### Metrics Operators

The following table lists the metrics supported operators and provides examples of queries containing each type of operator.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description and Syntax</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>accum</strong></td>
<td>Converts each time-series in the row to a series of running totals. The running total in each series starts from the value of the first data point in the series, then iteratively adds up successive values.</td>
<td></td>
</tr>
</tbody>
</table>
For example, if `RequestCount` is: 
{ 2, 0, 4, 3, 0, 0 }

`RequestCount | accum` is:
{ 2, 2, 6, 9, 9, 9 } |
| **avg** | Calculates the average of all the resulting time series. If grouping is specified, it calculates the average for each group. | 
`avg [by FIELD [, FIELD, ...]]` |
| **along** | For use when joining metric queries, results in the Sumo evaluating the summation expression by one or more metric fields. | 
`#A + #B along [FIELD (,FIELD, ...)]` |
| **bottomk** | Select the bottom specified time series sorted by the value of a mathematical expression evaluated over the query time range. | 
Take the bottom 5 time series with the highest maximum value: |
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description and Syntax</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>bottomk (number, aggregator)</td>
<td>Counts the total number of time series that match the query. If grouping is specified, it counts the total number for each group.</td>
<td>dep=prod metric=cpu_system</td>
</tr>
</tbody>
</table>

**Supported aggregate functions:**
min, max, avg, count, sum, pct(n), latest

**Examples:**

dep=prod | count
class=host | count by node

**count**

Computes the backward difference at each data point in the time series to determine how much the metric has changed from its last value in the series.

This operator also assigns the value of the metric tag to be `$delta(metric)$`.

**Examples:**

```
metric=Net_InBytes Interface=eth0 | delta
```  

**delta**

Evaluates a time series based on a user-specified math expression.

```
metrics query | eval <math expression>
```  

where math expression is a valid math expression with `_value` as the placeholder for each data point in the time series.  

**Supported Basic operations:**
+
-, *
,

**Supported Math functions:**

sin, cos, abs, log, round, ceil, floor, tan, exp, sqrt, min, max

**Examples:**

```
_metrics=request_per_sec | rate | eval max(_value, 0)
```  

```
_sourceCategory=ApacheHttpServer
_metrics=cpu_idle | eval _value * 100
```  

**eval**

Filters a query to help reduce the number of series returned by applying a boolean test to some aggregate quantity.

**Examples:**

```
cpu | filter avg > 80
```  

```
cpu | filter min > 20 and max < 50
```  

**filter**

Copyright 2018 Sumo Logic, Inc. All rights reserved.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description and Syntax</th>
<th>Examples</th>
</tr>
</thead>
</table>
| **min, max, avg, count, sum, pct(n), latest** | Calculates the maximum value of the time series that match the query. If grouping is specified, it calculates the maximum for each group. | **dep=prod metric=cpu_system | max**  
**cluster=search metric=cpu_idle | max by node** |
| **max** | Calculates the maximum value of the time series that match the query. If grouping is specified, it calculates the maximum for each group. | **dep=prod metric=cpu_system | max by node** |
| **min** | Calculates the minimum value of the time series that match the query. If grouping is specified, it calculates the minimum for each group. | **dep=prod metric=cpu_system | min by node** |
| **parse** | Parses the given field to create new fields to use in the metrics query. If no field is specified while parsing Graphite metrics, the metric name is used. Each wildcard in the pattern corresponds to a specified field. The parse operator supports both lazy (shortest match) and greedy (longest match) wildcard matches. Use '*' for a lazy match, or '**' for a greedy match. | **dep=prod | parse *-search-* as deployment, instance**  
**cluster=frontend | parse field=user **-* as user_id, user_type****|
| **pct** | Calculates the specified percentile of the metrics that match the query. If grouping is specified, it calculates the specified percentile for each group. | **dep=prod metric=cpu_system | pct(95)**  
**cluster=search metric=cpu_idle | pct(99.9) by node** |
| **quantize** | Segregates time series data by time period. This allows you to create aggregated results in buckets of fixed intervals (for example, 5-minute intervals). | **_sourceCategory=hostmetrics | quantize to 5m**  
**logins | quantize to 5m using sum** |
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description and Syntax</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantize to INTERVAL [using ROLLUP]</td>
<td>where ROLLUP is avg, min, max, sum or count. For information about quantization, see Metric Quantization.</td>
<td>Computes a rate based on the forward difference at each time in the time series. The difference between the current and the next recorded value in a time series is scaled to a value per second.</td>
</tr>
<tr>
<td>rate</td>
<td>This operator also assigns the value of the metric tag to be rate($metric) and the value of the unit metadata field to be $unit/second.</td>
<td>metric=Net_InBytes Interface=eth0</td>
</tr>
<tr>
<td>sum [by FIELD [, FIELD, ...]]</td>
<td>Shifts the time series from your metrics query by the specified amount of time. This can help when comparing a time series across multiple time periods.</td>
<td>dep=prod metric=cpu_system</td>
</tr>
<tr>
<td>timeshift TIME_INTERVAL</td>
<td>Select the top specified time series sorted by the value of a mathematical expression evaluated over the query time range.</td>
<td>cluster=search metric=cpu_idle</td>
</tr>
<tr>
<td>topk (number, aggregator)</td>
<td>Supported aggregate functions:</td>
<td>Take the top 10 time series with the highest maximum value:</td>
</tr>
</tbody>
</table>

Supported aggregate functions:

- min, max, avg, count, sum
- pct(n), latest

Reduce each time series by calculating (max / avg * 2) for it. Sort by this reduced value and take the top 10 values: