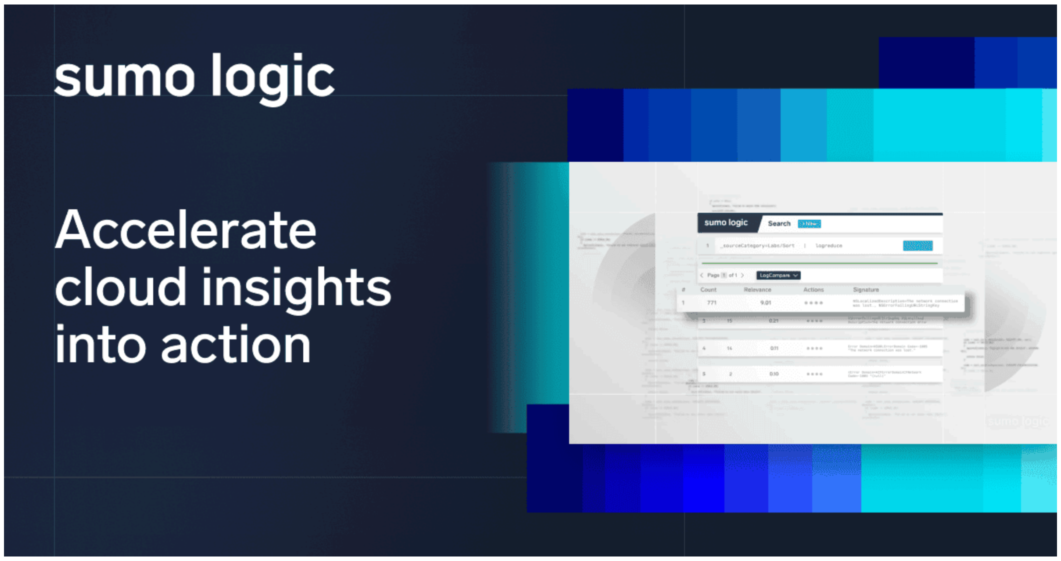
Log Operators Cheat Sheet | Sumo Logic Docs

sumologic.com/help/docs/search/search-cheat-sheets/log-operators

October 10, 2025



The Log Operators cheat sheet provides a list of available parsers, aggregators, search operators, and mathematical expressions with links to full details for each item. For a complete list of Sumo Logic Search operators, download the PDF version.

The following tables provide a list of available Sumo Logic parsers, aggregators, search operators, and mathematical expressions.

Parsing

Sumo provides a number of ways to parse fields in your log messages.

Operator	Description	Example
parse (anchor)	The parse operator, also called parse anchor, parses strings according to specified start and stop anchors, and then labels them as fields for use in subsequent aggregation functions in the query such as sorting, grouping, or other functions.	`| parse "User=*:" as user`
parse regex	The parse regex operator (also called the extract operator) enables users comfortable with regular expression syntax to extract more complex data from log lines. Parse regex can be used, for example, to extract nested fields.	`| parse regex field=url "[0-9A-Za-z-]+.(?<domain>[A-Za-z-]+.(?:co.uk|com|com.au))/.*"`
keyvalue	Typically, log files contain information that follow a key-value pair structure. The keyvalue operator allows you to get values from a log message by specifying the key paired with each value.	`| keyvalue "module", "thread"`
csv	The csv operator allows you to parse Comma Separated Values (CSV) formatted log entries. It uses a comma as the default delimiter.csv operator allows you to parse Comma Separated Values (CSV) formatted log entries. It uses a comma as the default delimiter.	`| csv_raw extract 1 as user, 2 as id, 3 as name`

