Everything you didn't know about Metrics in Splunk Platform

DEV1136

Forward-looking statements

This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's current expectations, estimates and assumptions based on the information currently available to us. These forward-looking statements are not guarantees of future performance and involve significant risks, uncertainties and other factors that may cause our actual results, performance or achievements to be materially different from results, performance or achievements expressed or implied by the forward-looking statements contained in this presentation.

For additional information about factors that could cause actual results to differ materially from those described in the forward-looking statements made in this presentation, please refer to our periodic reports and other filings with the SEC, including the risk factors identified in our most recent quarterly reports on Form 10-Q and annual reports on Form 10-K, copies of which may be obtained by visiting the Splunk Investor Relations website at www.investors.splunk.com or the SEC's website at www.sec.gov. The forward-looking statements made in this presentation are made as of the time and date of this presentation. If reviewed after the initial presentation, even if made available by us, on our website or otherwise, it may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statement based on new information, future events or otherwise, except as required by applicable law.

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. We undertake no obligation either to develop the features or functionalities described, in beta or in preview (used interchangeably), or to include any such feature or functionality in a future release.

splunk> .conf25

Everything you didn't know about Metrics in Splunk Platform

Brett

Adams

Splunk Practice Lead | Deloitte Australia
SplunkTrust and Splunk MVP

.conf25

splunk>

Metrics 1.01

Special index type that stores **floating point** values and string dimensions.

Really good for performing **super fast** statistical aggregation and analysis.

Really bad for showing the raw data / all data recorded data points.

A brief history of Metrics in Splunk

Splunk .conf17

Splunk 7.0

- Introduction of metric index
- 150 byte ingest license usage per event

Splunk .conf18

Splunk 7.1-7.2:

- Optimised dimension storage
- Improved search performance

Splunk 7.3:

- Metric rollups
- Max 150 byte ingest license usage

Splunk .conf19

Splunk 8.0

- Multi-metric format
- Histogram support
- Major performance and storage improvement
- Floating point value compression

Splunk .conf22 to now

Splunk 9.0

- Federated Search support for searching metric indexes

Splunk 9.3

- Federated Search support for mcollect

Splunk 9.4

- Federated Search support for mcatalog

A brief history of *my* Metrics journey

Forza Telemetry

July 2019

Ingests UDP structs from the Forza series of video games into single mode metrics.

Extremely costly due to fixed 150 byte license model

Torque

December 2019

Ingests HTTP query strings send by the Torque Android app which collects OBD2 data from vehicles.

Created to experiment with multi-metric format

Racing Telemetry

February 2020

Ingests UDP structs from Forza Horizon 4+, Forza Motorsport 7+, Project Cars 2, and F1 2019+.

Created to test sub-second timestamps and creating hundreds of metrics per second

See Conf22 OBS1157B

Better Perfmon Metrics

August 2022

An improved method of ingesting windows performance metrics.

Reduces ingest license usage by 50%

See Conf23 PLA1163C

Multi-Metric Perfmon

June 2022

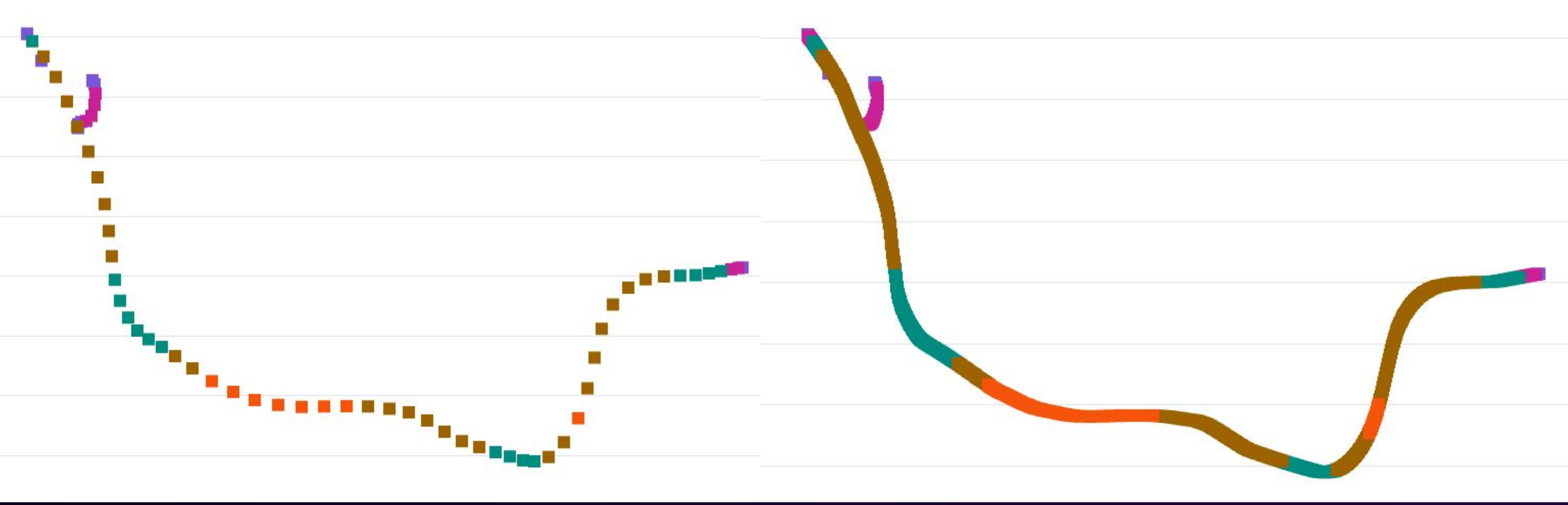
An experiment in parsing Windows perfmon in the multi-metric format using middleware.

