|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithms** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| anomalies() | Overlay a gray band on the metric showing the expected behavior of a series based on past. | not supported | anomalies(avg:system.load.1{\*}, 'robust', 2) | not supported |
| Outliers() | Highlight outliers series. | not supported | outliers(avg:system.load.1{\*}, 'DBSCAN', 3) | not supported |
| Forecast  linear | Predicts where a metric is heading in the future. | Partially equivalent, predictlinear not support DEVIATIONS variable | forecast(avg:system.disk.in\_use{\*}, 'linear', 1) | SELECT predictlinear(cpuUserPercent,1 hour) from SystemSample TIMESERIES |
| Forecast  seasonal | Predicts where a metric is heading in the future. for metrics with repeating patterns | not supported | forecast(avg:system.disk.in\_use{\*}, 'seasonal', 1) | not supported |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Arithmetic** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| abs() | Graph the absolute value of the metric | abs(average(cpuUserPercent)) | abs(system.load.1{\*}) | SELECT abs(average(cpuUserPercent) ) from SystemSample |
| log2() | graph the Base-2 logarithm of the metric. | log2(average(cpuUserPercent)) | log2(system.load.1{\*}) | SELECT log2(average(cpuUserPercent)) from SystemSample timeseries |
| log10() | Graph the Base-10 logarithm of the metric. | Log10(average(cpuUserPercent)) | log10(system.load.1{\*}) | SELECT log10(average(cpuUserPercent)) from SystemSample timeseries |
| cumsum() | Graph the cumulative sum of the metric over the visible time window. | not supported | cumsum(system.load.1{\*}) | not supported |
| Integral() | Graph the integral of the metric | not supported | integral(system.load.1{\*}) | not supported |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Count** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| count\_nonzero() | Compute count of all non-zero values | Where condition != 0 in nrql query | count\_nonzero(system.load.1{\*}) | SELECT count(cpuUserPercent) FROM SystemSample WHERE cpuUserPercent != 0 TIMESERIES |
| count\_not\_null() | Compute count of all not null values. | Where condition is not NULL in nrql query | count\_not\_null(system.load.1{\*}) | SELECT count(cpuUserPercent) FROM SystemSample WHERE cpuUserPercent is NOT NULL TIMESERIES |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Exclusion** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| exclude\_null() | Remove N/A groups from your graph or top list | WHERE condition IS NOT NULL | exclude\_null(avg:system.load.1{\*} by {host}) | SELECT cpuUserPercent from SystemSample WHERE cpuUserPercent is NOT NULL |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Interpolation** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| .fill(null) | Interpolate missing metric values for the metric. | not supported | system.load.1{\*}.fill(null) | not supported |
| .fill(zero) | Fill missing points with zero. | not supported | system.load.1{\*}.fill(zero) | not supported |
| .fill(linear) | Gives you a linear interpolation between the beginning and the end of the gap. | not supported | system.load.1{\*}.fill(linear) | not supported |
| .fill(last) | Fills the gap with the last value of the gap | not supported | system.load.1{\*}.fill(last) | not supported |
| default\_zero() | Adds a default value to sparse metrics. | not supported | default\_zero(system.load.1{\*}) | not supported |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rank** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| top() | Graph the top N elements | does not apply | top(<METRIC\_NAME>{\*}, <LIMIT\_TO>, '<BY>', '<DIR>') | does not apply |
| **Variants between by and dir parameter** | | | | |
| top(metric\_name,10,mean,asc) | * Mean of all metrics values. * Rank the results in ascending order * The number of series to be displayed is 10 | ORDER BY, ASC, LIMIT clause and  MEDIAN function | top(system.load.1{\*}, 10, 'mean', 'asc') | SELECT median(cpuUserPercent) from SystemSample order by 'cpuUserPercent' asc limit 10 timeseries |