JSON	The JSON operator is a search query language operator that allows you to extract values from JSON input. Because JSON supports both nested keys and arrays that contain ordered sequences of values, the Sumo Logic JSON operator allows you to extract single top-level fields, multiple fields, nested keys, and keys in arrays.	`| parse "explainJsonPlan] *" as jsonobject`
		`| json field=jsonobject "sessionId"`
		`| json auto`

split	The split operator allows you to split strings into multiple strings, and parse delimited log entries, such as space-delimited formats.	Full query example:
		`_sourceCategory=colon`
		`| parse "] * *" as log_level, text`
		`| split text delim=':' extract 1 as user, 2 as account_id, 3 as session_id, 4 as result`

xml	The XML operator uses a subset of the XPath 1.0 specification to provide a way for you to parse fields from XML documents. Using it, you can specify what to extract from an XML document using an XPath reference.	`| parse xml "/af/minimum/@requested_bytes"`

Aggregating

Aggregating functions evaluate messages and place them into groups. The group operator is used in conjunction with group-by functions. When using any grouping function, the word by is sufficient for representing the group operator.

ⓘnote
An aggregation function cannot take another function (such as a math function). For example, you cannot use:

```
... | avg(x + y) as average
```

Instead, use separate steps:

```
... | x + y as z | avg(z) as average
```

Operator	Description	Default Alias	Restrictions	Example
avg	The averaging function (avg) calculates the average value of the numerical field being evaluated within the time range analyzed.	_avg		`| avg(request_received) by _timeslice`

count, count_distinct, and count_frequent	Aggregating (group-by) functions are used in conjunction with the group operator and a field name. Only the word by is required to represent the group operator. The count function is also an operator in its own right and therefore can be used with or without the word by.	_count
		_count_distinct
		_approxcount
			count_frequent can return up to 100 results when used in dashboard panels.	Example 1:
				`| count by url`
				
				Example 2:
				`| count_distinct(referrer) by status_code`

fillmissing	When you run a standard group-by query, Sumo Logic only returns non-empty groups in the results. For example, if you are grouping by timeslice, then only the timeslices that have data are returned.
	This operator allows you to specify groups to present in the output, even if those groups have no data.
			Not supported in Auto Refresh Dashboards or any continuous query.	`error`
				`| count by _sourceCategory`
				`| fillmissing values("backend", "database", "webapp") in _sourceCategory`

first and last	First finds the earliest occurrence in search results, and last finds the result that follows all others, based on the sort order for the query.	_first
		_last
			Not supported in auto refresh dashboards or any continuous query.	`| sort by _timeslice`
				`| first(error_message) by hostname`

min and max	Use the min and max functions to find the smallest or largest value in a set of values.	_min _max		`| max(request_received) by hour`

most_recent and least_recent	The most_recent and least_recent operators, used with the withtime operator, allow you to order data from newest to oldest.	_most_recent
		_least_recent
				`ip OR address`
				`| parse regex "(?<IP>\b\d3.\d3.\d3.\d3)"`
				`| lookup latitude, longitude, country_code from geo://location on ip=IP`
				`| where !isNull(country_code)`
				`| withtime IP`
				`| most_recent(ip_withtime) by country_code`

pct	The percentile function (pct) finds the percentile of a given field. Multiple pct functions can be included in one query.	_<fieldname>*pct*<percentile>	`| parse "value=*" as value`
			`| pct(value, 95) as value_95pct`

stddev	The standard deviation function (stddev) finds the standard deviation value for a distribution of numerical values within the time range analyzed and associated with a group designated by the "group by" field.	_stddev	`... | stddev(request_received) group by hour | sort by _stddev`
sum	Sum adds the values of the numerical field being evaluated within the time range analyzed.	_sum	`... | sum(bytes_received) group by hostname`

Search Operators

This section provides detailed syntax, rules, and examples for Sumo Logic Operators, Expressions, and Search Language.

