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> .conf23

Maximizing Splunk® SPL™

Foreach and the Power of Iterative, Templatized Evals

PLA1881C

Ryan Wood

Sr. Splunk Solutions Architect
GuidePoint Security

splunk> .conf23

Who are you?

Architect
Engineer
Analyst
Enthusiast

… that writes SPL queries.

Someone who wants to…

- Simplify your queries.
- Build confidence in your data.
- Learn methods to more easily demonstrate and share insights.
- Nerd out together on some neat SPL.

Base Knowledge Pre-Reqs

to get the most out of this presentation…

- Familiar with the concept of programmatic "loops"
- Intermediate understanding of SPL search optimization

Objectives

What are the goals that drove this presentation?

A bit of a surprise…

- In May 2023, `foreach` ranked #21 out of 147 commands in terms of usage *by users* across all Splunk Cloud™ stacks[1]
- While researching, I found less than 20 total example queries using `| foreach` publicly available across Github + Major Search Engines (first 3 pages)

[1] Based on telemetry statistics provided by Splunk

Objectives

What are the goals that drove this presentation?

A bit of a surprise…

- In May 2023, `foreach` ranked #21 out of 147 commands in terms of usage *by users* across all Splunk Cloud™ stacks[1]
- While researching, I found less than 20 total example queries using `| foreach` publicly available across Github + Major Search Engines (first 3 pages)

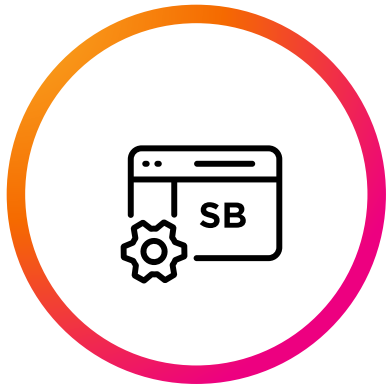
Introduce

Establish the behavior of `| foreach` & how to think about it's role.

Inspire

Walk through a series of methods utilizing `| foreach` in increasing complexity.

Enable



Provide examples and sample code to utilize in your own environment.

[1] Based on telemetry statistics provided by Splunk

Agenda

`| foreach`

more `| foreach`

even more `| foreach`

1) Introductions
   - Background
   - Objectives
   - Overview of Technical Topics
2) Framing the Presentation Style & Content
3) foreach Command Overview
   - Walkthrough of modes
4) Usecases & Examples
   - Highlight: Handling Zero-Length Fields
   - Highlight: Compiling Fields for Review
5) Wrap-up & Resources

So who is this guy?

Ryan Wood

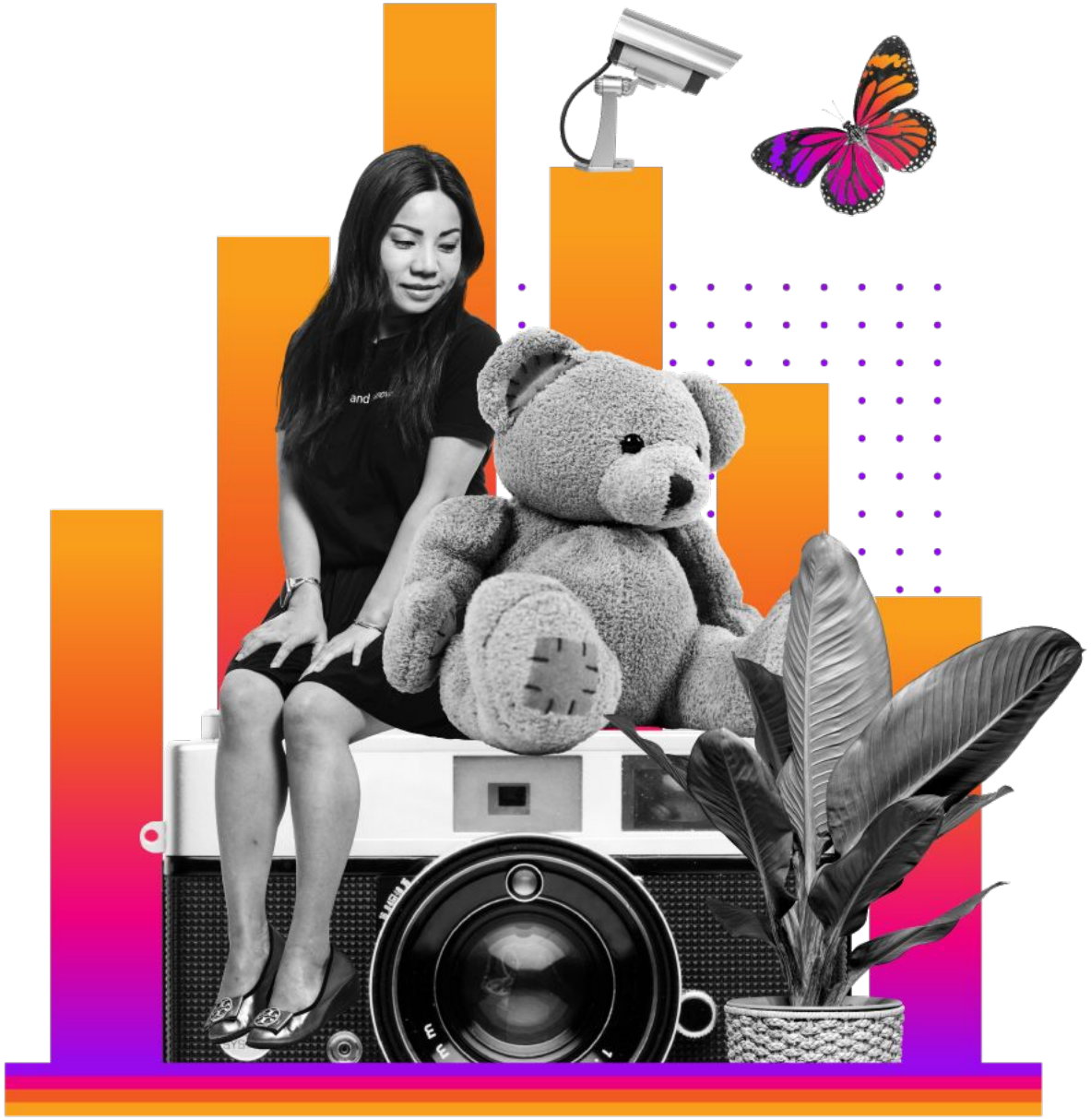
Sr. Splunk Solutions Architect
GuidePoint Security

Ryan Wood

Background

- Certifications:
  - Splunk Core Certified Consultant
  - Splunk Enterprise Security Certified Admin
  - Security+
- Daily Splunk User for over 10 years
- Delivering Splunk Professional Services for over 5 years
- Passionate about enabling others to find success

"The only time I don't enjoy my job is when I'm doing repetitive, tedious things."