Reduces ingest license usage by 90% at the cost of client CPU usage.

Everything you didn't know about Metrics

#1 - Timestamp Resolution

- Whole second _time resolution by default
- Opt in to millisecond resolution at the cost of search performance.
- Useful only when you have data that is read multiple times per second



#1 - Timestamp Resolution

indexes.conf

```
metric.timestampResolution = <s|ms>
```

Example from
.conf22 OBS1157B

#2 - Licensing

Event indexes are licensed on:

_raw data in bytes

Metric indexes are licensed on:

_raw data in bytes, *plus 18, up to a maximum of 150**

This means events less than 150 bytes are cheaper in event indexes than metrics.

#2 - Licensing

Example Perfmon Event

04/20/2023 12:03:06.708 +1000
collection=CPU
object=Processor
counter=% Processor Time
instance=1
Value=38.88121747891

Cost as Event: 113 bytes

Cost as Metric: 131 bytes

Actually Useful information: 43 bytes

Processor.% Processor Time 1 38.88121747891

Using Metrics was worse!

Example JSON Event

```
{
   instance: _Total
   metric_name:Processor.%_C1_Time: 3.0375430950361033
   metric_name:Processor.%_C2_Time: 83.9737923052476
   metric_name:Processor.%_C3_Time: 0
   metric_name:Processor.%_DPC_Time: 0.31212002508146586
   metric_name:Processor.%_Idle_Time: 87.0113354002837
   … 10 more rows
}
```

Cost as Event: 840 bytes

Cost as Metric: 150 bytes

Using Metrics was better!

#3 - Disk Usage

Metric indexes have been heavily optimized to be as efficient as possible and improve performance.

Metrics can use **four times less disk space** than events

(your results will vary)

#3 - Disk Usage

Example Perfmon Event

04/20/2023 12:03:06.708 +1000
collection=CPU
object=Processor
counter=% Processor Time
instance=1
Value=38.88121747891

Average disk usage per Event: 200 bytes

Average disk usage per Metric: 45 bytes

Using Metrics was better!

See .conf23 PLA1163C for more details

#4 - Raw Journal

Event indexes retain all _raw data in their compressed raw journal, which is shown at search time.

Metric indexes only retain the raw journal in indexer clusters

```
metric.stubOutRawdataJournal = <boolean>
* For metrics indexes only.
* Determines whether the data in the rawdata file is deleted when the hot bucket
  rolls to warm. The rawdata file itself remains in place in the bucket.
* This setting does not take effect for indexes that have replication enabled ("repFactor=auto")
  in an indexer cluster deployment.
```

#5 - DDAS & DDAA

DDAS = Dynamic Data: Active Searchable

DDAA = Dynamic Data: Active Archive

Despite the difference in license and disk usage, your DDAS & DDAA usage in Splunk Cloud will be the **same** as event indexes

However, a reduction to _raw will also save you Dynamic Data

#5 - DDAS & DDAA

Unmodified Perfmon Event

04/20/2023 12:03:06.708 +1000
collection=CPU
object=Processor
counter=% Processor Time
instance=1
Value=38.88121747891

Raw size: 113 bytes

Modified Perfmon Event

Processor.%_Processor_Time 1 38.881

Raw size: 35 bytes (3x smaller)

Both events are parsed as:

```
metric_name::Processor.%_Processor_Time
instance::1
_value::38.881
```

(technically has lower precision)

Still with me?