Operator	Description	Default Alias	Restrictions	Example

accum	The accum operator calculates the cumulative sum of a field. It can be used to find a count by a specific time interval, and can be used to find a total running count across all intervals.	_accum	Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`_sourceCategory=IIS (Wyatt OR Luke)`
				`| parse "[user=*]" as cs_username`
				`| timeslice by 1m`
				`| count as requests by _timeslice,cs_username`
				`| sort by _timeslice asc,cs_username`
				`| accum requests as running_total`

asn lookup	Sumo Logic can lookup an Autonomous System Number (ASN) and organization name by an IP address. Any IP addresses that do not have an ASN will return null values.			`_sourceCategory=stream "remote_ip="`
				`| parse regex "(?<ip>\d3.\d3.\d3.\d3)"`
				`| lookup organization, asn from asn://default on ip = ip`

backshift	The backshift operator compares values as they change over time. Backshift can be used with rollingstd, smooth, or any other operators whose results could be affected by spikes of data (where a spike could possibly throw off future results).	_backshift	Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`_sourceCategory=katta`
				`| timeslice by 1m`
				`| count by _timeslice,_sourcehost`
				`| sort + _timeslice`
				`| backshift _count,1 by _sourcehost`

base64Decode	The base64Decode operator takes a base64 string and converts it to an ASCII string.			`| base64Decode("aHR0cDovL2NvZGVjLmFwYWNoZS5vcmcvY29tbW1vbnM=") as V`
base64Encode	The base64Encode operator takes an ASCII string and converts it to a base64 string.			`| base64Encode("hello world") as base64`

bin	Use the bin operator to sort results in a histogram.	_bin_label
		_bin_lower
		_bin_upper
				`_sourceCategory=analytics`
				`| parse "ms: *" as time`
				`| bin time width=10, min = 0, max = 500`
				`| count by _bin, _bin_upper`
				`| sort by _bin_upper`

CIDR	The CIDR operator allows you to leverage Classless Inter-Domain Routing (CIDS) notations to analyze IP network traffic in order to narrow analysis to specific subnets. CIDR notations specify the routing prefix of IP addresses.			`(denied OR rejected AND _sourcecategory=firewall`
				`| parse "ip=*," as ip_address`
				`| where compareCIDRPrefix("10.10.1.32", ip_address, toInt(27))`
				`| count by ip_address`

concat	The Concat operator allows you to concatenate or join multiple strings, numbers, and fields into a single user-defined field. It concatenates strings end-to-end and joins them into a new string that you define.		Not supported in Dashboards.	`... | concat(octet1, ".", octet2, ".",octet3, ".",octet4) as ip_address`
contains	The contains operator compares string values of two parsed fields and returns a boolean result based on whether the second field's value exists in the first.			`... | contains("hello world", "hello") as containing`
decToHex	The decToHex operator converts a long value of 16 or fewer digits to a hexadecimal string using Two's Complement for negative values.			`... | decToHex("4919") as V`

diff	The diff operator calculates the rate of change in a field between consecutive rows. To produce results, diff requires that a specified field contain numeric data; any non-numerical values are removed from the search results.	_diff	Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`* | parse "bytes transmitted: '*'" as bytes`
				`| timeslice 1m`
				`| sum(bytes) as bytes by _timeslice`
				`| sort _timeslice`
				`| diff bytes as diff_bytes`

fields	The fields operator allows you to choose which fields are displayed in the results of a query. Use a fields operator to reduce the "clutter" of a search output that contains fields that aren't completely relevant to your query.		`_sourceCategory=access_logs`
			`| parse "[status=*]" as status_code`
			`| fields method, status_code`

filter	The filter operator can filter the output of a search using the results of a different search based on the filtering criteria of a subquery. The filter operator keeps only the records that match the filter criteria, allowing you to restrict search results to the most relevant information.	The operator can process up to 100,000 data points for a single query. It automatically drops the data points that exceed the limit and issues a warning.	`_sourceCategory=HttpServers`
			`| timeslice 1m`
			`| count by _timeslice, _sourceHost`
			`| filter _sourcehost in (outlier _count by _sourceHost | where _count_violation > 0)`
			`| transpose row _timeslice column _sourcehost`

format	The format operator allows you to format and combine data from fields in message logs—including numbers, strings, and dates—into a single user-defined string. This allows data in message logs, such as dates or currency amounts, to be formatted as human readable, when otherwise it would be hard to decipher.		`error`
			`| parse "fiveMinuteRate=*," as rate`
			`| format("%s : %s","Five Minute Rate is" , rate) as formattedVal`