Let's Get Into It

Framing the Presentation Style & Content

splunk> .conf23

Framing the Presentation Style

Dynamic Thresholds

Validation

Compilation

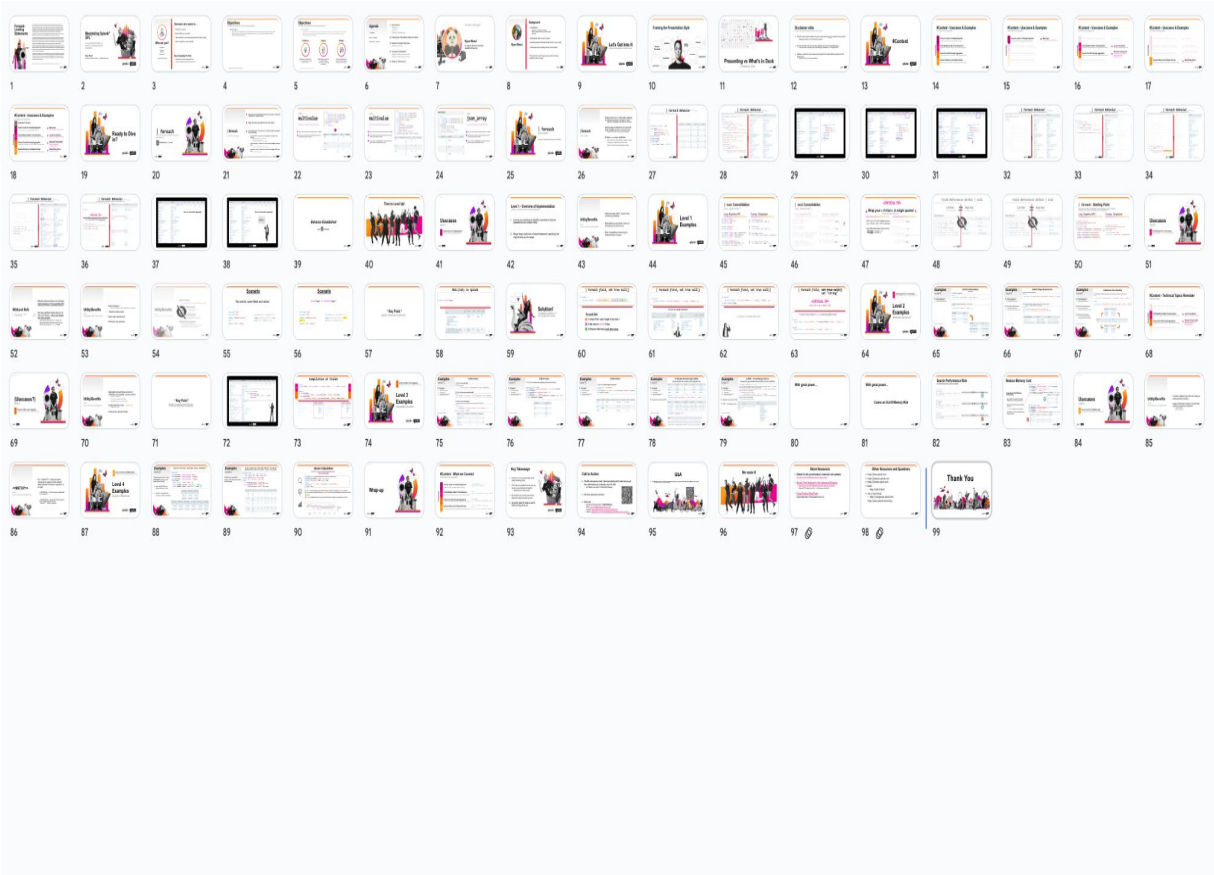
<<MATCHSTR>>

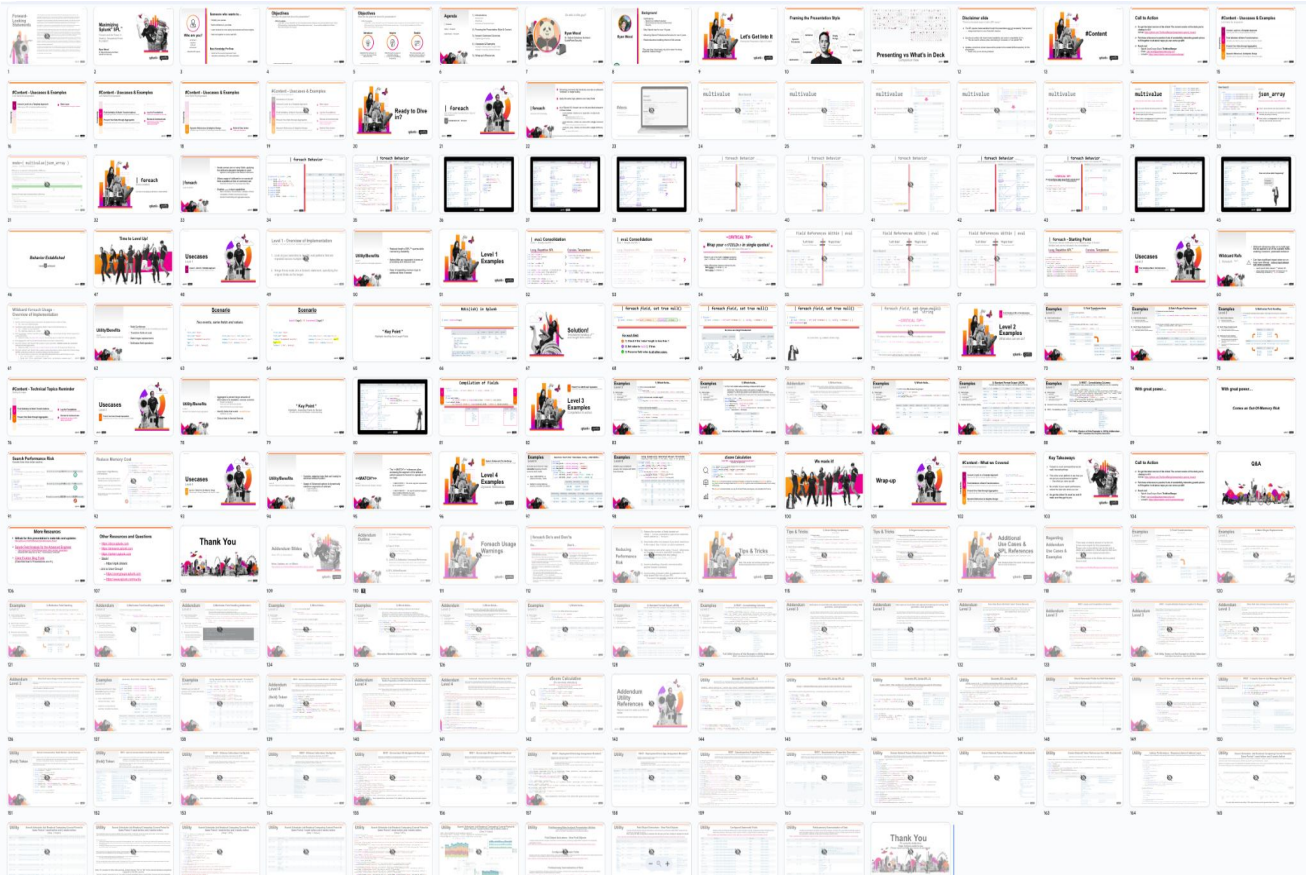
Empty Strings

zScores

Transformation

Aggregation





Presenting vs What's in Deck

Comparison View (v2.0)

