servicenow

Cloud Observability Unified Query Language (UQL) Cheat Sheet

Visit Cloud Observability Learning Portal for an intro to UQL and reference documentation

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.

Stage 1: Fetch
```
metric request
```
Stage 2: Align
```
| rate
```
Stage 3: Filter
```
| filter service == checkout
```
Stage 4: Group & Aggregate
```
| group_by [operation], sum
```

Fetch (required)

Returns the specified data type (metric or spans). Metric requires the metric name. Spans requires 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)
```

Allowed values: `latency` `count` `lightstep.bytesize`

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

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

Fetch the rate of all logs that contain an error in severity_text grouped by service name (logs/s)

```
logs count
| filter severity_text == "error"
| rate
| group_by [service.name], sum
```

Fetch all logs where the service equals web logs

```
logs
| filter service.name == "web"
```

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
```

Allowed values: `=` `==` `>` `>=` `<` `<=` `!` `!=` `&&` `||` `defined` `undefined` `contains` `phrase_match` `!~ (regex no match)` `=~ (regex match)`

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)
```

Quotes required for string values with spaces

Time shift

Moves each point in the time series forwards by the amount of time specified in the duration. A time_shift doesn't have to immediately follow a fetch operation, but it must come before a window operation or group_by.

Difference in request rate from one week prior

```
with current = metric requests | rate |
group_by [customer, method], sum; last_week =
metric requests | time_shift 7d | rate |
group_by [customer, method], sum;
join current - last_week
```

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 the point value is less than 1,000 requests

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

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
```

Allowed values: `rate` `delta` `latest` `reduce`

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

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

Allowed values: `s` `m` `h` `d` `w`

Change in HTTP request metric

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

Group by (required for spans & logs time series queries)

Separates time series into groups by the values of the given attributes, then for each value group, combines the time series using a reducer. Queries for span and log time series data must include a group_by. 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
```

Allowed values: `max` `min` `mean` `sum` `distribution` `count` `count_nonzero` `std_dev`

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

```
metric requests | rate 2m |  group_by [region,
zone], max
```

p95 latency for spans from the iOS service

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

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

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

Point

Takes 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)
```

Change in HTTP requests over previous hour

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

Maximum value of each time series over previous hour

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

Allowed values: `max` `min` `mean` `sum` `distribution` `count` `count_nonzero` `std_dev`

Maximum change over 1 minute of requests in the previous hour

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

Explicit output period required for intermediate aligners (final aligners don't have any output period)

Gauge points aligned to a consistent period

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

Join

Combines two or more sets of time series by matching attributes, then applies arithmetic to create a single set of time series. A join can result in a very large results set when labels have high cardinality. Using group_by before a join reduces the labels the join will match on.

Fraction of requests that were successful, broken down by HTTP method (for example, GET, PUT)

```
with
  successes  = metric http.requests.success |
delta 1m | group_by [http.method], sum;
  total = metric http.requests.total | delta
1m | group_by [http.method], sum;
join successes/total, successes = 0
```

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

```
with
  usage = metric kubernetes.memory.usage |
latest | filter kube_app == dataingest |
group_by [pod_name, container_name], sum;
  limits = metric kubernetes.memory.limits |
latest | filter kube_app == dataingest |
group_by [pod_name, container_name], sum;
join usage / limits * 100
```

p99 latency for spans from the database-update operation on the warehouse service

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

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

```
spans latency | delta | filter service =
warehouse && operation = database-update |
group_by [], sum | point percentile(value,
99.9), percentile(value, 99),
percentile(value, 90)
```

Allowed values: `*` `/` `+` `-` `pow(a,b)` `percentile(a,b)` `dist_sum(a)` `dist_count(a)`

servicenow

Cloud Observability Unified Query Language (UQL) Cheat Sheet

Visit Cloud Observability Learning Portal for an intro to UQL and reference documentation

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.

Stage 1: Fetch
```
metric request
```

Stage 2: Align
```
| rate
```

Stage 3: Filter
```
| filter service == checkout
```

Stage 4: Group & Aggregate
```
| group_by [operation], sum
```

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**.

Allowed values for time_shift: `s` `m` `h` `d` `w`

<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

```
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

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

Delta/Cumulative distribution metrics

```
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 alerted on the comparison between **<metricA>** and **<metricB>.** For example, SQS messages in vs. out.

Gauge metrics

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

Delta/Cumulative scalar metrics

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