| top(metric\_name,10,mean,desc) | * Mean of all metrics values. * Rank the results in descending order * The number of series to be displayed is 10 | ORDER BY, DESC, LIMIT clause and  MEDIAN function | top(system.load.1{\*}, 10, 'mean', 'desc') | SELECT median(cpuUserPercent) from SystemSample order by 'cpuUserPercent' desc limit 10 timeseries |
| top(metric\_name,10,max,asc) | * maximum of all metrics values. * Rank the results in ascending order * The number of series to be displayed is 10 | ORDER BY, ASC, LIMIT clause and  MAX function | top(system.load.1{\*}, 10, 'max', 'asc') | SELECT max(cpuUserPercent) from SystemSample order by 'cpuUserPercent' asc limit 10 timeseries |
| top(metric\_name,10,max,desc) | * maximum of all metrics values. * Rank the results in descending order * The number of series to be displayed is 10 | ORDER BY, DESC ,LIMIT clause and  MAX function | top(system.load.1{\*}, 10, 'max', 'desc') | SELECT max(cpuUserPercent) from SystemSample order by 'cpuUserPercent' desc limit 10 timeseries |
| top(metric\_name,10,min,asc) | * Min of all metrics values. * Rank the results in ascending order * The number of series to be displayed is 10 | ORDER BY, DESC ,LIMIT clause and  MIN function | top(system.load.1{\*}, 10, 'min', 'asc') | SELECT min(cpuUserPercent) from SystemSample order by 'cpuUserPercent' asc limit 10 timeseries |
| top(metric\_name,10,min,desc) | * Min of all metrics values. * Rank the results in descending order * The number of series to be displayed is 10 | ORDER BY, DESC ,LIMIT clause and  MIN function | top(system.load.1{\*}, 10, 'min', 'desc') | SELECT min(cpuUserPercent) from SystemSample order by 'cpuUserPercent' desc limit 10 timeseries |
| top(metric\_name,10,sum,asc) | * Sum of all metrics values. * Rank the results in ascending order * The number of series to be displayed is 10 | ORDER BY, DESC ,LIMIT clause and  SUM function | top(system.load.1{\*}, 10, 'sum', 'asc') | SELECT sum(cpuUserPercent) from SystemSample order by 'cpuUserPercent' asc limit 10 timeseries |
| top(metric\_name,10,sum,desc) | * Sum of all metrics values. * Rank the results in descending order * The number of series to be displayed is 10 | ORDER BY, DESC ,LIMIT clause and  SUM function | top(system.load.1{\*}, 10, 'sum', 'desc') | SELECT sum(cpuUserPercent) from SystemSample order by 'cpuUserPercent' desc limit 10 timeseries |
| top(metric\_name,10,last,asc) | * Last metrics value. * Rank the results in ascending order * The number of series to be displayed is 10 | ORDER BY, ASC ,LIMIT clause and  LATEST function | top(system.load.1{\*}, 100, 'last', 'asc') | SELECT latest(cpuUserPercent) from SystemSample order by 'cpuUserPercent' asc limit 10 timeseries |
| top(metric\_name,10,last,asc) | * Last metrics value. * Rank the results in descending order * The number of series to be displayed is 10 | ORDER BY, DESC ,LIMIT clause and  LATEST function | top(system.load.1{\*}, 100, 'last', 'desc') | SELECT latest(cpuUserPercent) from SystemSample order by 'cpuUserPercent' desc limit 10 timeseries |
| top(metric\_name,10,l2norm,asc) | * Uses the norm of the timeseries, which is always positive, to rank the series * Rank the results in ascending order * The number of series to be displayed is 10 | not supported | top(system.load.1{\*}, 10, ' l2norm', 'desc') | not supported |
| top(metric\_name,10,l2norm,desc) | * Uses the norm of the timeseries, which is always positive, to rank the series * Rank the results in descending order * The number of series to be displayed is 10 | not supported | top(system.load.1{\*}, 10, ' l2norm', 'asc') | not supported |
| top(metric\_name,10,area,asc) | * Signed area under the curve being graphed, which can be negative * Rank the results in ascending order * The number of series to be displayed is 10 | not supported | top(system.load.1{\*}, 10, ' area', 'asc') | not supported |
