INGEST_EVAL and CLONE_SOURCETYPE

Advanced pipeline configurations

Richard Morgan
Vladimir Skoryk
Splunk

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or plans of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results may differ materially. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, it may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements made herein.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only, and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.

Who Are We?

Veteran Splunkers

Richard Morgan

- Principal Architect
- 6 years Splunker
- Full stack expert
- Data junkie and SPL addict
- Enjoys reverse engineering
- Located in London EMEA

Vladimir Skoryk

- Principal PS Architect
- 7 years with Splunk PS
- Loves photography
- Lives at MIA

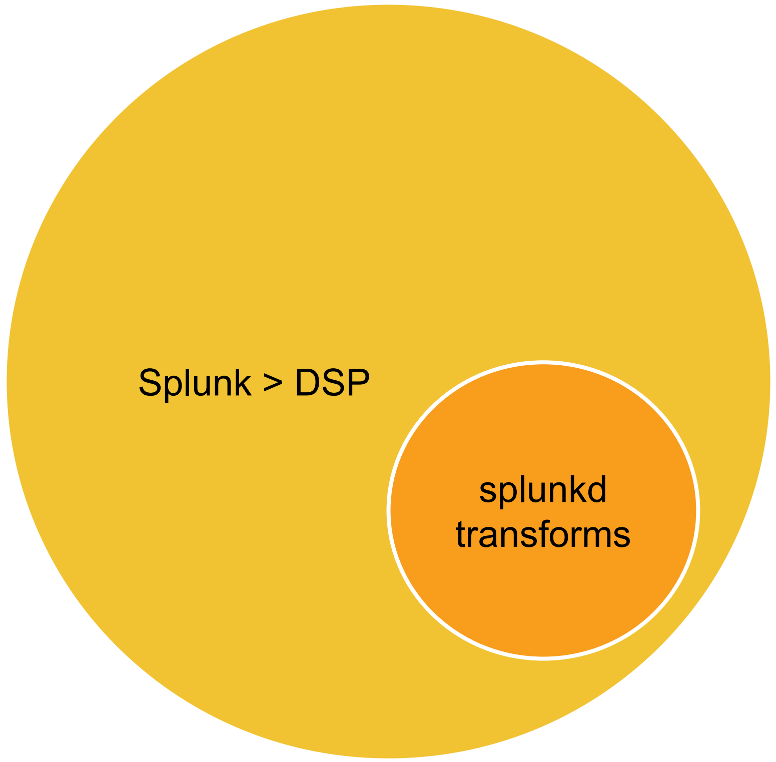
How splunkd Transforms Compare to DSP

DSP

- GUI for pipeline design
- Debugging tools
- Supports SPL2
- Multiple integration points in / out
- Is the future

Transforms

- Apply EVAL logic on indexed fields
- Apply REGEX to events
- Implemented in splunkd



Splunkd transforms are a subset of the functionality found in DSP

What Are Splunk Transforms?

(how to suck eggs, don't worry we will be really fast!!!)

Transforms Apply Rules to Incoming Data

- They are applied at on an indexer, heavy forwarder (and a UF if local_indexing=true)
- They can be applied during event parsing (and at search time, but we are ignoring that)
- They can address and modify the *fields _raw, host, sourcetype, source, etc*
- They can add, remove and modify user defined indexed fields from _meta
- They can clone (make another copy of) an event
- They can route events out via S2S, syslog or TCP
- They can selectively delete/ drop events

Splunkd's Major Transformation Options

INGEST_EVAL (7.2) - These allow you to address multiple fields, and provides the near full EVAL library from SPL into the ingestion process

CLONE_SOURCETYPE (6.2) - A extension to REGEX that allows you to create a fresh new copy of an event in the data stream

SED_CMD - This applies a SED command to your _raw string to replace and mask data

REGEX - These allow you apply regular expressions to extract text data and copy between the metadata keys / registers.

Combine these transforms for synergy and profit!

Functional Overlap Between Commands

99% of what you want to do can be achieved with INGEST_EVAL

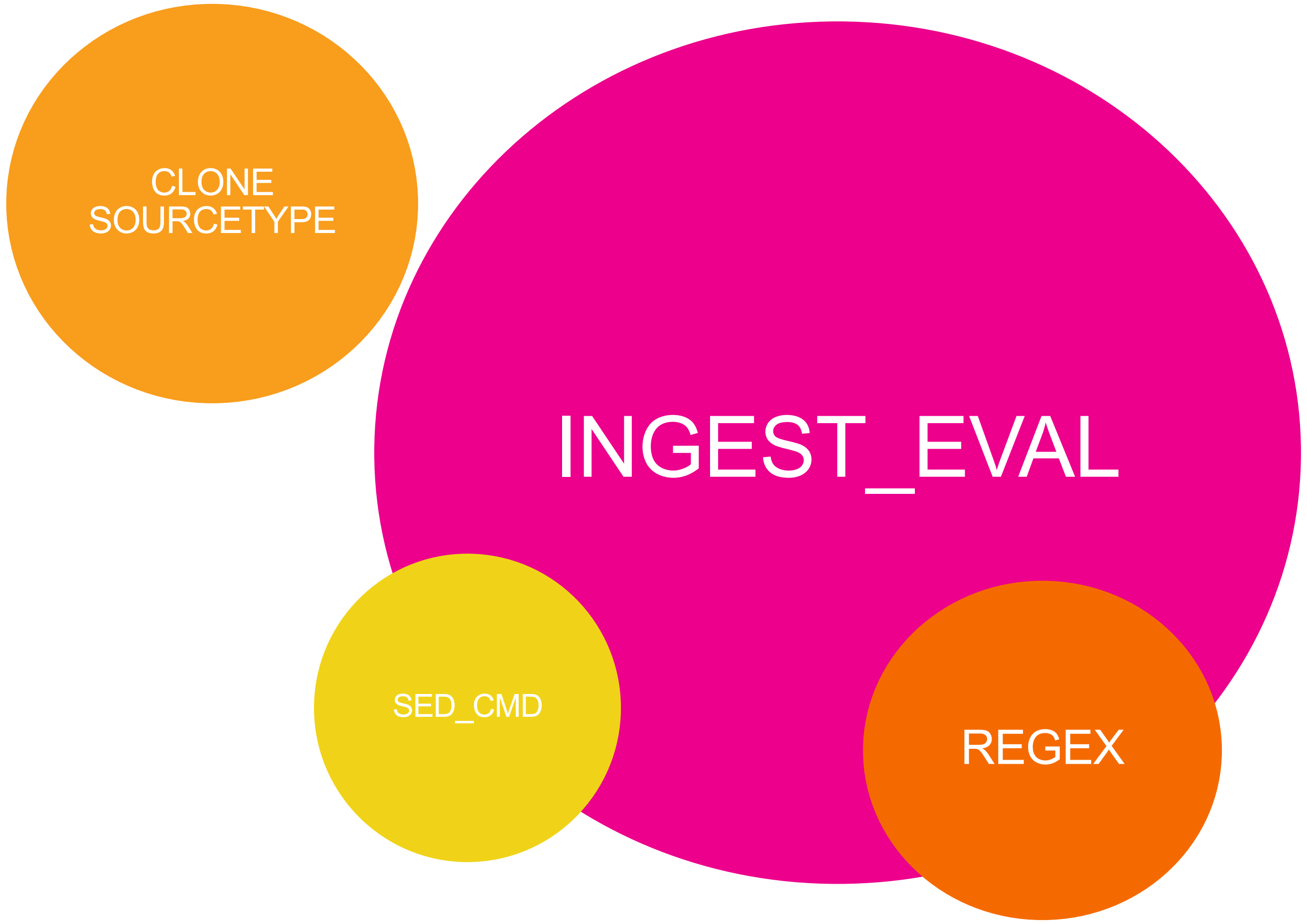
INGEST_EVAL has the greatest versatility and can mostly replace both SED_CMD and REGEX by with its replace() function. However there are exceptions:

1) REGEX allows you to build variables names and set values, whereas INGEST_EVAL only allows you to assign values to known names.

