggregation(s) to use. Default ***count()***
F	Fields to group by (e.g., service, severity)

Supports table, bar chart, and stat.

aggregation Syntax

```
f | transformation count()
```
Counts number of rows from ***f***.
```
f | transformation countIf(filter)
```
Counts rows where ***filter*** *predicate is true.*
```
f | transformation avg(field)
```
Produces an average over numerical **field**.
```
f | transformation avgIf(field, filter)
```
More functions like **avg**: **dcount**, **sum**, **avg**, **min**, **max**
All have **If** functions, allowing **filter** syntax.
```
f | transformation arg_max(fieldM, field)
```
Returns value of **field** *for row with maximum* `fieldM`.
```
f | transformation percentile(field, number)
```
All stats functions ignore non-numerical values.

field	The field to run aggregation over. e.g., latency
filter	Filter to apply (e.g., `user.id: "myid"`)
number	Percentile whose value to return

Multiple Aggregation *(Coming Soon)*

```
f | t aggregation1, aggregation2 ... by G
```
A query can have a single ***transformation*** that produces multiple aggregations and columns.

Math on Aggregations

```
f | transformation aggregation1 +/*-
aggregation2 ... by G
```
Aggregations can be added, divided, multiplied, or subtracted where valid. Most useful for alerting with **If** functions, where you might divide to create a ratio.

Aggregation Naming

```
f | transformation name_1 = aggregation1,
name_2 = aggregation1/aggregation2
```
Aggregations can be named to improve query and table readability. Charts and tables will use this name.

substring Function

```
substring(field, start_idx, LENGTH)
```

field	Field to take the substring of.
start_idx	Start index of the substring.
length	Optional. If unspecified, to the end.

Substring is a special function that can be used anywhere a field can be used. For example in the **filter** to match a substring VALUE, in **If** functions, or in the **by** clause to manipulate grouping (e.g., to remove a prefix).

post Processing Syntax

Post processing functions apply after all previous (optional) steps, and do not change the shape of the data after transformation. They can:

limit	Limits the number of rows
sort	Sorts by selected fields
project	Selects columns to include and add/compute

```
f | t a | limit [limit]
f | t a | sort by field1 asc|desc, field2...
f | t a | project field1, field2
```

Math and Aliasing on Project *(Coming soon)*

```
f | t a | project field1, ratio = field1/f2
```

Logging Cheat Sheet

chronosphere

This page contains queries using the following fields and functions defined above. Their types are below.

service *string*	severity *string*	message *string*	request.latency *float*	k8s.deployment.id *string*	TraceId *string*	isAdmin *boolean*

`isAdmin = true and message: "test"`	**Simple filtering** to find admin logs containing "test" in message
`isAdmin = true and message: "test"` `| make-series by service, severity`	**Creating a time series** grouping by service and severity Note that `make-series` default aggregation is count().
`isAdmin = true and message: "test"` `| make-series avg(request.latency) by service, severity`	**Creating a time series** grouping by error and severity with average latencies
`isAdmin = true and severity = "ERROR" and TraceId EXISTS` `| project service, severity, TraceId, request.latency`	**Finding admin errors with traces** and projecting the latency
`severity = "ERROR"` `| summarize percentile(latency, 95) by service`	**Creating a table of 95th percentile** error latencies across services
`severity = "ERROR"` `| summarize arg_max(latency) by service`	**Find the last (most recent) error latency** per service `arg_max` default first parameter is *_timestamp*, allowing it to be used like a "last" function. It always reduces to a single value, like `max`.
`service = "auth" isAdmin = false` `| make-series countIf(severity = "ERROR") / count() step 5m`	**Creating an error rate/ratio** for a specific service
`summarize max(request.latency)` `by substring(k8s.deployment.id, 11, 36)`	**Manipulating strings using substring** to find maximum deployment latency across environments For this example, assume `k8s.deployment.id` contains values following the format **k8.[36-chararacter-id].[prod, dev, test]**. We want to create one row for every **guid,** collapsing values from **prod, dev, test** into that row using substring.