It’s about to get complicated

#6 - Strings.data and merged_lexicon.lex

When data isn’t available in _raw, Splunk uses Strings.data and merged_lexicon.lex to store terms.

These files are **not** compressed, so a large number of unique strings can result is extraordinarily large buckets.

(foreshadowing)

Strings.data

```
1       instance          0     0       0       0
2       metric_name:Processor.%_Idle_Time          0…
3       metric_name:Processor.%_Interrupt_Time     0…
4       metric_name:Processor.%_Privileged_Time    0…
5       metric_name:Processor.%_Processor_Time     0…
6       metric_name:Processor.%_User_Time          0…
7       pod       0     0       0       0
8       rack      0     0       0       0
9       zone      0     0       0       0
```

JSON Metric Test Cases			
	Raw	Disk	Ingest License
Large JSON Metric	870 bytes	291 bytes	150 bytes

#7 - Using HTTP Event Collector

HEC payloads let you set the time, index, host, source, sourcetype, and indexed fields in addition to the raw event.

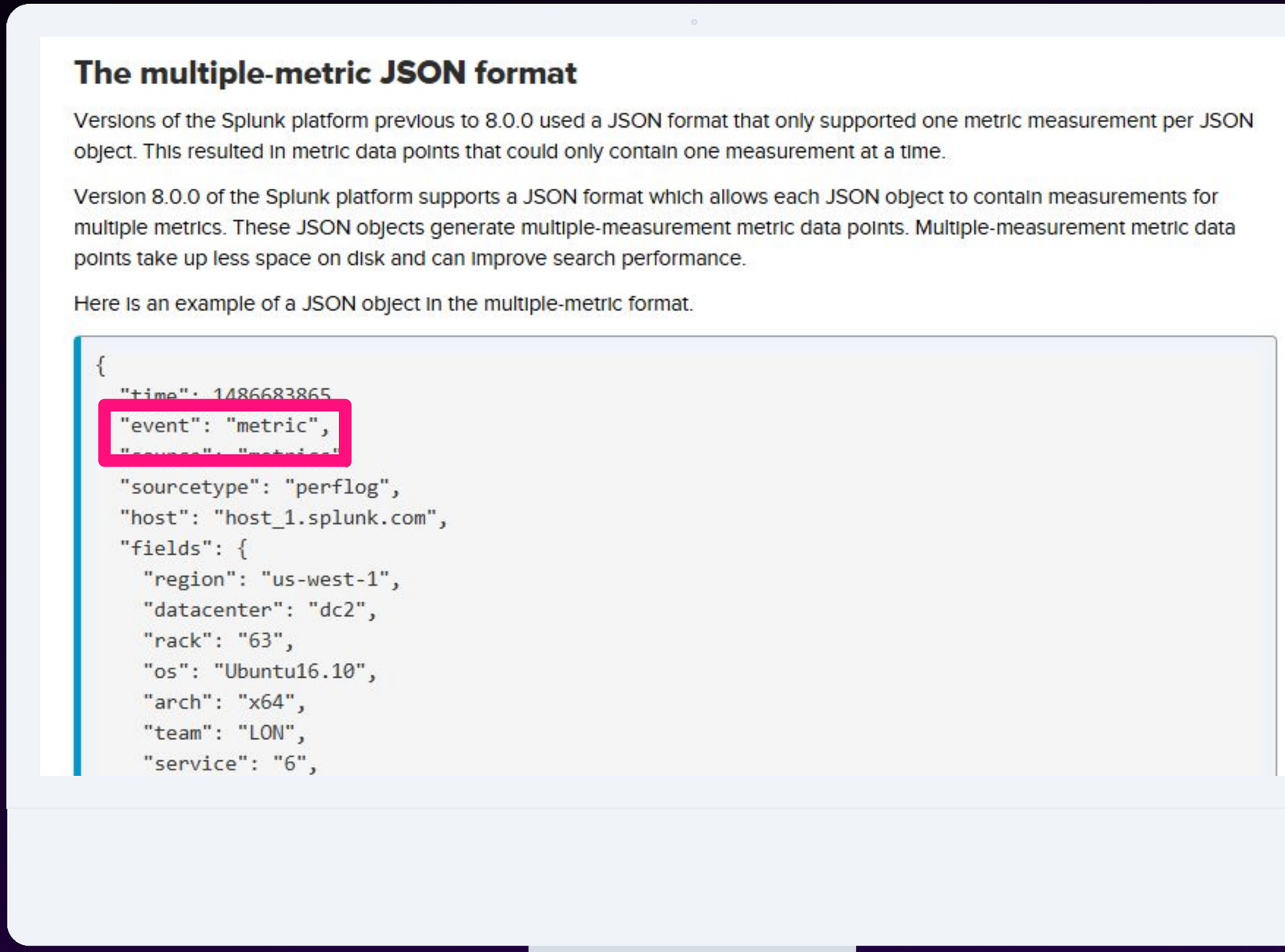
The event field cannot be blank for event indexes, but it can for metric indexes, however general advice has always been to set this to a short string.