2) REGEX allows for repeated matching, but the eval replace command does not.

3) SED_CMD also allows for repeated matching within the _raw string.

4) REGEX uses a compiled REGEX library and is more efficient on resources



Default Keys You Can Read and Write To

REGEX transform name	**INGEST_EVAL** name	**Notes**
_time	_time	The timestamp for the event in unix epoch time (the only number attribute)
_raw	_raw	The event string itself, the _raw you see in search
Metadata:Host	host	The label for the host, the host you see in search
Metadata:Source	source	The label for the source, the source you see in search
Metadata:Sourcetype	sourcetype	The label for the sourcetype, the host you see in sourcetype
_meta	**NOT AVAILABLE**	The buffer that contains a delimited list of indexed fields
index	index	The target index for the event
queue	queue	The next step in the parsing queue (rarely specified unless set to null)
_TCP_ROUTING	_TCP_ROUTING	Set to the target output group for forwarding to another host

This is not an exhaustive list, other keys may exist, but are not necessarily useful

What Is the *_meta* for?

Flipping the hood on how indexed fields are put in the lexicon

Cracking Open _*meta* for Inspection

```
[regex_copy_meta_to_raw]
SOURCE_KEY = _meta
DEST_KEY = _raw
REGEX = (.*)
FORMAT = $1
```

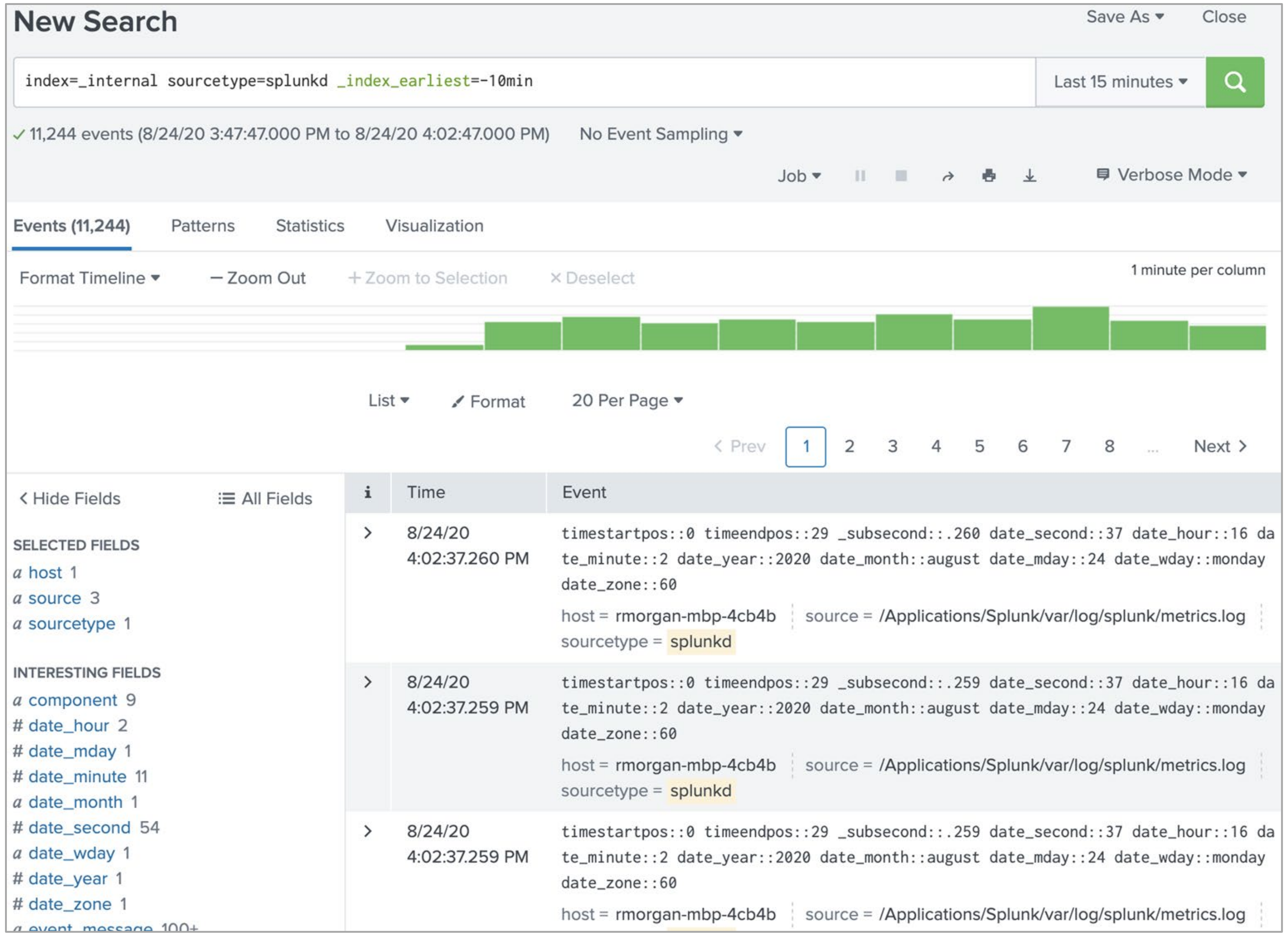
To aid understanding of how indexed fields are added we will look inside the _*meta* key

The transform on the LHS copies the contains of _*meta* into _*raw*

This allows the contents of _*meta* can be viewed in search

This is an invaluable technique for debugging transforms

Looking Inside _*meta* for splunkd.log Events



```
index=_internal sourcetype=splunkd _index_earliest=-10min
```

✓ 11,244 events (8/24/20 3:47:47.000 PM to 8/24/20 4:02:47.000 PM)

SELECTED FIELDS
- a host 1
- a source 3
- a sourcetype 1

INTERESTING FIELDS
- a component 9
- # date_hour 2
- # date_mday 1
- # date_minute 11
- a date_month 1
- # date_second 54
- a date_wday 1
- # date_year 1
- # date_zone 1
- a event_message 100+

i	Time	Event

>	8/24/20 4:02:37.260 PM	`timestartpos::0 timeendpos::29 _subsecond::.260 date_second::37 date_hour::16 date_minute::2 date_year::2020 date_month::august date_mday::24 date_wday::monday date_zone::60`
		host = rmorgan-mbp-4cb4b source = /Applications/Splunk/var/log/splunk/metrics.log sourcetype = splunkd

>	8/24/20 4:02:37.259 PM	`timestartpos::0 timeendpos::29 _subsecond::.259 date_second::37 date_hour::16 date_minute::2 date_year::2020 date_month::august date_mday::24 date_wday::monday date_zone::60`
		host = rmorgan-mbp-4cb4b source = /Applications/Splunk/var/log/splunk/metrics.log sourcetype = splunkd

>	8/24/20 4:02:37.259 PM	`timestartpos::0 timeendpos::29 _subsecond::.259 date_second::37 date_hour::16 date_minute::2 date_year::2020 date_month::august date_mday::24 date_wday::monday date_zone::60`
		host = rmorgan-mbp-4cb4b source = /Applications/Splunk/var/log/splunk/metrics.log

We have copied the _*meta* field into _*raw* for splunkd events using the transform on the previous slide. These fields become indexed fields.