| top(metric\_name,10,area,desc) | * Signed area under the curve being graphed, which can be negative * Rank the results in descending order * The number of series to be displayed is 10 | not supported | top(system.load.1{\*}, 10, ' area', 'desc') | not supported |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rate** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| per\_second() | Graph the rate at which the metric is changing per second | rate(count(\*),1 second) | per\_second(system.load.1{\*}) | SELECT rate(count(\*),1 second) from SystemSample TIMESERIES |
| per\_minute() | Graph the rate at which the metric is changing per minute | rate(count(\*),1 minute) | per\_minute(system.load.1{\*}) | SELECT rate(count(\*),1 minute) from SystemSample TIMESERIES |
| per\_hour() | Graph the rate at which the metric is changing per minute | rate(count(\*),1 minute) | per\_hour(system.load.1{\*}) | SELECT rate(count(\*),1 hour) from SystemSample TIMESERIES |
| dt() | Graph the time difference in seconds between submitted points | not supported | dt(system.load.1{\*}) | not supported |
| diff() | Graph the delta of the metric | Delta supported of new relic return the difference of one metric between now and time range, defined it is not equivalent but it can be alternative:  delta(metric\_name[time\_range]) | diff(system.load.1{\*}) | \*\*promql traduced to nrql syntax:  SELECT (latest(`metric\_name`) - earliest(`metric\_name`))FROM Metric SINCE 60 MINUTES AGO UNTIL NOW FACET dimensions() LIMIT 100 TIMESERIES |
| monotic\_diff() | Graph the delta of the metric like diff() but only if the delta is positive | not supported | monotic\_diff(system.load.1{\*}) | not supported |
| derivative() | Graph the derivative (diff/dt) of the metric | derivative(cpuUserPercent, 1 hour) | derivative(system.load.1{\*}) | SELECT derivative(cpuUserPercent,1 second) from SystemSample TIMESERIES |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Regresion** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| robust\_trend() | Fit a robust linear regression using Huber loss. | not supported | robust\_trend(avg:system.load.1{\*}) | not supported |
| trend\_line() | Fit an ordinary least squares regression line through the metric values | predictLinear() | trend\_line(avg:system.load.1{\*}) | SELECT predictLinear(cpuUserPercent,1 hour) from SystemSample TIMESERIES 1 hour  \*\* It uses a similar method of least-squares linear regression to predict the future values for a dataset. |
| piecewise\_constant() | Approximate the metric with a piecewise function composed of constant-valued segments | not supported | piecewise\_constant(avg:system.load.1{\*}) | not supported |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Rollup** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| .rollup() | Roll up the metric by its average value over the specified time period. | TIMESERIES clause and  AVERAGE function | system.load.1{\*}.rollup(avg), system.load.1{\*}.rollup(avg, 600) | SELECT average(cpuUserPercent) FROM SystemSample TIMESERIES 120 seconds |
| .rollup(min) | Roll up the metric by its minimum value in the specified time period. | TIMESERIES clause and  MIN function | system.load.1{\*}.rollup(min) | SELECT min(cpuUserPercent) FROM SystemSample TIMESERIES 120 seconds |
| .rollup(max) | Roll up the metric by its maximum value in the specified time period. | TIMESERIES clause and  MAX function | system.load.1{\*}.rollup(max) | SELECT max(cpuUserPercent) FROM SystemSample TIMESERIES 120 seconds |
| .rollup(sum) | Roll up the metric by the sum of values in the specified time period. | TIMESERIES clause and  SUM function | system.load.1{\*}.rollup(sum) | SELECT sum(cpuUserPercent) FROM SystemSample TIMESERIES 120 seconds |
| .rollup(count) | Roll up the metric by its count in the specified time period. | TIMESERIES clause and  COUNT function | system.load.1{\*}.rollup(count) | SELECT count(cpuUserPercent) FROM SystemSample TIMESERIES 120 seconds |
| moving\_rollup() | Rollup to combine the points in the last X seconds. | not supported | moving\_rollup(avg:system.load.1{\*}, 300 , sum) | not supported |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Smoothing** | | | | | | | | |
| **Function datadog** | **Description** | | **Equivalent newrelic** | | **e.g datadog** | | **e.g new relic** | |
