Metadata

_sourceHost	The host name of the Source. For local Sources the name of the Source is set when you configure the Source. For remote Collectors, this field uses the remote host's name. The _sourceHost metadata field is populated using a reverse DNS lookup. If the name cannot be resolved, _sourceHost is displayed as localhost.
_sourceName	The name of the log file, determined by the path you entered when you configured the Source.
_sourceCategory	This field is created when you enter text into the Source Category field at Source configuration time. Log categories can be somewhat complex, as many log files may belong to more than one logical category.
_collector	Returns results from the named Collector only. Entered when a Collector is installed and activated.
_source	Returns results from the named Source only. Entered when a Source is configured.

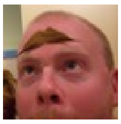
While `_sourcename = *api.log` works, `_sourcename = "*api.log"` will fail.

List all categories: `* | count by _sourceCategory | fields -_count`

Input format

keyvalue	For KVP type logs. The keyvalue operator allows you to get values from a log message by specifying the key paired with each value.	`| keyvalue "age"`
		`| keyvalue infer "hairColor", "lastVisit"`
		`| keyvalue regex "=(.*?)[,|}]" keys "serviceinfo.IP", "loggingcontext.region", "request.m`
		`| keyvalue auto`

csv	The csv operator allows you to parse Comma Separated Values (CSV) formatted log entries. It uses a comma as the default delimiter.	
	Parse comma delimited fields	`| csv_raw extract 1 as user, 2 as id, 3 as name`



Input format (cont)

<table>
<tr><td></td><td>Parse a stream query and extract search terms</td><td><code>"Starting stream query" | parse "query=[*], queryId" as query | csv query extract searchTerms, op1, op2, op3</code></td></tr>
<tr><td></td><td>Specify an escape, and quote character</td><td><code>csv fieldName escape='\', quote=''' extract A, B, _, _, E, F</code></td></tr>
<tr><td><b>JSON</b></td><td colspan="2">The JSON operator 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.</td></tr>
<tr><td></td><td>Extracting a single top-level field</td><td><code>_sourceCategory=stream RawOutputProcessor "\"message\"" | parse "explainJsonPlan.stream]*" as jsonobject | json field=jsonobject "sessionId" | fields -jsonobject</code></td></tr>
<tr><td></td><td>Extracting multiple fields</td><td><code>_sourceCategory=stream RawOutputProcessor "\"message\"" | parse "explainJsonPlan.stream]*" as jsonobject | json field=jsonobject "sessionId", "customerId" | fields -jsonobject</code></td></tr>
<tr><td></td><td>Extracting a nested key</td><td><code>* | json field=jsonobject "meta.type"</code></td></tr>
<tr><td></td><td>Finding values in a JSON array</td><td><code>* | json field=jsonobject "baselineIntervals"</code></td></tr>
<tr><td></td><td>Refer to one specific entry in an array</td><td><code>* | json field=jsonobject "baselineIntervals[1]"</code></td></tr>
<tr><td></td><td>Using the nodrop option</td><td><code>* | json field=jsonobject "baselineIntervals[0]" nodrop</code></td></tr>
<tr><td></td><td colspan="2">Note: The JSON operator also supports the nodrop option, which allows messages containing invalid JSON values to be displayed.</td></tr>
<tr><td></td><td>Using wildcard (*)</td><td><code>_sourceCategory=O365* | json "Actor[*].Type" as Actortype</code></td></tr>
</table>

Input format (cont)

json auto works by searching for json blobs beginning at the end of the message. Usually logs begin with a preamble, such as a timestamp. In cases where content appears at the end of the message after the json blob, the extraction could fail. Having the json blob at the end of the message is recommended, as having it in the middle could cause extraction failure.	`| json auto`

KVP: Key-Value Pairs. Logs formatted this way look something like this:

```
[2019-12-24 23:59:59.380 +1100] age=42 name="Rick Deckard" hairColor="brown" lastVisit="2018-04-19 13:00"
```

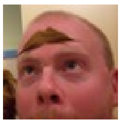
infer: Default mode. Uses an internal list of regex to extract the value for a key.

regex: In Regular Expression mode, you must explicitly match keys and values based on a regex.

auto: Extract up to N fields. N is 100 by default.

Conditions

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`
		`| 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`
where	The where operator must appear as a separate operator distinct from other operators, delimited by the pipe symbol ("|").	`//We recommend placing inclusive filters before exclusive filters in query strings`
		`| where status_code matches "4*"`
		`| where !(status_code matches "2*")`

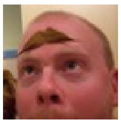


Conditions (cont)

<table>
<tr><td><b>isBlank</b></td><td>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.</td><td><code>| where isBlank(user)</code></td></tr>
<tr><td><b>isEmpty</b></td><td>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.</td><td><code>| if(isEmpty(src_ip),1,0) as null_ip_counts</code></td></tr>
<tr><td><b>isNull</b></td><td>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.</td><td><code>| where isNull(src_ip)</code></td></tr>
</table>