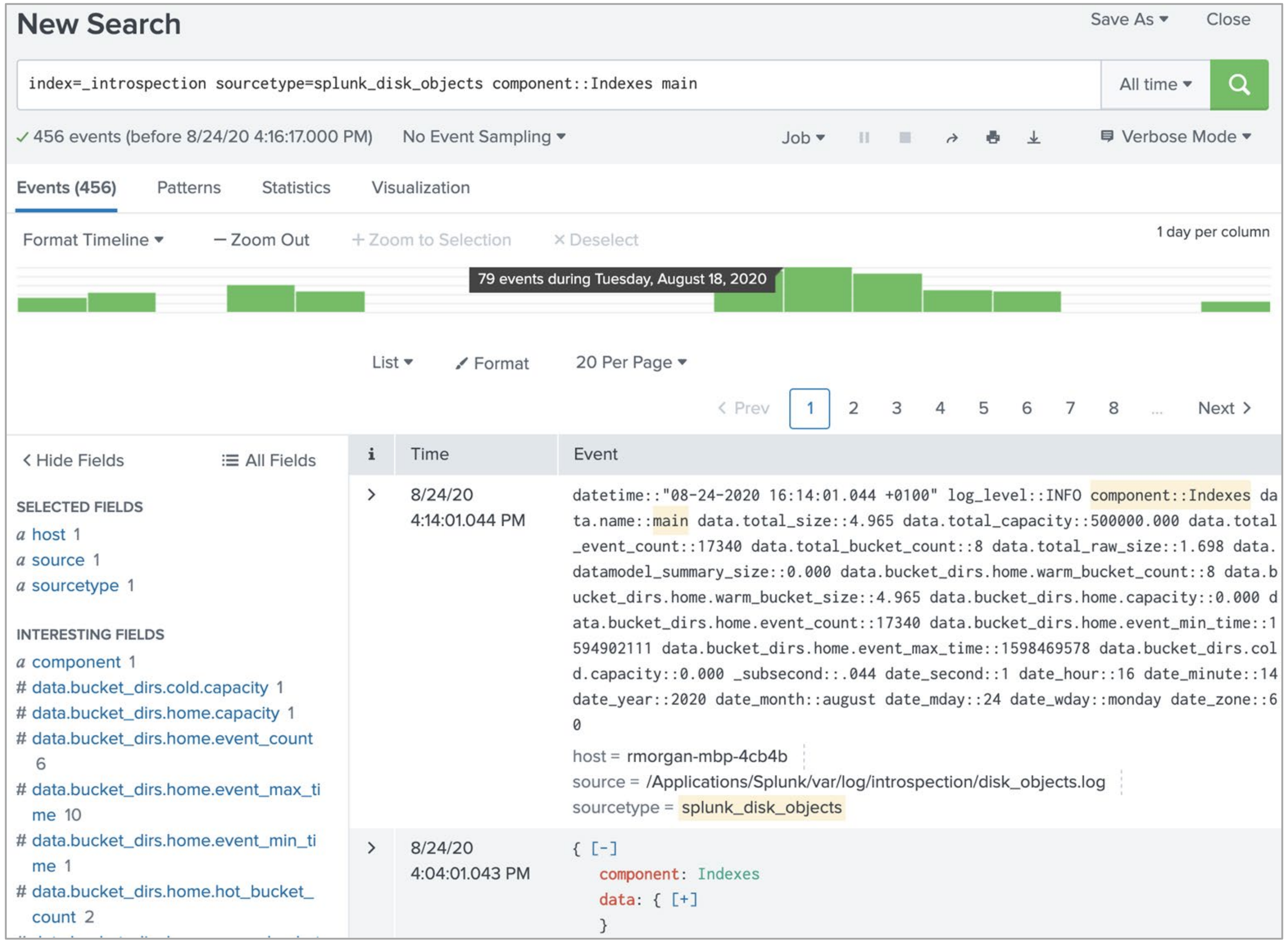
We can see the default user defined indexed fields generated by splunk when reading log files (not included when ingesting via HEC)

Note that *punct* is yet to computed as it happens in the tokenization process at the point of indexing

Note that *subseconds* are implemented as via indexed fields.

The values of *date_** are fossils from splunk version 4

Looking Inside _*meta* for disk_objects.log Events


New Search

Save As ▾ Close

```
index=_introspection sourcetype=splunk_disk_objects component::Indexes main
```

All time ▾

✓ 456 events (before 8/24/20 4:16:17.000 PM) No Event Sampling ▾ Job ▾ Verbose Mode ▾

Events (456) Patterns Statistics Visualization

Format Timeline ▾ — Zoom Out + Zoom to Selection × Deselect 1 day per column

79 events during Tuesday, August 18, 2020

List ▾ Format 20 Per Page ▾

‹ Prev 1 2 3 4 5 6 7 8 ... Next ›

‹ Hide Fields ☰ All Fields

SELECTED FIELDS
- a host 1
- a source 1
- a sourcetype 1

INTERESTING FIELDS
- a component 1
- # data.bucket_dirs.cold.capacity 1
- # data.bucket_dirs.home.capacity 1
- # data.bucket_dirs.home.event_count 6
- # data.bucket_dirs.home.event_max_time 10
- # data.bucket_dirs.home.event_min_time 1
- # data.bucket_dirs.home.hot_bucket_count 2

i	Time	Event
›	8/24/20 4:14:01.044 PM	`datetime::"08-24-2020 16:14:01.044 +0100" log_level::INFO component::Indexes data.name::main data.total_size::4.965 data.total_capacity::500000.000 data.total_event_count::17340 data.total_bucket_count::8 data.total_raw_size::1.698 data.datamodel_summary_size::0.000 data.bucket_dirs.home.warm_bucket_count::8 data.bucket_dirs.home.warm_bucket_size::4.965 data.bucket_dirs.home.capacity::0.000 data.bucket_dirs.home.event_count::17340 data.bucket_dirs.home.event_min_time::1594902111 data.bucket_dirs.home.event_max_time::1598469578 data.bucket_dirs.cold.capacity::0.000 _subsecond::.044 date_second::1 date_hour::16 date_minute::14 date_year::2020 date_month::august date_mday::24 date_wday::monday date_zone::60` host = rmorgan-mbp-4cb4b source = /Applications/Splunk/var/log/introspection/disk_objects.log sourcetype = splunk_disk_objects
›	8/24/20 4:04:01.043 PM	`{ [-] component: Indexes data: { [+] } }`

We have configured a transform that copies the contents of _*meta* to _*raw* for some events encoded with INDEXED_JSON. These values in _*meta* become indexed fields.

We can see a see how JSON is transformed into attribute values held in the _*meta* field.

We can also see the default values mentioned in the previous slide.

Manipulating _*meta* Field With REGEX

# This appends two indexed fields to the _meta field. If those fields exist, they become multi value

```
WRITE_META = True
FORMAT = abc::123 def::456
```

# This overwrites the _meta field and replaces it with just two indexed fields

```
DEST_KEY = _meta

FORMAT = abc::123 def::456
```

# This appends a single multi value field to meta

```
FORMAT = mv_field::1 mv_field::2
```

The original REGEX transform allowed for very simple manipulation of _*meta*. You could only overwrite or add to it

It is not possible to selectively delete from _*meta*, nor could you do any computation on the fields.