splunk> .conf23

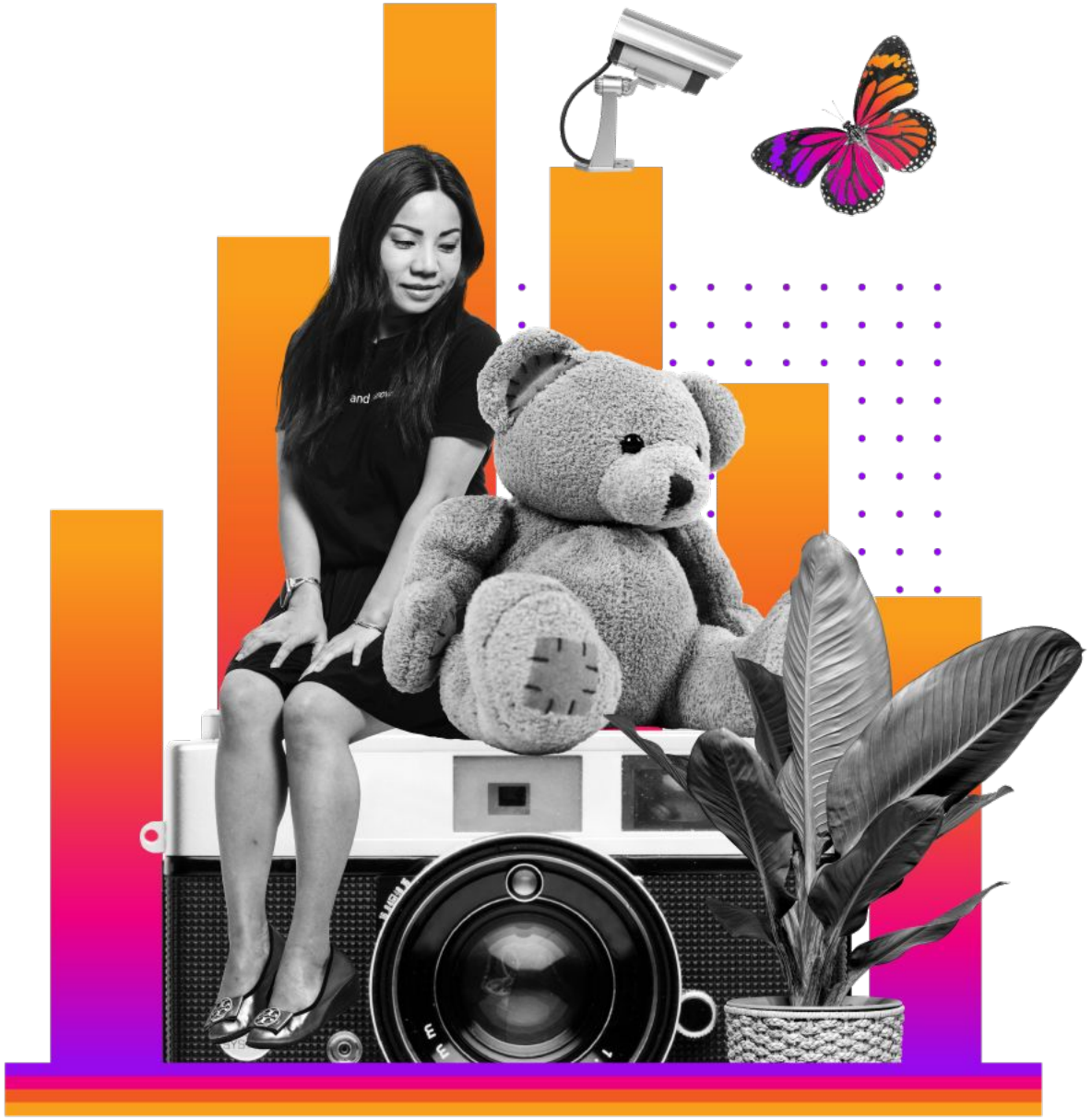
Disclaimer slide

"There's a thousand ways to build a SPL query."

- The SPL queries made available through this presentation are not necessarily "best practice".
  - Always learn/test on a non-Production instance

- Queries are written with intent toward readability and version compatibility (8.2+)
  - This can result in verbose syntax, odd ordering of commands, or "non-optimal" SPL
  - There are eval comment lines throughout - old habits die hard (and I still work with Splunk 7 orgs)

- Updates, corrections, known issues will be posted to the related GitHub repository for this presentation.
  - Reach out if you run into any problems!



#Content

splunk> .conf23

Call to Action

- **Go get the latest version of the slides! The current version of the deck you're viewing is v2.0**
  GitHub: https://github.com/TheWoodRanger/presentation-splunk_foreach

- **Put those references to practice! Lots of wonderfully interesting puzzle pieces to fit together in whatever ways you can come up with!**

- **Reach out!**
  - Splunk UserGroups Slack: **TheWoodRanger**
  - Email: ryan.wood@guidepointsecurity.com
  - LinkedIn: https://www.linkedin.com/in/ryanwoodranger/

#Content - Usecases & Examples

Let’s frame the progression

Complexity

1. **Convert | eval's to a Template Approach**
   Clean up your SPL by merging repetitive eval logic.
2. **Field Validation & Batch Transformations**
   Normalize your fields to ensure your logic works like it should.
3. **Present Your Data through Aggregation**
   Compile field values for review or to output in a specific format
4. **Dynamic References & Adaptive Design**
   Versatile Logic for Dynamic Thresholding & Advanced Insights

#Content - Usecases & Examples

Let’s frame the progression

Complexity

1 **Convert | eval's to a Template Approach**
Clean up your SPL by merging repetitive eval logic.

2 **Field Validation & Batch Transformations**
Normalize your fields to ensure your logic works like it should.

3 **Present Your Data through Aggregation**
Compile field values for review or to output in a specific format

4 **Dynamic References & Adaptive Design**
Versatile Logic for Dynamic Thresholding & Advanced Insights

Base Layer
The simplest form of using this command.

#Content - Usecases & Examples

Let’s frame the progression

Complexity

1
Convert | eval's to a Template Approach
Clean up your SPL by merging repetitive eval logic.

2
Field Validation & Batch Transformations
Normalize your fields to ensure your logic works like it should.

3
Present Your Data through Aggregation
Compile field values for review or to output in a specific format

4
Dynamic References & Adaptive Design
Versatile Logic for Dynamic Thresholding & Advanced Insights

Base Layer
The simplest form of using this command.

Lay the Foundations
Ensure your assumptions are accurate.

Review & Communicate
Analyze what you have available.
Make it understandable.
Identify opportunities.

#Content - Usecases & Examples

Let’s frame the progression

Complexity

1

Convert | eval's to a Template Approach

Clean up your SPL by merging repetitive eval logic.

Base Layer

The simplest form of using this command.

2

Field Validation & Batch Transformations

Normalize your fields to ensure your logic works like it should.

Lay the Foundations

Ensure your assumptions are accurate.

3

Present Your Data through Aggregation

Compile field values for review or to output in a specific format

Review & Communicate

Analyze what you have available.
Make it understandable.
Identify opportunities.

4

Dynamic References & Adaptive Design

Versatile Logic for Dynamic Thresholding & Advanced Insights

Build & Take Action

Utilize those foundations

#Content - Usecases & Examples

Let's frame the progression

Complexity

0 **Introduction of | foreach**

1 **Convert | eval's to a Template Approach**
Clean up your SPL by merging repetitive eval logic.

- **Base Layer**
  The simplest form of using this command.

2 **Field Validation & Batch Transformations**
Normalize your fields to ensure your logic works like it should.