formatDate	The formatDate operator allows you to format dates in log files as a string in the format you require, such as US date formatting, European formatting, timestamps, etc.		`* | formatDate(now(), "yyyy-MM-dd") as today`

geo lookup	Sumo Logic can match a parsed IPv4 or IPv6 address to its geographical location on a map. To create the map the lookup operator matches parsed IP addresses to their physical location based on the latitude and longitude of where the addresses originated.	latitude
		longitude
		_count
		continent
		country_code
		country_name
		region
		city
		state
		postal_code
		connection_type
		country_cf
		state_cf
		city_cf
			`| parse "remote_ip=*]" as remote_ip`
			`| lookup latitude, longitude, country_code, country_name, region, city, postal_code from geo://location on ip = remote_ip`
			`| count by latitude, longitude, country_code, country_name, region, city, postal_code`
			`| sort _count`

haversine	The haversine operator returns the distance between latitude and longitude values of two coordinates in kilometers. Coordinates need to be positive or negative values based on being north/south or east/west, instead of using the terms N/S, E/W.		`| haversine(39.04380, -77.48790, 45.73723, -119.81143) as distanceKMs`
hexToDec	The hexToDec operator converts a hexadecimal string of 16 or fewer characters to long using Two's Complement for negative values.		`| hexToDec("0000000000001337") as V`

if	There are two forms of ternary expression you can use in Sumo Logic queries: one is constructed using the IF operator, and the other uses the question mark (?) operator. These expressions are used to evaluate a condition as either true or false, with values assigned for each outcome. It is a shorthand way to express an if-else condition.	`| if(status_code matches "5*", 1, 0) as server_error`
		`Or`
		`| status_code matches "5*" ? 1 : 0 as server_error`

in	The In operator returns a Boolean value: true if the specified property is in the specified object, or false if it is not.	`| if (status_code in ("500", "501", "502", "503", "504", "505", "506", "401", "402", "403", "404"), "Error", "OK") as status_code_type`

ipv4ToNumber	The ipv4ToNumber operator allows you to convert an Internet Protocol version 4 (IPv4) IP address from the octet dot-decimal format to a decimal format. This decimal format makes it easier to compare one IP address to another, rather than relying on IP masking.	`_sourceCategory=service remote_ip`
		`| parse "[remote_ip=*]" as ip`
		`| ipv4ToNumber(ip) as num`
		`| fields ip, num`

isBlank	The isBlank operator checks to see that a string contains text. Specifically, it checks to see if a character sequence is whitespace, empty ("") ,or null. It takes a single parameter and returns a Boolean value: true if the variable is indeed blank, or false if the variable contains a value other than whitespace, empty, or null.	`| where isBlank(user)`
isEmpty	The isEmpty operator checks to see that a string contains text. Specifically, it checks to see whether a character sequence is empty ("") or null. It takes a single parameter and return a Boolean value: true if the variable is indeed empty, or false if the variable contains a value other than empty or null.	`| if(isEmpty(src_ip),1,0) as null_ip_counts`
isNull	The isNull operator takes a single parameter and returns a Boolean value: True if the variable is indeed null, or false if the variable contains a value other than null.	`| where isNull(src_ip)`

isNumeric	The isNumeric operator checks whether a string is a valid Java number.		`| isNumeric(num)`
isPrivateIP	The isPrivateIP operator checks if an IPv4 address is private and returns a boolean.		`| isPrivateIP(hostip)`
isPublicIP	The isPublicIP operator checks if an IPv4 address is public and returns a boolean.		`| isPublicIP("10.255.255.255") as isPublic`
isValidIP	The isValidIP operator checks if the value is a valid IP address. The isValidIPv4 and isValidIPv6 operators check if the value is a valid IPv4 or IPv6 address respectively.		`| isValidIP("10.255.255.255") as isIP`