Data extraction

<table>
<tr><td><b>parse(regex)</b></td><td>Best for variable patterns. Also called the extract operator; enables users to extract more complex data from log lines using regular expressions. Can be used to extract nested fields.</td><td><code>| parse "Content=*:" as content</code></td></tr>
<tr><td></td><td>Parsing an IP address</td><td><code>| parse regex "(?<ip_address>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}) "</code></td></tr>
<tr><td></td><td>Indicating an OR condition to use non-capturing groups</td><td><code>| parse regex "list 101 (accepted|denied) (?<protocol>.*?) "</code></td></tr>
<tr><td><b>parse(anchor)</b></td><td>Best for predictable patterns. Also called parse anchor, parses strings according to specified start and stop anchors and labels them as fields for use in subsequent aggregation functions in the query such as sorting, grouping...</td><td><code>| parse "User=*:" as user</code></td></tr>
<tr><td><b>split</b></td><td>The split operator allows you to split strings into multiple strings, and parse delimited log entries, such as space-delimited formats.</td><td><code>_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</code></td></tr>
</table>



Data extraction (cont)

fields	The fields operator allows you to choose which fields are displayed in the results of a query.	`_sourceCategory=access_logs | parse using public/apache | fields m`
limit	The limit operator reduces the number of raw messages or aggregate results returned.	`| count by _sourceCategory | sort by _count | limit 5`

matches	The matches operator can be used to match a string to a pattern.	`| if (agent matches "*MSIE*","Internet Explorer","Other") as Brows`
		`| if (agent matches "*Firefox*","Firefox",Browser) as Browser`

timeslice	The timeslice operator segregates data by time period.	`| timeslice 1h | count by _timeslice`
		`_sourcename=*tomcat* | timeslice by 5m | count by _timeslice`

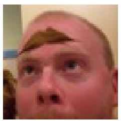
	Output of last example:	`# Time         _count`
		`1 09/07/2017 11:25:00 AM +1000  9,234`
		`2 09/07/2017 11:30:00 AM +1000  14,496`
		`3 09/07/2017 11:35:00 AM +1000  15,988`
		`4 09/07/2017 11:40:00 AM +1000  3,383`

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.	`| trace "ID=( [0-9a-fA-F] {4} )" "7F92"`

About limit: Can be used in Dashboard Panels, but in the search they must be included after the first group-by phrase.
About timeslice: Timeslices greater than 1 day cannot be used in Dashboard Live mode.
About trace: Not supported in Live Dashboards or any continuous query.

Crunch numbers

<table>
<tr><th></th><th></th><th></th></tr>
<tr><td><strong>count</strong>
<strong>count_distinct</strong>
<strong>count_frequent</strong></td><td>Used in conjunction with the group operator and a field name. Only the word by is required. The count function is also an operator in its own right and therefore can be used with or without the word by.</td><td><code>| count by url</code>
<code>| count_distinct(referrer) by status_code</code>
<code>_sourcename=*tomcat* | count_distinct(_sourceName) group by _sourceHost | sort by _c</code></td></tr>
<tr><td><strong>sum</strong></td><td>Sum adds the values of the numerical field being evaluated within the time range analyzed.</td><td><code>| sum(bytes_received) group by _sourceHost</code></td></tr>
</table>



By **TME520** (TME520)
cheatography.com/tme520/
tme520.com

Crunch numbers (cont)

<table>
<tr><td><b>avg</b></td><td>The averaging function (avg) calculates the average value of the numerical field being evaluated within the time range analyzed.</td><td><code>| avg(request_received) by _timeslice</code></td></tr>
<tr><td><b>median</b></td><td>In order to calculate the median value for a particular field, you can utilize the Percentile (pct) operator with a percentile argument of 50.</td><td><code>| parse "value=*" as value | pct(value, 50) as median</code></td></tr>
<tr><td><b>outlier</b></td><td>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.</td><td><code>_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=+-</code></td></tr>
<tr><td><b>sort</b></td><td>The sort operator orders aggregated search results. The default sort order is descending.</td><td><code>| count as page_hits by _sourceHost | sort by page_hits asc</code></td></tr>
<tr><td><b>top</b></td><td>Use the top operator with the sort operator, to reduce the number of sorted results returned.</td><td><code>| top 5 _sourcecategory</code></td></tr>
<tr><td><b>min</b></td><td>The minimum function returns the smaller of two values.</td><td><code>| min(1, 2) as v</code>
<code>// v = 1</code></td></tr>
<tr><td><b>max</b></td><td>The maximum function returns the larger of two values.</td><td><code>| max(1, 2) as v</code>
<code>// v = 2</code></td></tr>
</table>

About count_frequent: You can use the count_frequent operator in Dashboard queries, but the number of results returned is limited to the top 100 most frequent results.
About top: Can be used in Dashboard Panels, but in the search they must be included after the first group-by phrase.

Geo lookup

Sumo Logic can match an extracted IP address to it's geographical location on a map. To create the map, after parsing the IP addresses from log files, the lookup operator matches extracted IP addresses to the physical location where the addresses originated.	`| parse "remote_ip=*]" as remote_ip | lookup latitude, longitude, country_code, country_name, region, city, postal_code, area_code, metro_code fromgeo://default on ip = remote_ip | count by latitude, longitude, country_code, country_name, region, city, postal_code, area_code, metro_code | sort _count`

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	`| logcompare timeshift -24h`

About logcompare: Not supported in Dashboards.

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.	`| logreduce`

About logreduce: Not supported in Dashboards.

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.	`| save /shared/lookups/daily_users`

About save: Not supported in Dashboards.

Visualization

transpose	Turn a list into a table in the Aggregates tab.	`transpose row [row fields] column [column fields]`
		`_sourceCategory=Labs/Apache/Access | timeslice 5m | count by _timeslice, status_code | transpose row _timeslice column status_code`