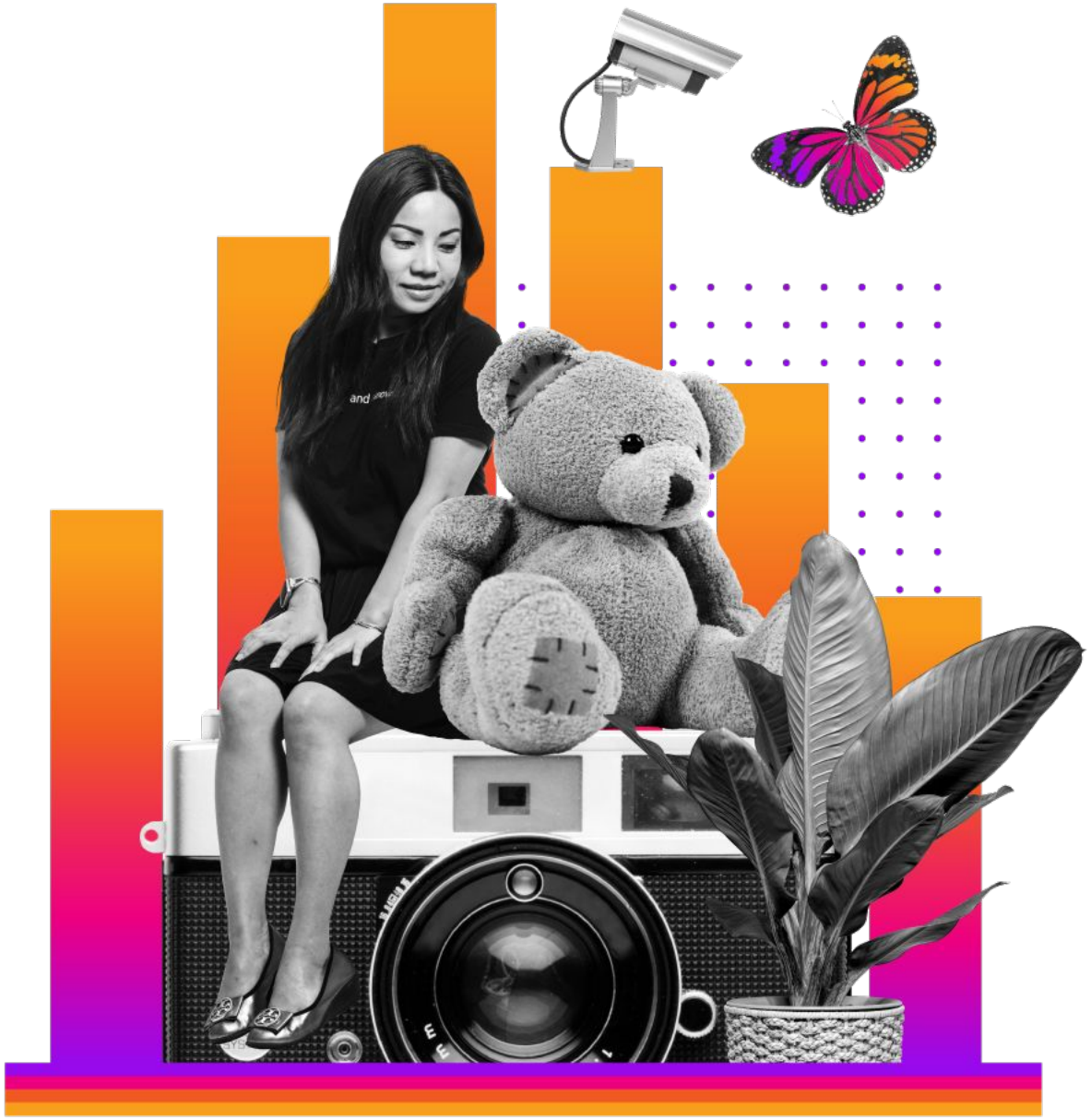
- **Lay the Foundations**
  Ensure your assumptions are accurate.

3 **Present Your Data through Aggregation**
Compile field values for review or to output in a specific format

- **Review & Communicate**
  Analyze what you have available.
  Make it understandable.
  Identify opportunities.

4 **Dynamic References & Adaptive Design**
Versatile Logic for Dynamic Thresholding & Advanced Insights

- **Build & Take Action**
  Utilize those foundations



Ready to Dive in?

splunk> .conf23

| foreach

One of the coolest commands you may not have heard of.

0 **Introduction of | foreach**

| foreach

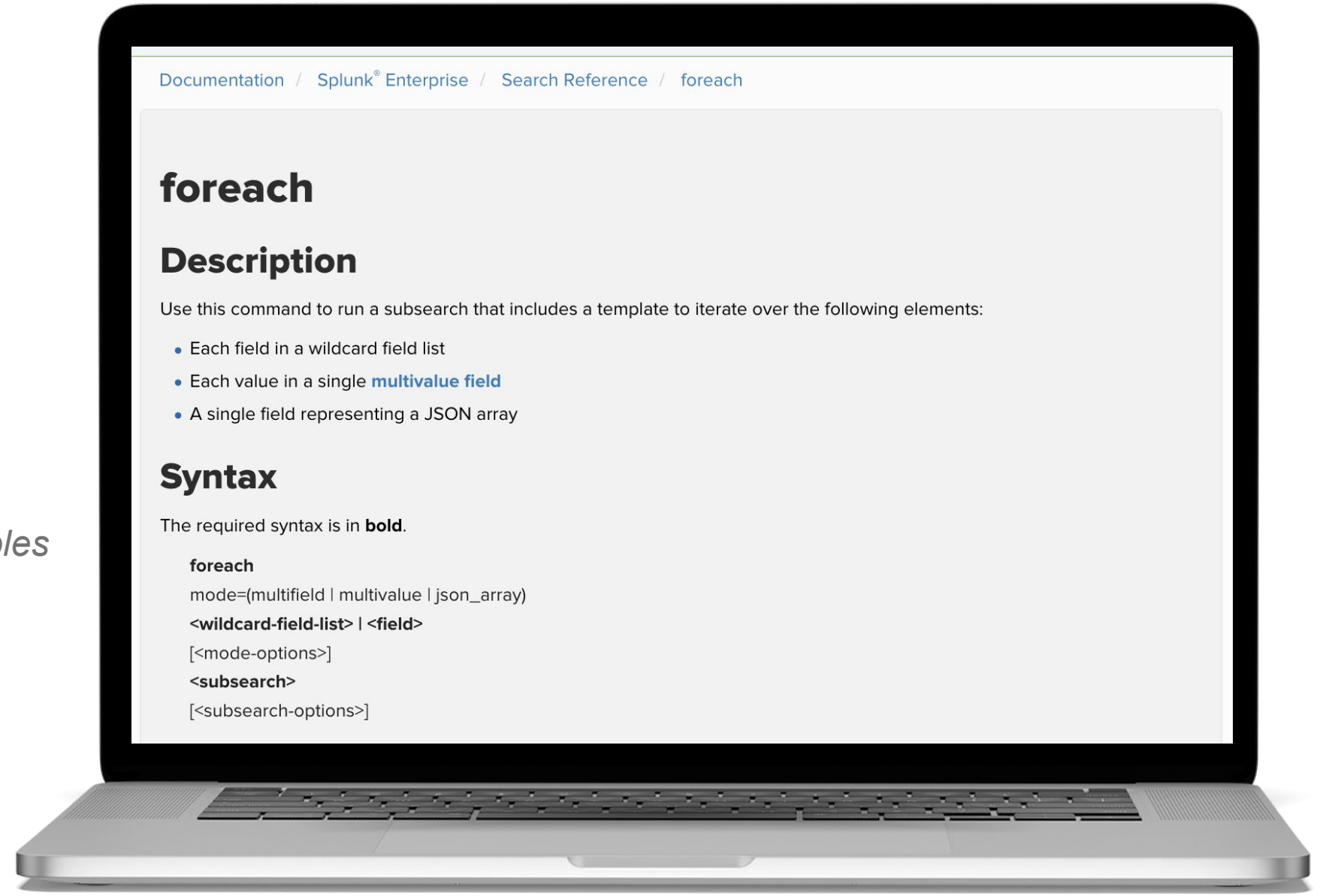
Overview of Functionality

- Streaming command that iteratively executes a subsearch “template” on target field(s).
- Apply the same logic patterns over many fields
- As of Splunk 9.0, foreach can run the provided subsearch in three modes:
  - *mode=multifield* - Iterates over a single field or multiple fields (default)
    - Allows wildcard match “*” for field names
    - Only option for versions < 9.0
    - Cannot iterate across values within a MV field
  - *mode=multivalue* - Iterates over values within a **single** multivalued field.
    - Similar to *eval mvmap()* but allows multiple evals
  - *mode=json_array* - Iterates over items within a **single** JSON array field.
    - Use with eval JSON functions[1]