join	The join operator combines records of two or more data streams. Results are admitted on-the-fly to allow real time tables to be built. Values common to each table are then delivered as search results.	Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	Full query example:
			`("starting stream from" OR "starting search")`
			`| join`
			`(parse "starting stream from *" AS a) AS T1,`
			`(parse "starting search * from parent stream *" AS b, c) AS T2`
			`on T1.a = T2.c`

length	The length operator returns the number of characters in a string. You can use it in where clauses or to create new fields. It returns 0 if the string is null.		`| where length(query) <= 20`

limit	The limit operator reduces the number of raw messages or aggregate results returned. If you simply query for a particular term, for example "error" without using an aggregation operator such as group by, limit will reduce the number of raw messages returned. If you first use group-by or other aggregation operator, the limit operator will reduce the number of grouped results instead.		Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`| count by _sourceCategory`
				`| sort by _count`
				`| limit 5`

logcompare	The logcompare operator allows you to compare two sets of logs: baseline (historical) and target (current). To run a LogCompare operation, you can use the **LogCompare** button on the **Messages** tab to generate a properly formatted query.	_count _deltaPercentage _anomalyScore _isNew	Not supported in Dashboards.	`| logcompare timeshift -24h`

logexplain	The logexplain operator allows you to compare sets of structured logs based on events you're interested in. Structured logs can be in JSON, CSV, key-value, or any structured format.	_explanation
		_relevance
		_test_coverage
		_control_coverage
			Time Compare and the compare operator are not supported against LogExplain results.	`_sourceCategory=stream`
				`| if(_raw matches "error", 1, 0) as hasError`
				`| logexplain hasError == 1 on _sourceHost`

logreduce	The LogReduce algorithm uses fuzzy logic to cluster messages together based on string and pattern similarity. Use the **LogReduce** button and operator to quickly assess activity patterns for things like a range of devices or traffic on a website. (Formerly Summarize.)		Not supported in Dashboards.	`| logreduce`

logreduce keys	The logreduce keys operator allows you to quickly explore JSON or key-value formatted logs by schemas.	_signature_id
		_schema
		_count
				`_sourcecategory="Labs/AWS/GuardDuty_V8"`
				`| json keys "region", "partition", "resource"`
				`| logreduce keys field=resource`

logreduce values	The logreduce values operator allows you to quickly explore structured logs by known keys. Structured logs can be in JSON, CSV, key-value, or any structured format.	_cluster_id
		_signature
		_count
				`_sourceCategory= cloudtrail errorCode`
				`| json field=_raw "eventSource" as eventSource`
				`| json field=_raw "eventName" as eventName`
				`| json field=_raw "errorCode" as errorCode`
				`| logreduce values on eventSource, eventName, errorCode`

lookup	Using a lookup operator, you can map data in your log messages to meaningful information. For example, you could use a lookup operator to map "userID" to a real user's name. Or, you could use a lookup operator to find black-listed IP addresses.			`| parse "name=, phone number=," as (name, phone)`
				`| count by name, phone`
				`//We recommend doing a lookup after an aggregation`
				`| lookup email from https://compay.com/userTable.csv on name=userName, phone=cell`

luhn (credit card validator)	The Luhn operator uses Luhn's algorithm to check message logs for strings of numbers that may be credit card numbers, and then validates them. It takes a string as an input, strips out all characters that are not numerals, and checks if the resulting string is a valid credit card number, returning true or false accordingly.	`| parse regex "(?<maybecc>\d4-\d4-\d4-\d4)" nodrop`
		`| parse regex "(?<maybecc>\d4\s\d4\s\d4\s\d4)" nodrop`
		`| parse regex "(?<maybecc>\d16)" nodrop`
		`| if (luhn(maybecc), true, false) as valid`

matches	The matches operator can be used to match a string to a wildcard pattern or an RE2 compliant regex. The return of the operator is Boolean; the operator can be used with where or if expressions.	`| if (agent matches "MSIE","Internet Explorer","Other") as Browser`
		`| if (agent matches "Firefox","Firefox",Browser) as Browser`

median	In order to calculate the median value for a particular field, you can utilize the Percentile (pct) operator with a percentile argument of 50.	`| parse "value=*" as value`
		`| pct(value, 50) as median`

merge	The merge operator reduces a stream of events to a single event using a specified merge strategy. It is particularly useful as a subquery for the Transactionize operator.	`| parse "BytesSentPersec = "*"" as BytesPersec`
		`| merge BytesPersec join with "--", _messageTime takeLast`