It is very easy to create a corrupted _*meta* string when using REGEX to build them

However REGEX remains the primary way to extract data from events and write them into _*meta*

REGEX transforms offer a REPEAT_MATCH option

Manipulating *_meta* With INGEST_EVAL

```
# This appends two fields abc def to _meta. It may create multi value fields if they already exist
INGEST_EVAL = abc=123, def=456

# This appends to _meta and will overwrite any existing values for xyz
INGEST_EVAL = xyz:=789

# This will delete the field abc from _meta
INGEST_EVAL = abc:=null()
```

INGEST_EVAL is effectively a wrapper around _*meta* that allows you to atomically manipulate the entries in the field

On the left we can see the basic operations for manipulating the entities in the _*meta* string

In addition to this we have the (near) full library of EVAL expressions as found in SPL

Tips for When Working with Transforms

Use Visual Studio Code with the Splunk Extension to manage configs

Instead of restarting Splunk, use the reload URL http://localhost:8000/en-US/debug/refresh

Use `_index_earliest` and `_index_latest` to view recently ingested data

Only develop on your laptop as messing with ingestion is **ultra dangerous**

Use the `[copy_to_meta]` transform to debug

Don't use "`one shot`" use "`nom on`" and enjoy the easter egg!

Advanced Pipeline Configurations

Ten of them….Sorry we made you wait so long!

1. License Usage

Splunk typically charges on an ingestion and some customers chose to bill their internal customers via this method as well

Typically people refer to *license_usage.log* to compute and allocate the ingestion costs. However this is a pretty crude, and the searches are expensive

We can use INGEST_EVAL to compute the string length for each event and write it to an indexed field where we can performance license calculation

1. Add Event Length as Indexed Field

props.conf

```
# Thanks to "default" this configuration is applied to all sourcetypes and it adds an indexed field len
of the event

# This is very useful for using with tstats to sum up all ingested data from any source very quickly

# We must try and make sure that this transforms is the last to be applied to event, otherwise it is
possible that subsequent transforms may shorten or lengthen _raw after it is computed.

[default]

TRANSFORM-z-last_transform = add_raw_length_to_meta_field
```

transforms.conf

```
# This transforms calls len() to determine the length of the string in _raw and write the result to an
indexed field event_length

[add_raw_length_to_meta_field]

INGEST_EVAL = event_length=len(_raw)
```

1. Using the Indexed Field Using tstats

```
| tstats
    sum(event_length) AS total_ingestion
    WHERE index=* _index_earliest=-30d@d _index_latest=-1d@d
    BY sourcetype _time span=1d@d
| xyseries _time sourcetype total_ingestion
```

Modify the example to split by other indexed fields like host and source, etc.

2. Data Estimation Pipeline Metrics Edition

Prior to a net new data source being onboarded, additional context might be required to ensure sufficient compute and storage is available.

We can use INGEST_EVAL and CLONE_SOURCETYPE functionality to emit metrics events can describe the data coming in, original information does not need to be kept!

- Note, metric event takes ***up to 150 bytes*** of license

2. Data Estimation Pipeline Metrics

props.conf

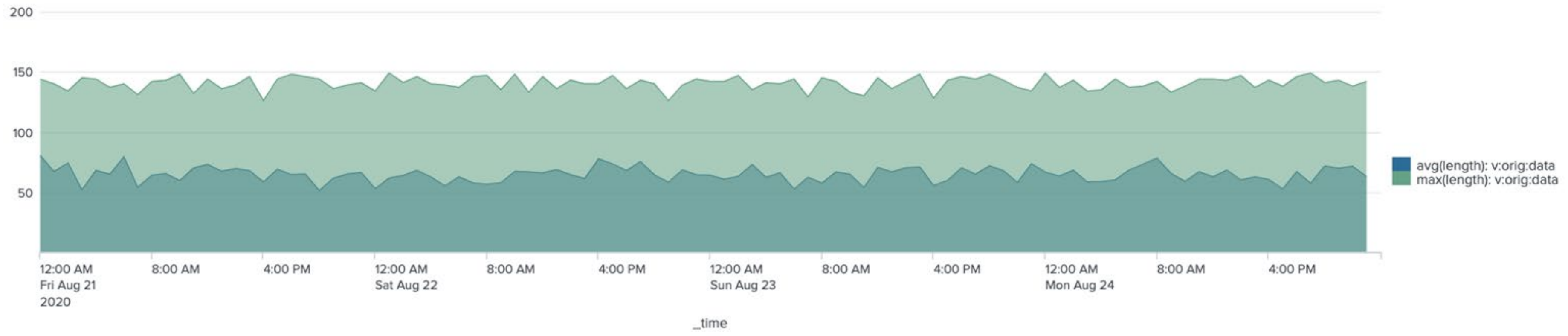
```
# data comes in on this sourcetype
[v:orig:data]
# this configuration is universal, and can be reused
TRANSFORMS-enable_estimate_mode_drop_orig =
v_estimation_set_metrics, v_estimation_create_metrics,
v_estimation_drop_orig


# metrics of metadata are created on this sourcetype
[v:estimate:pipeline]
TRANSFORMS-set_metric_name = v_estimation_metric_info
```

transforms.conf

```
# clone original data, will be transformed into metrics
event
[v_estimation_create_metrics]
REGEX = (.*)
CLONE_SOURCETYPE = v:estimate:pipeline

# create metadata about the event, preserve original
attributes
# these fields will become metric dimensions!
[v_estimation_set_metrics]
INGEST_EVAL = orig_host=host, orig_source=source,
orig_sourcetype=sourcetype, orig_index=index

# we do not need to keep the original data, only want
the metadata, let's drop
[v_estimation_drop_orig]
INGEST_EVAL = queue="nullQueue"

# format event into a metric, route it to appropriate
metrics index
[v_estimation_metric_info]
INGEST_EVAL = index="data_metrics",
metric_name="estimation_mode", _value=len(_raw)
```

2. Data Estimation Pipeline Metrics

```
| mstats prestats=t max(_value) avg(_value)
      WHERE index=data_metrics AND metric_name="estimation_mode"
      BY orig_sourcetype span=5m
| timechart max(_value) avg(_value) BY orig_sourcetype
```



3. Selective Routing to Other Destinations

Sometimes data needs to be shared with other destinations, Splunk or 3rd party systems.

Use cases can require *all* or *subset* of data to be shared. **INGEST_EVAL** can be used to control data routing groups.

Tip: For advanced routing pipelines, see **Splunk Data Stream Processor (DSP)**

3. Selective Routing to Other Destinations

props.conf

```
# data comes in on this sourcetype
[v:interesting:data]
TRANSFORMS-example_data_route = v_sample_route_buttercup_bu
```

transforms.conf

```
# if event data host is from buttercup1 OR buttercup2
# route data to Splunk Cloud, otherwise send it to onprem indexers
[v_sample_route_buttercup_bu]
INGEST_EVAL = _TCP_ROUTING=if(match(host, "buttercup[12]"), "splunkcloud_indexers", "splunk_onprem_indexers")
```

outputs.conf

```
[tcpout]
defaultGroup = splunk_onprem_indexers

# this output group routes data to Splunk Cloud
[tcpout:splunkcloud_indexers]
server = inputs.buttercup.splunkcloud.com:9997

# this output group keeps data on-prem
[tcpout:splunk_onprem_indexers]
server = 10.10.10.10:9997
```

4. Manage Conflicting Time Formats

Any well curated splunk instance will use sourcetype to accurate identify the event format timestamp

However occasionally collisions occur in a single sourcetype where there are conflicting date stamps. An example of such data is shown on the RHS

Traditionally the solution is to use *datetime_config.xml* and hope for the best or roll your own