[1] http://docs.splunk.com/Documentation/Splunk/latest/SearchReference/JSONFunctions

#docs

Starting point
Contains good baseline examples



https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Foreach

mode=

multivalue

Iterate across the values within a single multivalue field.

- Token to access individual value being operated on: **<<ITEM>>**
- Elements within the subsearch eval expression(s) must be of the same type (strings or numbers)

New Search

```
1  | makeresults | fields - _time
2  | eval values_mv = mvappend("1", "2", "3")
3  | eval total = 0
4  | foreach mode=multivalue values_mv
5      [eval total = total + '<<ITEM>>']
6  | table values_mv, total
```

values_mv	total
1	6
2	
3	

mode=

multivalue

Iterate across the values within a single multivalue field.

- Token to access individual value being operated on: **<<ITEM>>**
- Elements within the subsearch eval expression(s) must be of the same type (strings or numbers)

```
| union
   [| makeresults count=3 | eval teacher="James"]
   [| makeresults count=3 | eval teacher="Jessica"]
   [| makeresults count=3 | eval teacher="Roberto"]
| eval student_grades = (random() % 100) + 1
| stats values(*) AS * BY teacher
| eval sum = 0,
       iterations = 0
| foreach mode=multivalue student_grades
   [| eval sum = sum + '<<ITEM>>',
   iterations = iterations + 1]
| eval average = round(sum / iterations, 0)
| table teacher, student_grades, average, sum, iterations
```

mode=

multivalue

Iterate across the values within a single multivalue field.

- Token to access individual value being operated on: **<<ITEM>>**
- Elements within the subsearch eval expression(s) must be of the same type (strings or numbers)

```
| union
    [| makeresults count=3 | eval teacher="James"]
    [| makeresults count=3 | eval teacher="Jessica"]
    [| makeresults count=3 | eval teacher="Roberto"]
| eval student_grades = (random() % 100) + 1
| stats values(*) AS * BY teacher
| eval sum = 0,
       iterations = 0
```

teacher	student_grades
James	48
James	77
James	12
Jessica	2
Jessica	26
Jessica	95
Roberto	84
Roberto	20
Roberto	54

mode=

multivalue

Iterate across the values within a single multivalue field.

- Token to access individual value being operated on: **<<ITEM>>**
- Elements within the subsearch eval expression(s) must be of the same type (strings or numbers)

```
1  | union
2      [| makeresults count=3 | eval teacher="James"]
3      [| makeresults count=3 | eval teacher="Jessica"]
4      [| makeresults count=3 | eval teacher="Roberto"]
5  | eval student_grades = (random() % 100) + 1
6  | stats values(*) AS * BY teacher
7  | eval sum = 0,
8         iterations = 0
9  | foreach mode=multivalue student_grades
10     [| eval sum = sum + '<<ITEM>>',
11        iterations = iterations + 1]
12 | eval average = round(sum / iterations, 0)
```

teacher	student_grades	sum	iterations
James	12 48 77	0	0
Jessica	2 26 95	0	0
Roberto	20 54 84	0	0

mode=

multivalue

Iterate across the values within a single multivalue field.

```
1  | union
2      [| makeresults count=3 | eval teacher="James"]
3      [| makeresults count=3 | eval teacher="Jessica"]
4      [| makeresults count=3 | eval teacher="Roberto"]
5  | eval student_grades = (random() % 100) + 1
6  | stats values(*) AS * BY teacher
7  | eval sum = 0,
8         iterations = 0
9  | foreach mode=multivalue student_grades
10     [| eval sum = sum + '<<ITEM>>',
11     iterations = iterations + 1]
12 | eval average = round(sum / iterations, 0)
13 | table teacher, student_grades, average, sum, iterations
```

- Token to access individual value being operated on: **<<ITEM>>**
- Elements within the subsearch eval expression(s) must be of the same type (strings or numbers)
- Allows multiple | *eval* assignments to be applied to each item within the same command execution scope:

```
| foreach mode=multivalue student_grades
      [eval total = total + '<<ITEM>>', count = count + 1]
```

```
| foreach mode=multivalue student_grades
      [eval total = total + '<<ITEM>>'
      | eval count = count + 1]
```


splunk> .conf23

mode=

multivalue

Iterate across the values within a single multivalue field.

- Token to access individual value being operated on: **<<ITEM>>**
- Elements within the subsearch eval expression(s) must be of the same type (strings or numbers)
- Allows multiple | *eval* assignments to be applied to each item within the same command execution scope

```
1  | union
2      [| makeresults count=3 | eval teacher="James"]
3      [| makeresults count=3 | eval teacher="Jessica"]
4      [| makeresults count=3 | eval teacher="Roberto"]
5  | eval student_grades = (random() % 100) + 1
6  | stats values(*) AS * BY teacher
7  | eval sum = 0,
8         iterations = 0
9  | foreach mode=multivalue student_grades
10     [| eval sum = sum + '<<ITEM>>',
11     iterations = iterations + 1]
12 | eval average = round(sum / iterations, 0)
13 | table teacher, student_grades, average, sum, iterations
```

teacher	student_grades	average	sum	iterations
James	12 48 77	46	137	3
Jessica	2 26 95	41	123	3
Roberto	20 54 84	53	158	3

New Search

```
| union
    [| makeresults | eval jsonfield=json_array(1, 2, 3)]
    [| makeresults | eval jsonfield=json_array(4, 5, 6)]
    [| makeresults | eval jsonfield=json_array(7, 8, 9)]
| eval total=0
| foreach mode=json_array jsonfield
    [eval total = total + '<<ITEM>>',
    json_mv = mvappend(json_mv, '<<ITEM>>')]
| table json_mv, jsonfield, total
```

json_mv	jsonfield	total
1 2 3	[1,2,3]	6
4 5 6	[4,5,6]	15
7 8 9	[7,8,9]	24

mode=

json_array

Iterate across the items within a single JSON array field.