Third party data pipeline software also follows this pattern.

Docs

The multiple-metric JSON format

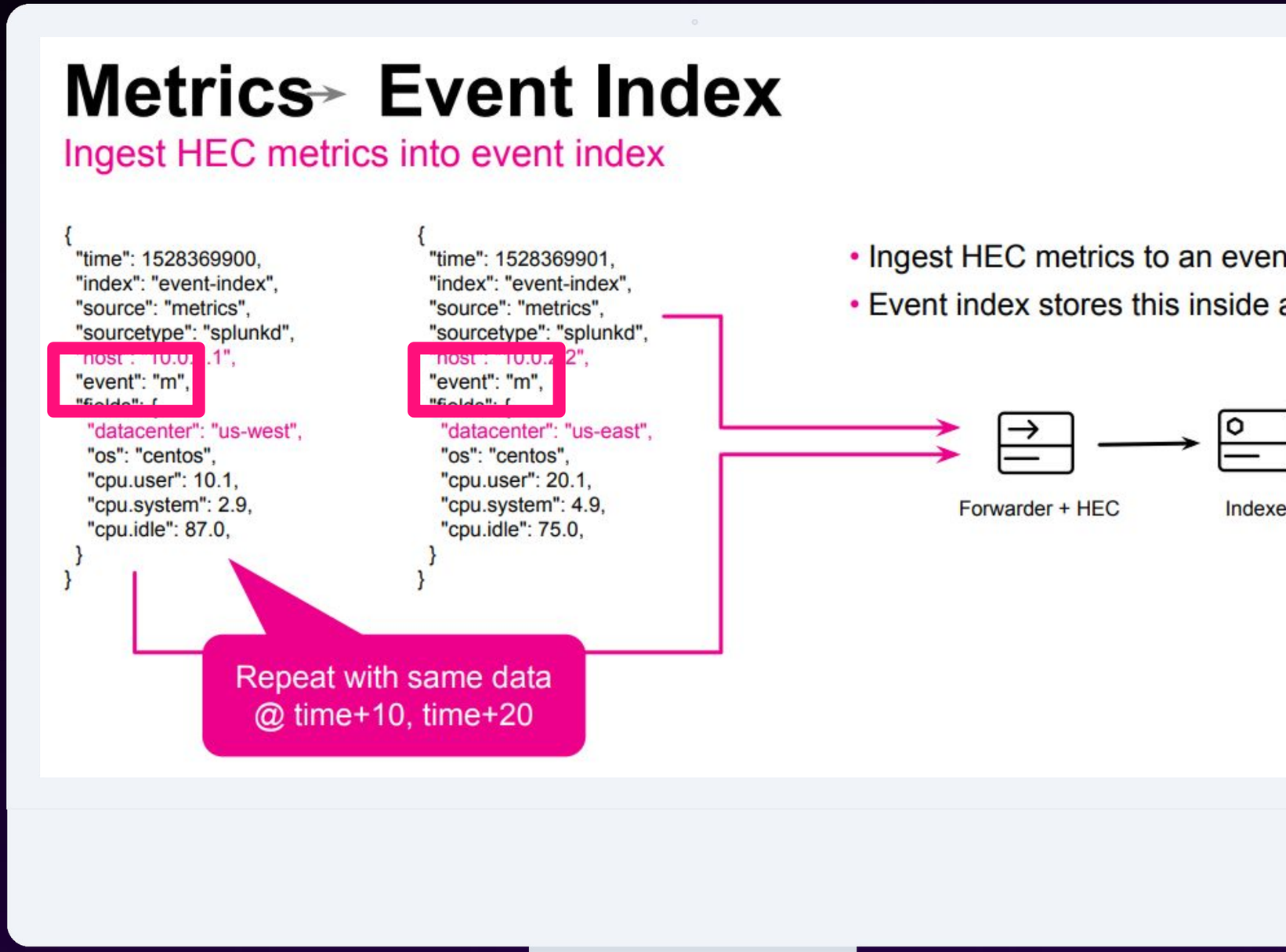
https://docs.splunk.com/Documentation/Splunk/9.4.0/Metrics/GetMetricsInOther#Get_metrics_in_from_clients_over_HTTP_or_HTTPS