INGEST_EVAL offers a new approach of using *strptime() function*

mutliplexed_datetime_formats.log

```
Fri Aug 21 20:39:18 2020 splunk> We enjoy breaks more than Unions
20:42:36 20-08-21 splunk> I gotta fever, and the only cure is MOAR LICENSE!
20:46:49 20-08-21 splunk> All batbelt. No tights.
2020-08-21 21:02:08 splunk> Finding disturbances in the Force before the Jedi
Masters
Wed Aug 26 13:38:47 2020 splunk> These are the droids you are looking for
Wed Aug 26 13:57:24 2020 Splunk> Take the sh out of IT.
Wed Aug 26 14:00:21 2020 Splunk> Winning the War on Error
Wed Aug 26 14:01:37 2020 Splunk> See your world. Maybe wish you hadn’t.
2020-08-26 14:05:52 Splunk> Be an IT superhero. Go home early.
14:24:50 20-08-26 Splunk> Winning the War on Error
2020-08-26 14:45:14 splunk> More flexible than an Olympic gymnast.
2020-08-26 14:57:46 splunk> Finding your faults, just like mom.
2020-08-26 15:04:27 Splunk> The IT Search Engine.
15:11:32 20-08-26 Splunk> Take the sh out of IT.
Wed Aug 26 15:13:01 2020 Splunk> see the light before you tunnel
Wed Aug 26 15:30:55 2020 splunk> don't get caught up in the game of pwns
15:33:06 20-08-26 splunk> We line break for regular expressions
15:37:49 20-08-26 splunk> ""\. nuff said.
2020-08-26 15:41:32 splunk> These are the droids you are looking for
15:49:09 20-08-26 splunk> IT like you mean it
2020-08-26 15:54:34 splunk> These are the droids you are looking for
Wed Aug 26 15:55:53 2020 Splunk> Needle. Haystack. Found.
2020-08-26 16:03:06 Splunk> I like big data and I cannot lie.
16:13:14 20-08-26 splunk> ""\. nuff said.
2020-08-26 16:13:27 Splunk> see the forest, and the trees
```

4. Configuration to Demultiplex Conflicting Time Formats

props.conf

```
[demutliplexed_datetime_formats]

DATETIME_CONFIG = CURRENT

TRANSFORMS-extract_date = demultiplex_datetime
```

transforms.conf

```
[demultiplex_datetime]
# add fall-through case to set custom date or route “unknown” data to special quarantine index

INGEST_EVAL= _time=case(isnotnull(strptime(_raw, "%c")), strptime(_raw, "%c"), isnotnull(strptime(_raw, "%H:%M:%S %y-
%m-%d")),strptime(_raw, "%H:%M:%S %y-%m-%d"), isnotnull(strptime(_raw, "%Y-%m-%d %H:%M:%S")), strptime(_raw, "%Y-%m-
%d %H:%M:%S"))
```

In this example we initially set the time of the event to be the current time. After this we use a transform to try and replace that time by testing the known time formats using a case statement and pick the first that matches.
This is not very computationally efficient as we are invoking strptime multiple times, but we are able to get the answer in a single invocation of INGEST_EVAL.

5. Extract Time and Data From File Name

Sometimes the date and time files are split up and need to be rejoined for date parsing.

Previously we would need to use *datetime_config.xml* and hope for the best or roll your own.

With **INGEST_EVAL** we can tackle this problem more elegantly

The RHS shows an examples of such an output

```
(base) rmorgan-mbp-4cb4b:compound_date_time rmorgan$ ls -l
total 152
-rw-r--r--  1 rmorgan  wheel  1052 17 Aug 22:15 2020-08-17.log
-rw-r--r--  1 rmorgan  wheel   808 17 Aug 22:15 2020-08-18.log
-rw-r--r--  1 rmorgan  wheel   891 17 Aug 22:15 2020-08-19.log
-rw-r--r--  1 rmorgan  wheel   932 17 Aug 22:15 2020-08-20.log
-rw-r--r--  1 rmorgan  wheel  9063 21 Aug 20:19 2020-08-21.log
-rw-r--r--  1 rmorgan  wheel  8692 21 Aug 20:19 2020-08-22.log

(base) rmorgan-mbp-4cb4b:compound_date_time rmorgan$ head -10 2020-
08-17.log
01:23:11 splunk> Digs deeper than a jealous spouse.
03:42:27 Splunk> Be an IT superhero. Go home early.
04:07:08 splunk> More flexible than an Olympic gymnast.
04:24:47 splunk> Walking War Room!!
04:30:41 Splunk> see the light before you tunnel
06:10:07 Splunk> data with destiny
06:29:28 splunk> More flexible than an Olympic gymnast.
06:52:00 splunk> ""\. nuff said.
07:52:59 Splunk> Take the sh out of IT.
08:44:00 Splunk> See your world. Maybe wish you hadn’t.
```

5. Combine Source and _raw to Create Date Stamp

props.conf

```
[compound_date_time]

DATETIME_CONFIG = CURRENT

TRANSFORMS-get-date = construct_compound_date

SHOULD_LINEMERGE = false

LINE_BREAKER = ([\n\r]+)
```

transforms.conf

```
# use regex replace to pop out the date form the source, append on the first 10 chars from _raw and then run
through strftime and assign the result to _time. If the eval fails to execute _time is not updated and the
previously set CURRENT time will remain

[construct_compound_date]

INGEST_EVAL=_time=strptime(replace(source,".*/(20\d\d\-\d\d\-\d\d)\.log","\1").substr(_raw,0,10),"%Y-%m-
%d%H:%M:%S")
```

6. Event Sampling

Consider you have a web server generating **1000's** events per second, we only care about errors, and the ratio of errors to OK. We can sample the OK, and provide high resolution for errors

Also a great way to route *subset* of data to a TEST, DEV, or UAT environments!

- ..when combined with selective routing example

6. Event Sampling

props.conf

```
# data comes in on this sourcetype
[v:orig:data]
TRANSFORMS-sample_200_data = v_sample_200_data
```

transforms.conf

```
# will look for events with status code 200 AND random number not equal to zero
# if true, drop the data
# if false, example will keep *roughly* one event out of 100
[v_sample_200_data]
INGEST_EVAL = queue=if(match(_raw, "status=200") AND (random()%100)!=0, "nullQueue",
"indexQueue")
```

7. Dropping Fields from INDEXED_CSV

Both *INDEXED_CSV* and *INDEXED_JSON* is very cool but it creates indexed fields for every column or element which can inflate your TSIDX size that increases disk usage.