[now](#)	The now operator returns the current epoch time in milliseconds. It can be used with the [formatDate](#) operator to get the formatted current time.		Can be used in Dashboard Panels, but the `now()` time presented in Live mode (the time the data is processed) doesn't match the search time, so the results are different. The results for search could be hours or days later than the time presented in Live mode.	`| now() as current_date`

[num](#)	The num operator converts a field to a number. Using Num in a query is useful for sorting results by number instead of alphabetically, which is the default. You can also use double as the operator, as an alias equivalent, if you prefer.			`| parse "Execution duration: * s" as duration`
				`| num(duration)`
				`| sort by duration`

[outlier](#)	Given a series of time-stamped numerical values, using the outlier operator in a query can identify values in a sequence that seem unexpected, and would identify an alert or violation, for example, for a scheduled search.	<field>_error
		<field>_lower
		<field>_upper
		<field>_indicator
		<field>_violation
				Full query example:
				`_sourceCategory=IIS/Access`
				`| parse regex "\d+-\d+-\d+ \d+:\d+:\d+ (?<server_ip>\S+) (?<method>\S+) (?<cs_uri_stem>/\S+?) \S+ \d+ (?<user>\S+) (?<client_ip>[.\d]+) "`
				`| parse regex "\d+ \d+ \d+ (?<response_time>\d+)$"`
				`| timeslice 1m`
				`| max(response_time) as response_time by _timeslice`
				`| outlier response_time window=5,threshold=3,consecutive=2,direction=+-`

[parseHex](#)	The parseHex operator allows you to convert a hexadecimal string of 16 or fewer characters to a number.			`| parseHex("12D230") as decimalValue`

[predict]()	The predict operator uses a series of time stamped numerical values to predict future values. For example, you could use this operator to take your current disk space capacity numbers, and predict when your system might run out of disk space.	*<agg field>* <agg field>*predicted* <agg field>*error* <agg field>_linear		Full query example: `_sourceCategory=taskmanager` `| jobState=InQueue` `| timeslice 1m` `| count by _timeslice` `| toDouble(_count)` `| predict _count by 1m forecast=5`
[replace]()	The replace operator allows you to replace all instances of a specified string with another string. You can specify the string to replace with a matching regex or literal text. You might use it to find all instances of a name and change it to a new name or to replace punctuation in a field with different punctuation. This operator is useful anytime you need to rename something.			`| replace(query, ".","->") as query`
[rollingstd]()	The rollingstd (rolling standard) operator provides the rolling standard deviation of a field over a defined window. Rollingstd displays this value in a new column named _rollingstd.	_rollingstd	Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`| rollingstd _count,1 by _sourcehost`

save	Using the Save operator allows you to save the results of a query into the Sumo Logic file system. Later, you can use the lookup operator to access the saved data. The Save operator saves data in a simple format to a location you choose.		Not supported in Dashboards.	`| save /shared/lookups/daily_users`
sessionize	The sessionize operator allows you to use an extracted value from one log message (generated from one system) to find correlating values in log messages from other systems. After you run Sessionize, these related events are displayed on the same page. The thread of logs woven together is called a session.		Not supported in auto refresh dashboards or any continuous query.	Full query example: `(SearchServiceImpl Creating Query) or (Stream SessionId using searchSessionId) or (Started search with sessionId) | sessionize "session: '', streamSessionID: ''" as (serviceSessionId, streamSessionId), "Stream SessionId=$streamSessionId using searchSessionId=* and rawSessionId=*" as (searchSessionId, rawSessionId), "Started search with sessionId: $searchSessionId, customerId: *, query: *" as (customerId, query)`
smooth	The smooth operator calculates the rolling (or moving) average of a field, measuring the average of a value to "smooth" random variation. Smooth operator reveals trends in the data set you include in a query.	_smooth	Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`| smooth _count,1 by _sourcehost`

