## Lightstep Unified Query Language (UQL) Cheat Sheet

**A UQL query is piped, made up of two or more stages that operate on the metric or span data returned by the Fetch stage. Subsequent stages operate on the data returned from the previous stage.**

<table>
<thead>
<tr>
<th>Stage 1: Fetch</th>
<th>Stage 2: Align</th>
<th>Stage 3: Filter</th>
<th>Stage 4: Group &amp; Aggregate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>metric request</strong></td>
<td><strong>rate</strong></td>
<td><strong>filter service = checkout</strong></td>
<td><strong>group_by [operation], sum</strong></td>
</tr>
</tbody>
</table>

### Fetch (required)

Returns the specified data type (metric or spans). Metric requires the metric name. Spans require a computation type.

**Fetching the rate (ops/s) of HTTP requests metric**

```
metric http.requests | rate | group_by [], sum
```

**Fetching the p95 latency of all spans**

```
spans latency | delta | group_by [], sum | point percentile(value, 95)
```

**Fetching the count of all spans as a rate (ops/s)**

```
spans count | rate | group_by [], sum
```

### Align (required)

Aligns points to regular time periods and can include an input window and output period. Output periods are required and only allowed on intermediate aligners. Delta and cumulative type metrics typically use the rate and delta aligners. Gauge metrics typically use reduce. Latest can only be used with gauges.

**Rate (ops/s) of HTTP requests metric**

```
metric http.requests | rate | group_by [], sum
```

**Rate (ops/s) of HTTP requests metric, averaged over previous 5 mins**

```
metric http.requests | rate 5m | group_by [], sum
```

### Filter

Drops data that doesn’t match the predicate.

**Rate (ops/s) of HTTP requests only if from the checkout service and in the us-east-1 time zone**

```
metric http.requests | rate 1m | filter service == checkout & zone == us-east-1 | group_by [], sum
```

**p95 latency for spans from the iOS service**

```
spans latency | delta | filter service == ios | group_by [], sum | point percentile(value, 95)
```

**p95 latency for spans from the iOS service and the customer Packing Kings**

```
spans latency | delta | filter service == ios & customer == "Packing Kings" | group_by [], sum | point percentile(value, 95)
```

### Time series

Separates time series into groups by the values of the given attributes, then for each value group, combines the time series using a reducing. Queries for spans data must include a group_by parameter. To get a single time series, use an empty group_by value, reduced by the sum.

**Time series for memory usage, by host**

```
metric kubernetes.memory.usage | latest | group_by [host], sum
```

**Maximum value of the rate of requests per second over a two minute period, grouped by the region and zone**

```
m先行 counts | rate 2m | group_by [region, zone], max
```

**Rate of spans per second on the checkout service, grouped by customer**

```
spans count | rate | filter service == checkout | group_by [customer], sum
```

### Group by (required for span queries)

Separates time series according to the input value of each data point (called value) and applies an expression to it.

**p95 value of the distribution metric my.hist**

```
metric my.hist | delta | group_by [], sum | point percentile(value, 95)
```

**Boolean-valued time series where a point is true if the value of my.metric is greater than 5**

```
metric my.metric | latest | group_by [], sum | point value > 5
```

**Squared value of each point in my.metric**

```
metric my.metric | latest | group_by [], sum | point pow(value, 2)
```

### Point filter

Keeps all points that match the predicate and removes all other points. This will produce gaps in the time series or remove time series altogether that don’t match the predicate.