Sometimes we would like a subset of these fields for fast search but have the remaining available via schema on the fly.

```
primary_key,primary_value,repeated_field,random_nonsense,long_payload
0,285719,same silly value,98e41eba-90d4-4820-ac24-0b8135072857,splunk> this way: Run-D.M.C.
1,282189,same silly value,f86517cb-7a96-4363-9d50-6e890611827a,splunk> We enjoy breaks more than Unions
2,775074,same silly value,4e98b505-bd8d-49c0-8f7b-c889905350b3,splunk> this way: Run-D.M.C.
3,883007,same silly value,0ac163c5-f78e-4d20-8dba-b3c9f576f3dc,splunk> The mars rover of the IT landfill.
4,904525,same silly value,3ced7433-aaf0-4f5c-ac26-1c9b3b1cdb8f,splunk> The mars rover of the IT landfill.
5,939794,same silly value,46390cae-fd48-4a4e-a550-883f012e6145,splunk> We enjoy breaks more than Unionsers
7,183164,same silly value,b3cd4a3e-b2a2-46b7-938f-352009e6d420,Splunk> The IT Search Engine.
8,636841,same silly value,88903bf1-ed92-4120-ba4e-16c977318e07,splunk> this way: Run-D.M.C.
9,724250,same silly value,e03375dd-53a4-4a24-8356-c6c6ab097d51,Splunk> The Notorious B.I.G. D.A.T.A.
10,515046,same silly value,1e952330-7e52-4f9a-99d7-f9f72399c148,Splunk> Australian for grep.
11,492531,same silly value,3ace4c1a-e09c-4da2-83d3-be3dff566ca0,splunk> Walking War Room!!
12,869021,same silly value,447b350c-9a56-4848-bc95-7e479d75b2b7,splunk> Digs deeper than a jealous spouse.
13,821334,same silly value,2036fa19-e2a8-480b-9b09-df85e08b1696,splunk> Show me your logs
14,385989,same silly value,522e36bb-1c8f-4279-94e5-6c7cb72147bd,Splunk> All batbelt. No tights.
16,925553,same silly value,f99ae57a-24e2-4b0b-b45c-e3259ad0e5de,Splunk> The IT Search Engine.
17,583905,same silly value,cb77de52-f021-46ba-b22d-0ee35c8aac4f,splunk> Show me your logs
18,237347,same silly value,4266d252-a616-4492-b8e9-17f18146746c,Splunk> Take the sh out of IT.
19,570619,same silly value,436e610d-e8a2-4890-8868-dbd9f2a98278,Splunk> 4TW
20,970552,same silly value,1099bf0e-c7e7-45d2-be20-2fa2581451e5,splunk> Walking War Room!!
21,872840,same silly value,edf64f2f-0985-4e00-9ce0-2023dd6be07d,Splunk> The IT Search Engine.
22,155976,same silly value,1a43ca31-1f1c-425d-89a2-c6aa3584b607,splunk> More flexible than an Olympic
gymnast.
```

useless_columns.csv

7. Drop Indexed Fields, Replace With regex

props.conf

```
[reduced_columns]
DATETIME_CONFIG = CURRENT
INDEXED_EXTRACTIONS = CSV
TRANSFORMS-drop_fields = drop_useless_fields
EXTRACT-removed-columns = [^,]+,[^,]+,[^,]+,(?<random_nonsense>[^,]+),(?<long_payload>[^,]+)
```

transforms.conf

```
[drop_useless_fields]
# note the := syntax
INGEST_EVAL = repeated_field:=null(), random_nonsense:=null(), long_payload:=null()
```

8. Export and Import Data From Splunk

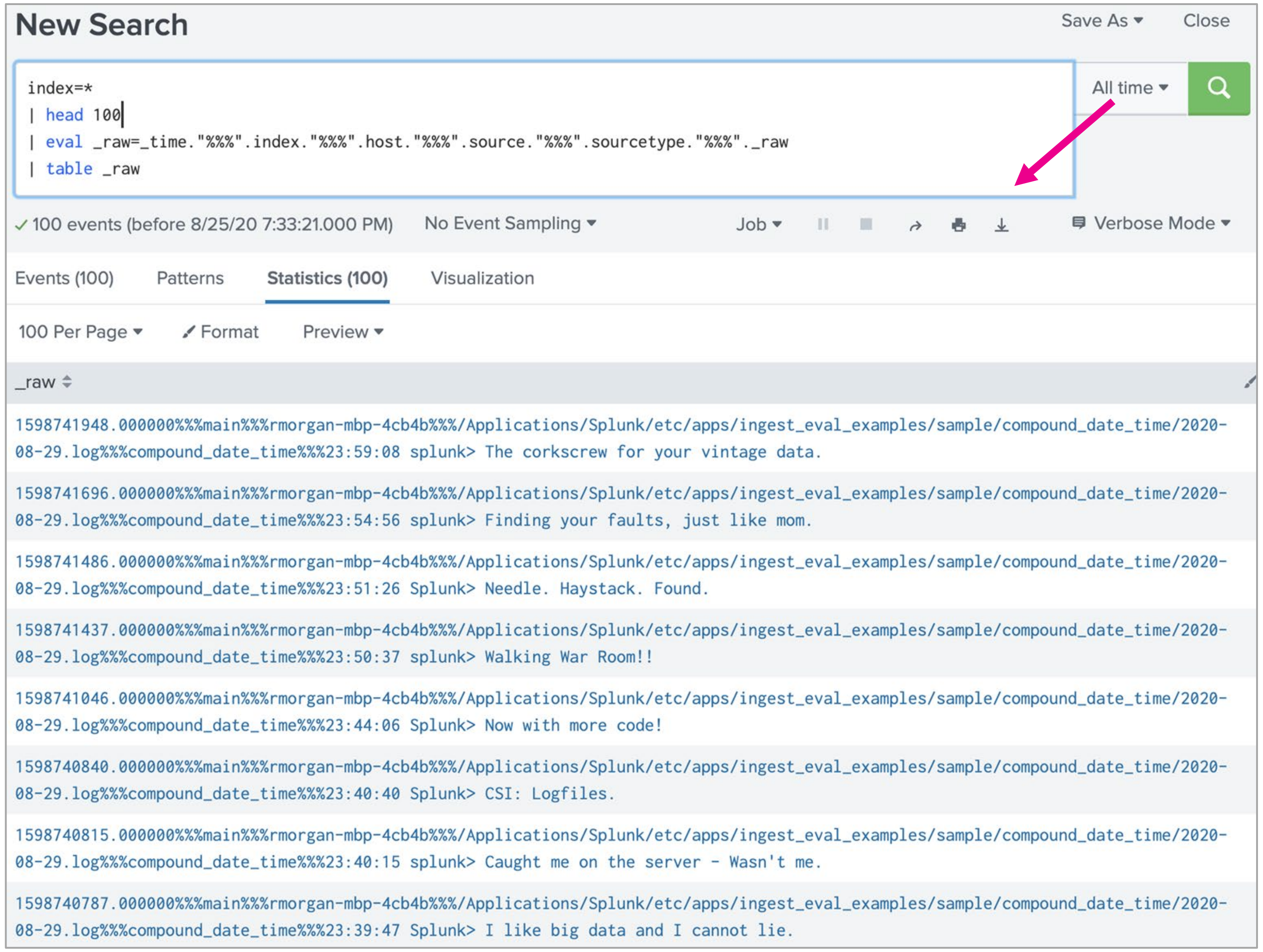
Sometimes you would like to bulk export data from an existing Splunk index and reingest on your laptop for development

This pattern allows you to run a search that extracts data from an install via CSV export and import it again via a specific sourcetype.

This is achieved by creating a “protocol” for encoding via search, and then decoding via transforms.

Note that version this does not reparse data or does it carry any indexed fields across.

8. Export Data Into Export Format



```
index=*
| head 100
| eval _raw=_time."%%%".index."%%%".host."%%%".source."%%%".sourcetype."%%%"._raw
| table _raw
```

✓ 100 events (before 8/25/20 7:33:21.000 PM)

```
_raw
1598741948.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:59:08 splunk> The corkscrew for your vintage data.
1598741696.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:54:56 splunk> Finding your faults, just like mom.
1598741486.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:51:26 Splunk> Needle. Haystack. Found.
1598741437.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:50:37 splunk> Walking War Room!!
1598741046.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:44:06 Splunk> Now with more code!
1598740840.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:40:40 Splunk> CSI: Logfiles.
1598740815.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:40:15 splunk> Caught me on the server - Wasn't me.
1598740787.000000%%%main%%%rmorgan-mbp-4cb4b%%%/Applications/Splunk/etc/apps/ingest_eval_examples/sample/compound_date_time/2020-08-29.log%%%compound_date_time%%%23:39:47 Splunk> I like big data and I cannot lie.
```

The screen host on the LHS shows 100 events encoded by search into a single column table and ready for export via CSV.