- Token to access individual value being operated on: **<<ITEM>>**
- Elements within the subsearch eval expression(s) must be of the same type (strings or numbers)
- Allows multiple | *eval* assignments to be applied to each item within the same command execution scope

mode=(multivalue|json_array)

#docs - Common limitations

Wildcards are not supported in multivalue fields or JSON arrays

Unlike multifield mode, the modes for multivalue fields and JSON arrays don't support wildcards in search expressions. Instead, these modes treat a wildcard as part of the field name. For example, the following search includes a field called `mv*`, which looks like a wildcarded field.

```
| makeresults
| eval mv1=mvappend("1", "2"), mv2=mvappend("3", "4"), mv*=mvappend("100", "300"), total = 0
| foreach mode=multivalue mv*
    [eval total = total + <<ITEM>>]
```

However, multivalue and JSON array modes don't recognize the wildcard and add up all of the fields containing `mv`. As a result, the search results for multivalue mode look something like this:

_time	mv*	mv1	mv2	total
2022-4-20 15:55:50	100 300	1 2	3 4	400

Elements of the same type in multivalue fields or JSON arrays

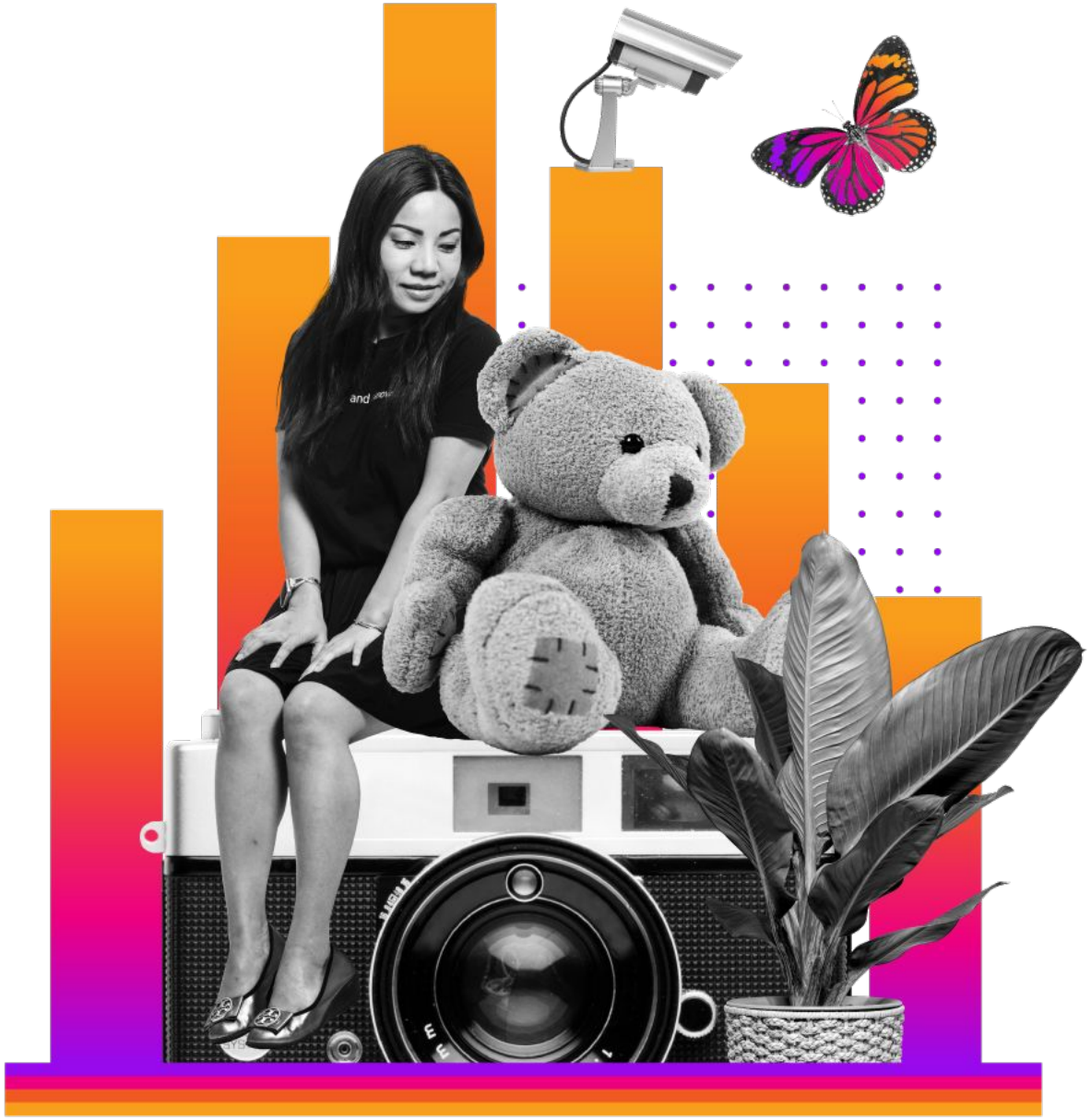
Elements in a subsearch and the `eval` expression must be of the same type as either strings or numbers. For example, the following search correctly adds up the three numbers in the JSON array because all of the elements are numbers.

```
| foreach json_array(1, 2, 3)
    [eval total = total + <<ITEM>>]
```

However, the following search results in an error because adding a number to a string isn't allowed.

```
| foreach json_array(1, 2, "hello")
    [eval total = total + <<ITEM>>]
```

https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Foreach



| foreach

mode=multifield

From here out everything we talk about is mode=multifield

splunk> .conf23

| foreach

mode=multifield

- Iterate across one or many fields, applying the defined subsearch template to each
  - Applied in Lexicographic order based on field name.
- Allows usage of wildcard to run across all fields available at time of command call.
  - Powerful, but slow if run over large input data
- Enables (mostly) unique capabilities:
  - Batch calculation, transformation, validation of fields
  - Compilation of fields into structured output
  - Dynamic thresholding and aggregate analysis

| foreach Behavior *mode=multifield*

```
| makeresults | fields - _time
| eval total = 0,
       test1=split("1,5", ","),
       test2=split("10,15", ","),
       test3=split("100,150", ",")
| mvexpand test1
| mvexpand test2
| mvexpand test3
| foreach test*
   [| eval total = total + <<FIELD>> ]
```

test1	test2	test3	total
1	10	100	111
1	10	150	161
1	15	100	116
1	15	150	166
5	10	100	115
5	10	150	165
5	15	100	120
5	15	150	170

| foreach Behavior *mode=multifield*

(do not embed foreach within foreach in any important SPL)

```
| makeresults | fields - _time
| eval total = 0, test1=split("1,5", ","), test2=split("10,15", ","), test3=split("100,150", ",")
| mvexpand test1 | mvexpand test2 | mvexpand test3 | eval counter = 0
| appendpipe
 [| foreach test*
   [| eval "Target Field" = "<<FIELD>>"
   | eval input_values = mvappend(input_values, "----------",
                    "ITERATION " . tostring(counter + 1) . " - INPUT", "----------")
   | foreach * fieldstr="#field#"
      [| eval input_values = if( isnotnull('#field#'),
                  mvappend(input_values, "#field# = " . '#field#'), input_values )
      ]
   | eval input_values=if(counter=0, mvappend(input_values,"total_str = "), input_values)
   | eval comment = if(true(), null(),
"Modify fields being iterated on mid-command execution. (Avoid doing this)")
   | eval test2 = if(counter = 0, test2 * 2, test2)
   | eval test3 = if(counter = 1, test3 * 5, test3)
   | eval total_str = total . " + " . '<<FIELD>>' . " (total + <<FIELD>>)"
   | eval total = total + <<FIELD>>
   | eval counter = counter + 1
   | eval output_values = mvappend(output_values, "----------",
                    "ITERATION " . counter . " - OUTPUT", "----------")
   | foreach * fieldstr="#field#"
      [| eval output_values = if( isnotnull('#field#') AND !match("#field#", "^input"),
                  mvappend(output_values, "#field# = " . '#field#'), output_values )
      ]
 ]
]
```