sort	The sort operator orders aggregated search results. The default sort order is descending. Then you can use the top or limit operators to reduce the number of sorted results returned.		Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`| count as page_hits by _sourceHost`
				`| sort by page_hits asc`

substring	The substring operator allows you to specify an offset that will output only part of a string, referred to as a substring. You can use this operator to output just a part of a string instead of the whole string, for example, if you wanted to output an employee's initials instead of their whole name.			`| substring("Hello world!", 6)`

timeslice	The timeslice operator segregates data by time period, so you can create bucketed results based on a fixed width in time, for example, five minute periods. Timeslice also supports bucketing by a fixed number of buckets across the search results, for example, 150 buckets over the last 60 minutes. An alias for the timeslice field is optional. When an alias is not provided, a default _timeslice field is created.	_timeslice	Timeslices greater than 1 day cannot be used in Dashboard Live mode.	`| timeslice 1h`
				`//You can further aggregate your data by these time groupings`
				`| count by _timeslice`

toLowerCase and toUpperCase	As the name implies, the toLowerCase operator takes a string and converts it to all lower case letters. The toUpperCase operator takes a string and converts it to all upper case letters.			`| toUpperCase(_sourceHost) as _sourceHost`
				`| where _sourceHost matches "NITE"`

topk	Select the top values from fields and group them by other fields.	_rank		`| topk(5, _count)`
top	Use the top operator with the sort operator, to reduce the number of sorted results returned.		Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`| top 5 _sourcecategory`
total	The total operator calculates the grand total of a field and injects that value into every row. It also supports grouping rows by a set of fields.	_total	Can be used in Dashboard Panels, but in the search they must be included after the first `group-by` phrase.	`| total gbytes as total_memory`
trace	A trace operator acts as a highly sophisticated filter to connect the dots across different log messages. You can use any identifying value with a trace operator (such as a user ID, IP address, session ID, etc.) to retrieve a comprehensive set of activity associated to that original ID.		Not supported in Auto Refresh Dashboards or any continuous query.	`| trace "ID=( [0-9a-fA-F] 4 )" "7F92"`

transaction	The transaction operator is used to analyze related sequences of logs. No matter what type of data you're analyzing, from tracking web site sign ups, to e-commerce data, to watching system activity across a distributed system, the transaction operator can be used in a variety of use cases.	_start_time
		_end_time
			Tables generated with unordered data can be added to Dashboards, but Flow Diagrams cannot be added to Dashboards. Transaction by flow cannot be used with Dashboards.	`| transaction on sessionid fringe=10m`
				`with "Starting session *" as init,`
				`with "Initiating countdown *" as countdown_start,`
				`with "Countdown reached *" as countdown_done,`
				`with "Launch *" as launch`
				`results by transaction`

transactionize	The transactionize operator groups logs that match on any fields you specify. Unlike other "group by" operators, where the logs in a group must match on all defined fields, transactionize just needs one field to match in order to assign logs to the same group.	_group
		_group_duration
		_group_size
		_group_orphaned
				`| parse "[system=001] [sessionId=]" as system1Id nodrop`
				`| parse "[system=002][sessionId=]" as system2Id nodrop`
				`| parse "[system=003][sessionId=]" as system3Id nodrop`
				`| parse "system=001 with sessionId=" as system1Id nodrop`
				`| transactionize system1Id, system2Id, system3Id`