The "protocol" uses a %%% as a separator and we order it as index, host, sourcetype, source and then *_raw*.

We assume that the % character is only found in *_raw* to optimize our REGEX statement.

Copy and paste version of the search in the image:

```
index=* | eval
_raw=_time."%%%".index."%%%".host."%%%".source."%%%".sourcetype."%%%".
_raw | table _raw
```

props.conf

```
[import_data]
DATETIME_CONFIG = CURRENT
TRANSFORMS-extract-metadata = drop_header, extract_metadata_copy_to_meta, reassign_meta_to_metadata,
remove_metadata_from_raw
# Splunk encodes quotes for CSV output, we need to undo this
SEDCMD-strip_double_quotes = s/""/"/g
```

transforms.conf

```
[drop_header]
# the header field form a Splunk CSV export starts with the first row being named after the header _raw. We want to
drop these
INGEST_EVAL = queue=if(_raw="\"_raw\"","nullQueue", queue)

[extract_metadata_copy_to_meta]
# we use REGEX to pop out the values for index, host, sourcetype & source, we then write them to temporary variables
in _meta. We assume that % is not found in the primary keys to optimize the REGEX
# alternatively, this can be done using INGEST_EVAL and split() function!
SOURCE_KEY=_raw
WRITE_META = true
REGEX = ^"\d+(?:\.\d+)?%%%([^%]+)%%%([^%]+)%%%([^%]+)%%%([^%]+)%%%
FORMAT = my_index::"$1" my_host::"$2" my_source::"$3" my_sourcetype::"$4"

[reassign_meta_to_metadata]
# copy the temporary user defined fields into the primary metadata locations and then delete the temporary fields
INGEST_EVAL = host:=my_host, source:=my_source, index:=my_index, sourcetype:=my_sourcetype, my_host:=null(),
my_source=null(), my_index:=null(), my_sourcetype:=null()

[remove_metadata_from_raw]
# extract the _raw field from the protocol and write back to _raw
INGEST_EVAL = _raw=replace(_raw, "^[^%]+%%%(?:[^%]+)%%%(?:[^%]+)%%%(?:[^%]+)%%%(?:[^%]+)%%%(.*)\"","\1")
```

9. REGEX Indexed Field Extraction

By default Splunk ingests data with its universal indexing algorithm which is a general-purpose tokenization process based around major and minor breakers.

However some log data is in a consistently named with value attribute pairs and in this instance, we can use REGEX transforms with **REPEAT_MATCH = true** to implement something similar to "INDEXED_CSV" and "INDEXED_JSON" but for logs.

We disable major breakers and write REGEX expressions that find value attribute pairs in the following forms `a="b", a=b, a='b'` and write out `a::b` into _*meta* to create an indexed field with the name "a" and value "b"

9. Example of a “Well Formed” Log File

Each of the value attribute pairs can be convert via a REGEX transform to indexed fields. Lots of log files follow this pattern, including the splunkd **metrics.log**

```
2020-08-26 16:34:14 group=no_quotes average=0.19585154741998068 group='single quotes'
2020-08-27 03:08:24 sum=8778 group="double quotes"
2020-08-27 17:07:50 average=0.4927135575360 name=no_quotes group="double quotes" sum=9288
2020-08-28 02:24:03 group='single quotes' label=no_quotes
2020-08-28 13:59:41 average=0.99297660285044498 name="double quotes"
2020-08-29 16:24:52 label=no_quotes name='single quotes'
2020-08-29 16:48:54 average=0.057624992829093724 sum=9092 sum=7238
2020-08-30 01:42:15 sum=6435 average=0.3208281756906822 average=0.5810043305482762
2020-08-31 05:06:31 name='single quotes' label=no_quotes
```

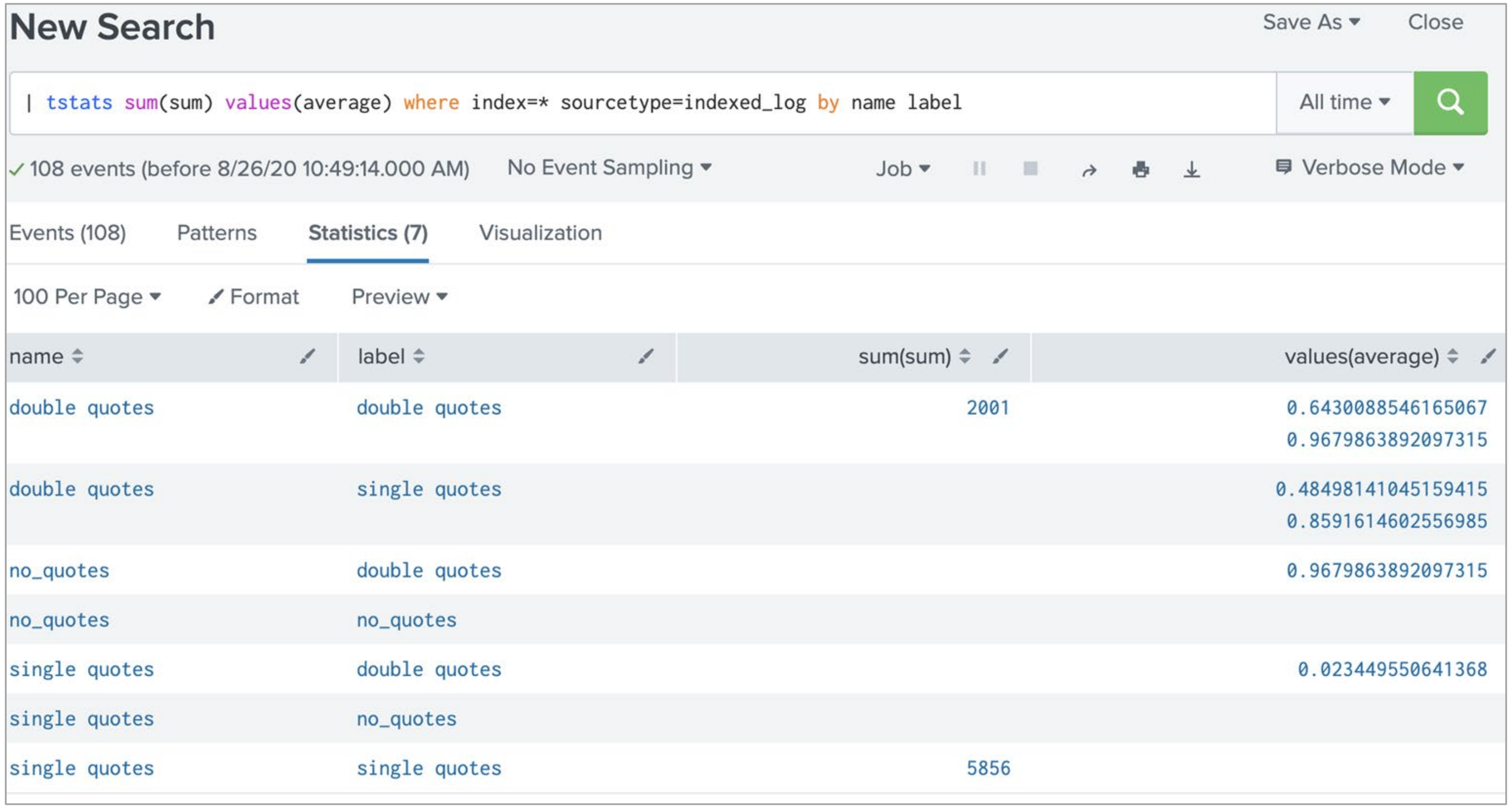
props.conf

```
# this sourcetype is an example for how we can use REPEAT_MATCH and regex to automatically extract fields from log files
[indexed_log]
TIME_FORMAT = %Y-%m-%d %H:%M:%S
SHOULD_LINEMERGE = false
LINE_BREAKER = ([\n\r]+)
TRANSFORMS-extract_indexed_fields = regex_extract_doubled_quoted_av_pairs, regex_extract_single_quoted_av_pairs,
regex_extract_unquoted_av_pairs
```