input_values	output_values
----------	----------
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT
----------	----------
Target Field = test1	Target Field = test1
counter = 0	counter = 1
test1 = 1	test1 = 1
test2 = 10	test2 = 20
test3 = 100	test3 = 100
total = 0	total = 1
total_str =	total_str = 0 + 1 (total + test1)
----------	----------
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT
----------	----------
Target Field = test2	Target Field = test2
counter = 1	counter = 2
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 100	test3 = 500
total = 1	total = 21
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)
----------	----------
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT
----------	----------
Target Field = test3	Target Field = test3
counter = 2	counter = 3
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 500	test3 = 500
total = 21	total = 521
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)

splunk> .conf23

input_values	output_values	test1	test2	test3	total	counter
	| head 1	1	10	100	0	0
----------	----------	1	20	500	521	3
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT					
----------	----------					
Target Field = test1	Target Field = test1					
counter = 0	counter = 1					
test1 = 1	test1 = 1					
test2 = 10	test2 = 20					
test3 = 100	test3 = 100					
total = 0	total = 1					
total_str =	total_str = 0 + 1 (total + test1)					
----------	----------					
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT					
----------	----------					
Target Field = test2	Target Field = test2					
counter = 1	counter = 2					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 100	test3 = 500					
total = 1	total = 21					
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)					
----------	----------					
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT					
----------	----------					
Target Field = test3	Target Field = test3					
counter = 2	counter = 3					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 500	test3 = 500					
total = 21	total = 521					
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)					

```
| makeresults | fields - _time
| eval total = 0,
      test1=split("1,5", ","),
      test2=split("10,15", ","),
      test3=split("100,150", ",")
| mvexpand test1
| mvexpand test2
| mvexpand test3
| head 1
| foreach test*
  [...]
```

input_values	output_values	test1	test2	test3	total	counter
		1	10	100	0	0
----------	----------	1	20	500	521	3
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT					
----------	----------					
Target Field = test1	Target Field = test1					
counter = 0	counter = 1					
test1 = 1	test1 = 1					
test2 = 10	test2 = 20					
test3 = 100	test3 = 100					
total = 0	total = 1					
total_str =	total_str = 0 + 1 (total + test1)					
----------	----------					
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT					
----------	----------					
Target Field = test2	Target Field = test2					
counter = 1	counter = 2					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 100	test3 = 500					
total = 1	total = 21					
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)					
----------	----------					
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT					
----------	----------					
Target Field = test3	Target Field = test3					
counter = 2	counter = 3					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 500	test3 = 500					
total = 21	total = 521					
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)					

```
| makeresults | fields - _time
| eval total = 0,
       test1=split("1,5", ","),
       test2=split("10,15", ","),
       test3=split("100,150", ",")
| mvexpand test1
| mvexpand test2
| mvexpand test3
| head 1
| foreach test*
    [...]
```

input_values	output_values	test1	test2	test3	total	counter
		1	10	100	0	0
----------	----------	1	20	500	521	3
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT					
----------	----------					
Target Field = test1	Target Field = test1					
counter = 0	counter = 1					
test1 = 1	test1 = 1					
test2 = 10	test2 = 20					
test3 = 100	test3 = 100					
total = 0	total = 1					
total_str =	total_str = 0 + 1 (total + test1)					
----------	----------					
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT					
----------	----------					
Target Field = test2	Target Field = test2					
counter = 1	counter = 2					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 100	test3 = 500					
total = 1	total = 21					
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)					
----------	----------					
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT					
----------	----------					
Target Field = test3	Target Field = test3					
counter = 2	counter = 3					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 500	test3 = 500					
total = 21	total = 521					
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)					

```
| makeresults | fields - _time
| eval total = 0,
       test1=split("1,5", ","),
       test2=split("10,15", ","),
       test3=split("100,150", ",")
| mvexpand test1
| mvexpand test2
| mvexpand test3
| head 1
| foreach test*
  [...]
```

?

| foreach Behavior *mode=multifield*

(do not embed foreach within foreach in any important SPL)

```
| makeresults | fields - _time
| eval total = 0, test1=split("1,5", ","), test2=split("10,15", ","), test3=split("100,150", ","), counter = 0
| mvexpand test1 | mvexpand test2 | mvexpand test3 | head 1
| appendpipe
 [| foreach test*
   [| eval "Target Field" = "<<FIELD>>"
   | eval input_values = mvappend(input_values, "----------",
                    "ITERATION " . tostring(counter + 1) . " - INPUT", "----------")
   | foreach * fieldstr="#field#"
     [| eval input_values = if( isnotnull('#field#'),
                 mvappend(input_values, "#field# = " . '#field#'), input_values )
     ]
   | eval input_values=if(counter=0, mvappend(input_values,"total_str = "), input_values)
   | eval comment = if(true(), null(),
"Modify fields being iterated on mid-command execution. (Avoid doing this)")
   | eval test2 = if(counter = 0, test2 * 2, test2)
   | eval test3 = if(counter = 1, test3 * 5, test3)
   | eval total_str = total . " + " . '<<FIELD>>' . " (total + <<FIELD>>)"
   | eval total = total + <<FIELD>>
   | eval counter = counter + 1
   | eval output_values = mvappend(output_values, "----------",
                    "ITERATION " . counter . " - OUTPUT", "----------")
   | foreach * fieldstr="#field#"
     [| eval output_values = if( isnotnull('#field#') AND !match("#field#", "^input"),
                 mvappend(output_values, "#field# = " . '#field#'), output_values )
     ]
 ]
]
```

splunk> .conf23

input_values	output_values
----------	----------
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT
----------	----------
Target Field = test1	Target Field = test1
counter = 0	counter = 1
test1 = 1	test1 = 1
test2 = 10	test2 = 20
test3 = 100	test3 = 100
total = 0	total = 1
total_str =	total_str = 0 + 1 (total + test1)
----------	----------
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT
----------	----------
Target Field = test2	Target Field = test2
counter = 1	counter = 2
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 100	test3 = 500
total = 1	total = 21
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)
----------	----------
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT
----------	----------
Target Field = test3	Target Field = test3
counter = 2	counter = 3
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 500	test3 = 500
total = 21	total = 521
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)

| foreach Behavior *mode=multifield*

(do not embed foreach within foreach in any important SPL)

```
| makeresults | fields - _time
| eval total = 0, test1=split("1,5", ","), test2=split("10,15", ","), test3=split("100,150", ","), counter = 0
| mvexpand test1 | mvexpand test2 | mvexpand test3 | head 1
| appendpipe
 [| foreach test*
   [| eval "Target Field" = "<<FIELD>>"
   | eval input_values = mvappend(input_values, "----------",
                    "ITERATION " . tostring(counter + 1) . " - INPUT", "----------")
   | foreach * fieldstr="#field#"
      [| eval input_values = if( isnotnull('#field#'),
                  mvappend(input_values, "#field# = " . '#field#'), input_values )
      ]
   | eval input_values=if(counter=0, mvappend(input_values,"total_str = "), input_values)
   | eval comment = if(true(), null(),
"Modify fields being iterated on mid-command execution. (Avoid doing this)")
   | eval test2 = if(counter = 0, test2 * 2, test2)
   | eval test3 = if(counter = 1, test3 * 5, test3)
   | eval total_str = total . " + " . '<<FIELD>>' . " (total + <<FIELD>>)"
   | eval total = total + <<FIELD>>
   | eval counter = counter + 1
   | eval output_values = mvappend(output_values, "----------",
                    "ITERATION " . counter . " - OUTPUT", "----------")
   | foreach * fieldstr="#field#"
      [| eval output_values = if( isnotnull('#field#') AND !match("#field#", "^input"),
                  mvappend(output_values, "#field# = " . '#field#'), output_values )
      ]
 ]
]
```