| autosmooth() | Automatically removes noise while preserving the trend of the metric. | | SLIDE BY clause and AVERAGE function | | autosmooth(avg:system.load.1{\*} by {host}) | | SELECT average(cpuUserPercent) FROM SystemSample TIMESERIES 5 minute SLIDE BY 1 minute | |
| ewma\_3() | Compute the exponentially weighted moving average over a span of 3. | | not supported | | ewma\_3(avg:system.load.1{\*} by {host}) | | not supported | |
| ewma\_5() | Compute the exponentially weighted moving average over a span of 5. | | not supported | | ewma\_5(avg:system.load.1{\*} by {host}) | | not supported | |
| ewma\_10() | Compute the exponentially weighted moving average over a span of 10. | | not supported | | ewma\_10(avg:system.load.1{\*} by {host}) | | not supported | |
| ewma\_20() | Compute the exponentially weighted moving average over a span of 20. | | not supported | | ewma\_20(avg:system.load.1{\*} by {host}) | | not supported | |
| median\_3() | Rolling median with a span of 3. | | not supported | | median\_3(avg:system.load.1{\*} by {host}) | | not supported | |
| median\_5() | Rolling median with a span of 5. | | not supported | | median\_5(avg:system.load.1{\*} by {host}) | | not supported | |
| median\_7() | Rolling median with a span of 7. | | not supported | | median\_7(avg:system.load.1{\*} by {host}) | | not supported | |
| median\_9() | Rolling median with a span of 9. | | not supported | | median\_9(avg:system.load.1{\*} by {host}) | | not supported | |
| **Timeshift** | | | | | | | | |
| **Function datadog** | | **Description** | | **Equivalent newrelic** | | **e.g datadog** | | **e.g new relic** |
| timeshift() | | Graph values from an arbitrary <TIME\_IN\_SECOND> before the current timestamp for the metric | | SINCE clause and  AVERAGE function | | timeshift(avg:system.load.1{\*}, -1209600) | | SELECT average(cpuUserPercent) from SystemSample since 1209600 seconds ago timeseries |
| hour\_before() | | Graph values from an hour before the current timestamp for the metric. | | SINCE clause and  AVERAGE function | | hour\_before(system.load.1{\*}) | | SELECT average(cpuUserPercent) from SystemSample since 1 hour ago timeseries |
| day\_before() | | Graph values from a day before the current timestamp for the metric. | | SINCE clause and  AVERAGE function | | day\_before(system.load.1{\*}) | | SELECT average(cpuUserPercent) from SystemSample since 1 day ago timeseries |
| week\_before() | | Graph values from a week (7 days) before the current timestamp for the metric. | | SINCE clause and  AVERAGE function | | week\_before(system.load.1{\*}) | | SELECT average(cpuUserPercent) from SystemSample since 1 week ago timeseries |
| month\_before() | | Graph values from a month (28 days / 4 weeks) before the current timestamp for the metric. | | SINCE clause and  AVERAGE function | | month\_before(system.load.1{\*}) | | SELECT average(cpuUserPercent) from SystemSample since 1 month ago timeseries |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Beta** | | | | |
| **Function datadog** | **Description** | **Equivalent newrelic** | **e.g datadog** | **e.g new relic** |
| rollingavg\_5() | Compute the rolling average over a span of 5. | not supported | rollingavg\_5(system.load.1{\*}) | not supported |
| rollingavg\_13() | Compute the rolling average over a span of 13. | not supported | rollingavg\_13(system.load.1{\*}) | not supported |
| rollingavg\_21() | Compute the rolling average over a span of 21. | not supported | rollingavg\_21(system.load.1{\*}) | not supported |
| rollingavg\_29() | Compute the rolling average over a span of 29. | not supported | rollingavg\_29(system.load.1{\*}) | not supported |

|  |  |
| --- | --- |
| **Additional observations** | |
| * Not support multiple querys in new relic and datadog support multiples querys and differents groups * Datadog has open canvas where we can put a widget anywhere * “multiple query” not supported for newrelic | |
| **Total functions of datadog:** | 66 |
| **Total functions of datadog not supported in new relic** | 31 |