transforms.conf

```
# this regex finds single quoted attribute value pairs, ie the form a="b", and appends them to _meta
[regex_extract_doubled_quoted_av_pairs]
SOURCE_KEY = _raw
REGEX = \s([a-zA-Z][a-zA-Z0-9_-]+)="([^"]+)"
REPEAT_MATCH = true
FORMAT = $1::"$2"
WRITE_META = true

# this regex finds single quoted attribute value pairs, ie the form a=b, and appends them to _meta
[regex_extract_unquoted_av_pairs]
SOURCE_KEY = _raw
REGEX = \s([a-zA-Z][a-zA-Z0-9_-]+)=([^\s"',]+)
REPEAT_MATCH = true
FORMAT = $1::"$2"
WRITE_META = true

# this regex finds single quoted attribute value pairs, ie the form a='b', and appends them to _meta
[regex_extract_single_quoted_av_pairs]
SOURCE_KEY = _raw
REGEX = \s([a-zA-Z0-9_-]+)='([^']+)'
REPEAT_MATCH = true
FORMAT = $1::"$2"
WRITE_META = true
```

9. Accessing Our Dynamically Created Fields



```
| tstats sum(sum) values(average) where index=* sourcetype=indexed_log by name label
```

✓ 108 events (before 8/26/20 10:49:14.000 AM) No Event Sampling

name	label	sum(sum)	values(average)
double quotes	double quotes	2001	0.6430088546165067 0.9679863892097315
double quotes	single quotes		0.48498141045159415 0.8591614602556985
no_quotes	double quotes		0.9679863892097315
no_quotes	no_quotes		
single quotes	double quotes		0.023449550641368
single quotes	no_quotes		
single quotes	single quotes	5856	

With the fields automatically converted into indexed fields via REGEX we can do computation on our log file entirely with `tstats` providing high speed computation.

Note how over precision in the numeric values will bloats the size TSIDX file due to high cardinality.

When dealing with high precision metrics indexes are superior as they store numbers as numbers.

10. Complex Selective Encryption Routing

There is often a need to obfuscate the data prior to storage in Splunk, but in some scenarios still give a possibility to reverse the obfuscation.

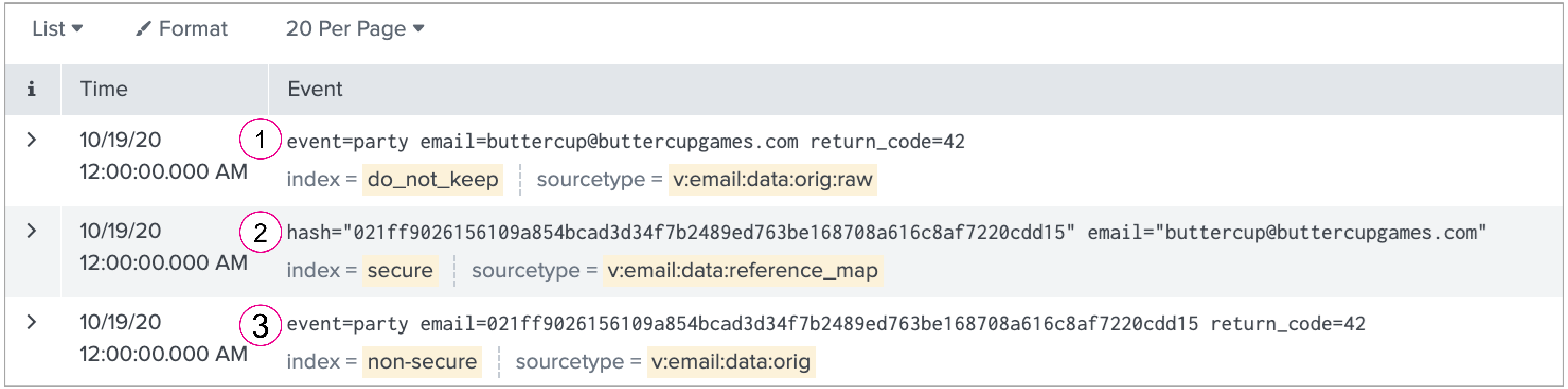
Concept of a low and high security index can be had, where general reporting occurs on obfuscated *"low security"* dataset, but for select few access to *"high security"* data set can be granted to perform the "reversal".

Also allow for different retentions, where for first 30 days reversal is possible, however, post that the reversal key is removed.

Think compliance and regulatory use-cases, especially in financial and health industries, also **GDPR** use-cases in EU.

Use of **INGEST_EVAL** and **CLONE_SOURCETYPE** makes this possible!

10. Complex Selective Encryption Routing



i	Time	Event
>	10/19/20 12:00:00.000 AM	1 `event=party email=buttercup@buttercupgames.com return_code=42` index = do_not_keep sourcetype = v:email:data:orig:raw
>	10/19/20 12:00:00.000 AM	2 `hash="021ff9026156109a854bcad3d34f7b2489ed763be168708a616c8af7220cdd15" email="buttercup@buttercupgames.com"` index = secure sourcetype = v:email:data:reference_map
>	10/19/20 12:00:00.000 AM	3 `event=party email=021ff9026156109a854bcad3d34f7b2489ed763be168708a616c8af7220cdd15 return_code=42` index = non-secure sourcetype = v:email:data:orig

splunk> .conf20

10. Complex Selective Encryption Routing

props.conf

```
# data comes in on this sourcetype
[v:email:data:orig]
TRANSFORMS-clone_data = v_hash_make_clone, v_hash_make_mask

# map reference data is created here
[v:email:data:reference_map]
TRANSFORMS-make_map_reference = v_hash_make_map_reference
```

transforms.conf

```
# this will clone the event for future processing as reference map event
[v_hash_make_clone]
REGEX = (.*)
CLONE_SOURCETYPE = v:email:data:reference_map

# this will re-write raw to replace email with sha256 hash for "low security" index
[v_hash_make_mask]
INGEST_EVAL = email_hash=sha256(replace(_raw, "^(.*)email=(\S+)(.*)$", "\2")), _raw=replace(_raw, "^(.*email)=(\S+)(.*)$",
"\1=".email_hash."\3")

# this transform routes data, and emits reference map _raw for "high security" index
[v_hash_make_map_reference]
INGEST_EVAL = index=secure, queue=if(match(_raw, "email="), "indexQueue", "nullQueue"), email_hash=sha256(replace(_raw,
"^(.*)email=(\S+)(.*)$", "\2")), _raw="hash=\"".email_hash."\" email=\"".replace(_raw, "^(.*)email=(\S+)(.*)$", "\2")."\""
```

All The Examples and Sample Data Can Be From This Deck Can Be Found at

https://github.com/silkyrich/ingest_eval_examples

Thank You

Please provide feedback via the

SESSION SURVEY