Conf19

FN2268

https://conf.splunk.com/files/2019/slides/FN2268.pdf



This *can* be bad

Impact to Strings.data

Using “event”

1	instance	0	0	0	0
2	metric_name:Processor.%_Idle_Time	0…			
3	metric_name:Processor.%_Interrupt_Time	0…			
4	metric_name:Processor.%_Privileged_Time	0…			
5	metric_name:Processor.%_Processor_Time	0…			
6	metric_name:Processor.%_User_Time	0…			
7	pod	0	0	0	0
8	rack	0	0	0	0
9	zone	0	0	0	0

Using “fields”

1	instance	0	0	0	0
2	_Total	0	0	0	0
3	metric_name:Processor.%_Idle_Time	0…			
4	metric_name:Processor.%_Interrupt_Time	0…			
5	metric_name:Processor.%_Privileged_Time	0…			
6	metric_name:Processor.%_Processor_Time	0…			
7	metric_name:Processor.%_User_Time	0…			
8	pod	0	0	0	0
9	pod0	0	0	0	0
10	rack	0	0	0	0
11	rack0	0	0	0	0
12	zone	0	0	0	0
13	zone0	0	0	0	0
14	pod1	0	0	0	0
15	rack1	0	0	0	0
16	pod2	0	0	0	0

… 510 rows truncated

#7 - Using HTTP Event Collector

Using the **Fields** key uses less ingest license, less disk, and less dynamic data, as long as dimensions dont have very high cardinality

Using the **Event** key (and _json sourcetype) works the same as fields key without the dimensions problem, however it will use up to 150 bytes of ingest license, and full _raw size against dynamic data quota.

JSON Metric Test Cases

	Raw	Disk	Ingest License
Large JSON Metric	870 bytes	291 bytes	150 bytes
Using HEC fields	7 bytes	201 bytes	25 bytes
Using HEC fields without Event	zero	202 bytes	18 bytes

#8 - Metric Schemas

Using the `metric_name:` prefix makes writing metrics easy and flexible

Technically costs more ingest, and raw

You can save 12 bytes per metric value by using a metric transform instead.

(does not save disk space)

transforms.conf

```
[metric-schema:<name>]
METRIC-SCHEMA-MEASURES = _ALLNUMS_
```

JSON Metric Test Cases

	Raw	Disk	Ingest License
Large JSON Metric	870 bytes	291 bytes	150 bytes
Using HEC fields	7 bytes	201 bytes	25 bytes
Using HEC fields without Event	zero	202 bytes	18 bytes
Using Metric Schema	690 bytes	291 bytes	150 bytes

#9 - Transforms

Size of the metric _raw contributes to:

- Ingest license usage
- Disk usage (in clusters)
- DDAS & DDAA usage

But we never see or use _raw after ingest

Using HEC we don’t have to include a raw

So can you just…. *remove _raw?*

transforms.conf

```
[no_raw]
INGEST_EVAL = _raw=""
```

#9 - Transforms

No you cannot! _raw cannot be empty

But it *can* be set to a very small string

transforms.conf

```
[no_raw]
INGEST_EVAL = _raw="x"
```

This means each metric event *should* cost:

- 19 bytes of ingest*
- 1 byte of dynamic data

JSON Metric Test Cases

	Raw	Disk	Ingest License
Large JSON Metric	870 bytes	291 bytes	150 bytes
Using HEC fields	7 bytes	201 bytes	25 bytes
Using HEC fields without Event	zero	202 bytes	18 bytes
Using Metric Schema	690 bytes	291 bytes	150 bytes
With Transformed Raw	1 byte	720 bytes	1 byte

These values are unexpected

Impact to Strings.data

Using Metric Schema

```
1       instance          0     0       0       0
2       metric_name:Processor.%_Idle_Time         0…
3       metric_name:Processor.%_Interrupt_Time    0…
4       metric_name:Processor.%_Privileged_Time   0…
5       metric_name:Processor.%_Processor_Time    0…
6       metric_name:Processor.%_User_Time         0…
7       pod       0     0       0       0
8       rack      0     0       0       0
9       zone      0     0       0       0
```