transpose	The transpose operator dynamically creates columns for aggregate search results. The dynamic functionality allows for changing the output of a query, turning search results into fields. It also means that queries can be designed without first knowing the output schema.			Full query example:
				`_sourceCategory=service`
				`| parse "Successful login for user '', organization: ''" as user, org_id`
				`| timeslice 1d`
				`| count _timeslice, user`
				`| transpose row _timeslice column user`

urldecode	The urldecode operator decodes a URL you include in a query, returning the decoded (unescaped) URL string.	`| urldecode(url) as decoded`
urlencode	The urlencode operator encodes the URL into an ASCII character set.	`| urlencode(url) as encoded`

where	To filter results in a search query, use "where" as a conditional operator. The where operator must appear as a separate operator distinct from other operators, delimited by the pipe symbol ("|"). In other words, the following construct will not work and will generate a syntax error:	`//We recommend placing inclusive filters before exclusive filters in query strings`
		`| where status_code matches "4*"`
		`| where !(status_code matches "2*")`

Math Expressions

You can use general mathematical expressions on numerical data extracted from log lines. For any mathematical or group-by function that implicitly requires integers, Sumo Logic casts the string data to a number for you.

Operator	Description	Example
Basic		

abs	The absolute function calculates the absolute value of x.	`| abs(-1.5) as v`
		`// v = 1.5`

round	The round function returns the closest integer to x.	`| round((bytes/1024)/1024) as MB`

ceil	The ceiling function rounds up to the smallest integer value. Returns the smallest integral value that is not less than x.	`| ceil(1.5) as v`
		`// v = 2`

floor	The floor function rounds down to the largest previous integer value. Returns the largest integer not greater than x.	`| floor(1.5) as v`
		`// v = 1`

max	The maximum function returns the larger of two values.	`| max(1, 2) as v`
		`// v = 2`

min	The minimum function returns the smaller of two values.	`| min(1, 2) as v`
		`// v = 1`

sqrt	The square root function returns the square root value of x.	`| sqrt(4) as v`
		`// v = 2`

cbrt	The cube root function returns the cube root value of x.	`| cbrt(8) as v`
		`// v = 2`

Exponents and Logs		
exp	The exponent function returns Euler's number e raised to the power of x.	`| exp(1) as v` `// v = 2.7182818284590455`
expm1	The expm1 function returns value of x in exp(x)-1, compensating for the roundoff in exp(x).	`| expm1(0.1) as v` `// v = 0.10517091807564763`
log	The logarithm function returns the natural logarithm of x.	`| log(2) as v` `// v = 0.6931471805599453`
log10	The log10 function returns the base 10 logarithm of x.	`| log10(2) as v` `// v = 0.3010299956639812`
log1p	The log1p function computes log(1+x) accurately for small values of x.	`| log1p(0.1) as v` `// v = 0.09531017980432487`
Trigonometric		
sin	Sine of argument in radians.	`| sin(1) as v` `// v = 0.8414709848078965`
cos	Cosine of argument in radians.	`| cos(1) as v` `// v = 0.5403023058681398`
tan	Tangent of argument in radians.	`| an(1) as v` `// v = 1.5574077246549023`
asin	Inverse sine; result is in radians.	`| asin(1) as v` `// v = 1.5707963267948966`
acos	Inverse cosine; result is in radians.	`| acos(x)\`
atan	Inverse tangent; result is in radians.	`| atan(x)`
atan2	Four-quadrant inverse tangent.	`| atan2(0, -1) as v` `// v = pi`
sinh	Hyperbolic sine of argument in radians.	`| sinh(x)`
cosh	Hyperbolic cosine of argument in radians.	`| cosh(x)`
tanh	Hyperbolic tangent of argument in radians.	`| tanh(x)`
Advanced		
hypot	Returns the square root of the sum of an array of squares.	`| hypot(1, 0) as v` `// v = 1`
toDegrees	Converts angles from radians to degrees.	`| toDegrees(asin(1)) as v` `// v = 90`
toRadians	Converts angles from degrees to radians.	`| toRadians(180) as v` `// v = pi`

- Parsing
- Aggregating
- Search Operators
- Math Expressions