**Latency values for services where the point value is greater than 1 second**

```
spans latency | delta | group_by [service], sum | point percentile(value, 99) | point_filter value > 1000
```

**Requests where the point value is less than 1,000 requests**

```
metric requests | delta | point_filter value < 1000
```

### Change in HTTP requests over previous hour

```
m先行 metric requests | delta 1h | group_by [], sum
```

### Maximum value of each time series over previous hour

```
m先行 metric http.requests | reduce 1h, max | group_by [], sum
```

### Maximum change over 1 minute of requests in the previous hour

```
m先行 metric http.requests | delta 1m, im | reduce 1h, max
```

### Exploit output period required for intermediate aligners (final aligners don’t have any output period)

```
m先行 metric requests | rate | group_by [], sum
```

### Gauge points aligned to a consistent period

```
m先行 metric.memory.usage | latest | group_by [], sum
```

### Percent of kube memory limits currently being used by dataingest, broken down by container and pod

```
m先行 metric kubernetes.memory.usage | latest | filter kube_app == dataingest | group_by [pod_name, container_name], sum
```

### Multiple latencies of the database-update operation on the warehouse service

```
m先行 spans latency | delta | filter service = warehouse & operation = database-update | group_by [], sum | point percentile(value, 99)
```

### Change in HTTP requests requests over previous hour

```
m先行 metric http.requests | delta 1h | group_by [], sum
```

### Maximum value of each time series over previous hour

```
m先行 metric http.requests | reduce 1h, max | group_by [], sum
```

### Maximum change over 1 minute of requests in the previous hour

```
m先行 metric http.requests | delta 1m, im | reduce 1h, max
```

### Exploit output period required for intermediate aligners (final aligners don’t have any output period)

```
m先行 metric requests | rate | group_by [], sum
```

### Gauge points aligned to a consistent period

```
m先行 metric.memory.usage | latest | group_by [], sum
```

### Exploit output period required for intermediate aligners (final aligners don’t have any output period)

```
m先行 metric requests | rate | group_by [], sum
```

### Exploit output period required for intermediate aligners (final aligners don’t have any output period)

```
m先行 metric requests | rate | group_by [], sum
```

### Exploit output period required for intermediate aligners (final aligners don’t have any output period)

```
m先行 metric requests | rate | group_by [], sum
```

### Exploit output period required for intermediate aligners (final aligners don’t have any output period)

```
m先行 metric requests | rate | group_by [], sum
```
A UQL query is piped, made up of two or more stages that operate on the metric or span data returned by the Fetch stage. Subsequent stages operate on the data returned from the previous stage.

### Alerts Cookbook

#### Seasonality alerts

Use this to get alerted on the historical comparison for `<metric>` between now and `<lookback>` duration ago. For example, to compare now and a week ago, use `time_shift 7d`.

<smoothingDuration> defines how much smoothing to apply to the historical data. The more smoothing, the less likely spikes in historical data will cause your alert to trigger. Too large a value here may cause your seasonality to be lost.

**Gauge metrics**

```plaintext
with
  a = metric <metric> | reduce <smoothingDuration>, mean | group_by [], mean;
  b = metric <metric> | time_shift <lookback> | reduce <smoothingDuration>, mean | group_by [], mean;
join ((a-b)/b) * 100
```

**Delta/Cumulative scalar metrics**

```plaintext
with
  a = metric <metricA> | delta | group_by [], sum;
  b = metric <metricB> | delta | group_by [], sum;
join a - b
```

**Delta/Cumulative distribution metrics**

```plaintext
with
  a = metric <metric> | delta <smoothingDuration> | group_by [], sum | point percentile(value, 95);
  b = metric <metric> | time_shift <lookback> | delta <smoothingDuration> | group_by [], sum | point percentile(value, 95);
join ((a-b)/b) * 100
```

#### Direct comparison alerts

Use this to get comparison alerts between `<metricA>` and `<metricB>`. For example, SQS messages in vs. out.

**Gauge metrics**

```plaintext
with
  a = metric <metricA> | latest | group_by [], mean;
  b = metric <metricB> | latest | group_by [], mean;
join a - b
```

**Delta/Cumulative scalar metrics**

```plaintext
with
  a = metric <metricA> | delta | group_by [], sum;
  b = metric <metricB> | delta | group_by [], sum;
join a - b
```