Using _raw="x"

```
1       instance          0     0       0       0
2       _Total    0     0       0       0
3       metric_name:Processor.%_C1_Time   0     0       0
4       9.535637105338008         0     0       0       0
5       metric_name:Processor.%_C2_Time   0     0       0
6       23.8468737414914          0     0       0       0
7       metric_name:Processor.%_C3_Time   0     0       0
8       metric_name:Processor.%_DPC_Time          0     0
9       0.39917205288963686       0     0       0       0
10      metric_name:Processor.%_Idle_Time         0     0
…
1562564 5467.029814816554                 0       0       0
1562565 36039.85460036337         0     0       0       0
1562566 810.8526232291038         0     0       0       0
1562567 31556.14463378248         0     0       0       0
```

So now what

Go check your metrics

Check if you could be optimising their _raw to reduce your costs

Check how much ingest license your being charged

Check how much DDAS/DDAA or disk they consume

Go check your HEC

Are you using Fields or Event?

Are your bucket sizes/counts healthy? (if using fields)

Could you be saving cost by removing the event key?

Go attend PLA1078

Your an expert in Splunk Metrics now, so go learn how to optimise your event indexes too in my session tomorrow at 1PM:

Get MORE data in.
Optimize Data Ingest: Practical Tips to Maximize Splunk's Value

#10 - mstats rocks!

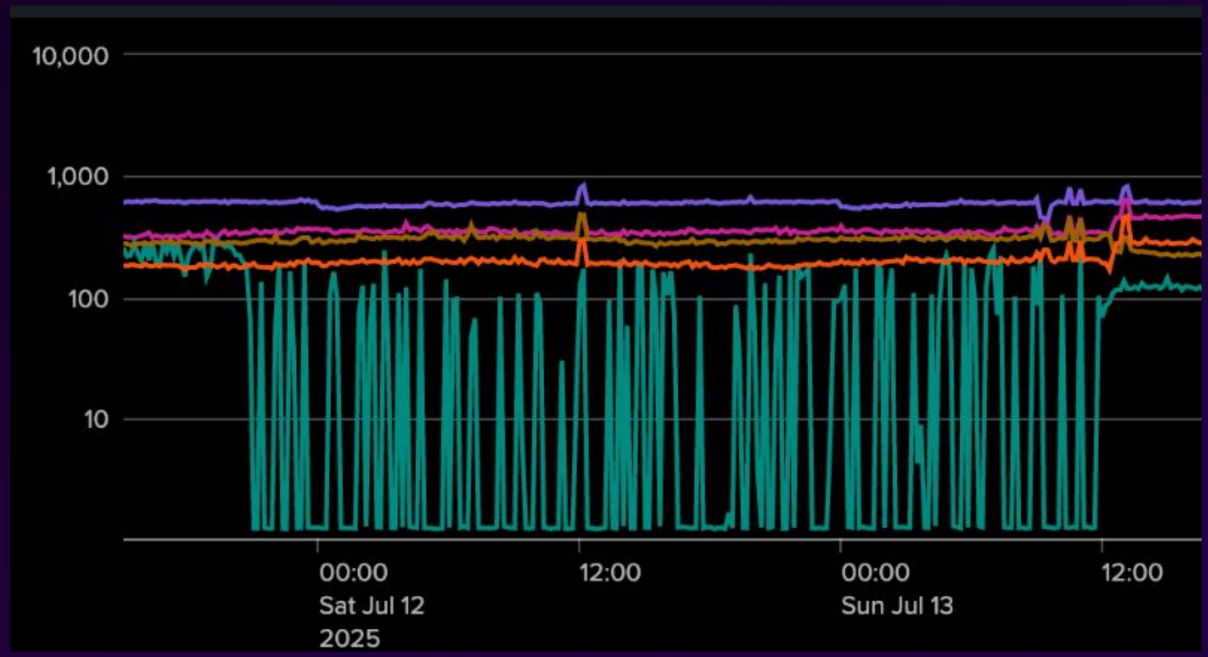
As one of the more recently added search commands, mstats lots of helpful features.

chart=true
No need to | xyseries or even | timechart when mstats can write a chart directly

span=1d
No need to by _time, just tell mstats your span

And more like prestats, append, fillnull_value, and realtime controls backfill & update_period

| mstats avg(duration) where index=foo by host span=600s chart=t



Thank you