input_values	output_values
----------	----------
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT
----------	----------
Target Field = test1	Target Field = test1
counter = 0	counter = 1
test1 = 1	test1 = 1
test2 = 10	test2 = 20
test3 = 100	test3 = 100
total = 0	total = 1
total_str =	total_str = 0 + 1 (total + test1)
----------	----------
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT
----------	----------
Target Field = test2	Target Field = test2
counter = 1	counter = 2
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 100	test3 = 500
total = 1	total = 21
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)
----------	----------
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT
----------	----------
Target Field = test3	Target Field = test3
counter = 2	counter = 3
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 500	test3 = 500
total = 21	total = 521
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)

splunk> .conf23

| foreach Behavior *mode=multifield*

(do not embed foreach within foreach in any important SPL)


```
| makeresults | fields - _time
| eval total = 0, test1=split("1,5", ","), test2=split("10,15", ","), test3=split("100,150", ","), counter = 0
| mvexpand test1 | mvexpand test2 | mvexpand test3 | head 1
| appendpipe
 [| foreach test*
   [| eval "Target Field" = "<<FIELD>>"
   | eval input_values = mvappend(input_values, "----------",
                    "ITERATION " . tostring(counter + 1) . " - INPUT", "----------")
   | foreach * fieldstr="#field#"
     [| eval input_values = if( isnotnull('#field#'),
                 mvappend(input_values, "#field# = " . '#field#'), input_values )
     ]
   | eval input_values=if(counter=0, mvappend(input_values,"total_str = "), input_values)
   | eval comment = if(true(), null(),
"Modify fields being iterated on mid-command execution. (Avoid doing this)")
   | eval test2 = if(counter = 0, test2 * 2, test2)
   | eval test3 = if(counter = 1, test3 * 5, test3)
   | eval total_str = total . " + " . '<<FIELD>>' . " (total + <<FIELD>>)"
   | eval total = total + <<FIELD>>
   | eval counter = counter + 1
   | eval output_values = mvappend(output_values, "----------",
                    "ITERATION " . counter . " - OUTPUT", "----------")
   | foreach * fieldstr="#field#"
     [| eval output_values = if( isnotnull('#field#') AND !match("#field#", "^input"),
                 mvappend(output_values, "#field# = " . '#field#'), output_values )
     ]
 ]
]
```

input_values	output_values
----------	----------
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT
----------	----------
Target Field = test1	Target Field = test1
counter = 0	counter = 1
test1 = 1	test1 = 1
test2 = 10	test2 = 20
test3 = 100	test3 = 100
total = 0	total = 1
total_str =	total_str = 0 + 1 (total + test1)
----------	----------
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT
----------	----------
Target Field = test2	Target Field = test2
counter = 1	counter = 2
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 100	test3 = 500
total = 1	total = 21
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)
----------	----------
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT
----------	----------
Target Field = test3	Target Field = test3
counter = 2	counter = 3
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 500	test3 = 500
total = 21	total = 521
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)

splunk> .conf23

| foreach Behavior *mode=multifield*

(do not embed foreach within foreach in any important SPL)


```
| makeresults | fields - _time
| eval total = 0, test1=split("1,5", ","), test2=split("10,15", ","), test3=split("100,150", ","), counter = 0
| mvexpand test1 | mvexpand test2 | mvexpand test3 | head 1
| appendpipe
 [| foreach test*
   [| eval "Target Field" = "<<FIELD>>"
   | eval input_values = mvappend(input_values, "----------",
                    "ITERATION " . tostring(counter + 1) . " - INPUT", "----------")
   | foreach * fieldstr="#field#"
     [| eval input_values = if( isnotnull('#field#'),
                 mvappend(input_values, "#field# = " . '#field#'), input_values )
     ]
   | eval input_values=if(counter=0, mvappend(input_values,"total_str = "), input_values)
   | eval comment = if(true(), null(),
"Modify fields being iterated on mid-command execution. (Avoid doing this)")
   | eval test2 = if(counter = 0, test2 * 2, test2)
   | eval test3 = if(counter = 1, test3 * 5, test3)
   | eval total_str = total . " + " . '<<FIELD>>' . " (total + <<FIELD>>)"
   | eval total = total + '<<FIELD>>'
   | eval counter = counter + 1
   | eval output_values = mvappend(output_values, "----------",
                    "ITERATION " . counter . " - OUTPUT", "----------")
   | foreach * fieldstr="#field#"
     [| eval output_values = if( isnotnull('#field#') AND !match("#field#", "^input"),
                 mvappend(output_values, "#field# = " . '#field#'), output_values )
     ]
 ]
]
```

input_values	output_values
----------	----------
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT
----------	----------
Target Field = test1	Target Field = test1
counter = 0	counter = 1
test1 = 1	test1 = 1
test2 = 10	test2 = 20
test3 = 100	test3 = 100
total = 0	total = 1
total_str =	total_str = 0 + 1 (total + test1)
----------	----------
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT
----------	----------
Target Field = test2	Target Field = test2
counter = 1	counter = 2
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 100	test3 = 500
total = 1	total = 21
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)
----------	----------
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT
----------	----------
Target Field = test3	Target Field = test3
counter = 2	counter = 3
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 500	test3 = 500
total = 21	total = 521
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)

splunk> .conf23

| foreach Behavior *mode=multifield*

(do not embed foreach within foreach in any important SPL)

```
[| foreach test*
```

~CRITICAL TIP~

Avoid modifying <u>other</u> target fields mid-iteration!

but it's okay to modify the field *<u>being iterated on</u>*

```
   | eval comment = if(true(), null(),
"Modify fields being iterated on mid-command execution. (Avoid doing this)")
   | eval test2 = if(counter = 0, test2 * 2, test2)
   | eval test3 = if(counter = 1, test3 * 5, test3)
   | eval total_str = total . " + " . '<<FIELD>>' . " (total + <<FIELD>>)"
   | eval total = total + '<<FIELD>>'
   | eval counter = counter + 1
   | eval output_values = mvappend(output_values, "----------",
                    "ITERATION " . counter . " - OUTPUT", "----------")
   | foreach * fieldstr="#field#"
     [| eval output_values = if( isnotnull('#field#') AND !match("#field#", "^input"),
                 mvappend(output_values, "#field# = " . '#field#'), output_values )
     ]
 ]
]
```

splunk> .conf23

input_values	output_values
----------	----------
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT
----------	----------
Target Field = test1	Target Field = test1
counter = 0	counter = 1
test1 = 1	test1 = 1
test2 = 10 →	test2 = 20
test3 = 100	test3 = 100
total = 0	total = 1
total_str =	total_str = 0 + 1 (total + test1)
----------	----------
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT
----------	----------
Target Field = test2	Target Field = test2
counter = 1	counter = 2
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 100 →	test3 = 500
total = 1	total = 21
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)
----------	----------
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT
----------	----------
Target Field = test3	Target Field = test3
counter = 2	counter = 3
test1 = 1	test1 = 1
test2 = 20	test2 = 20
test3 = 500	test3 = 500
total = 21	total = 521
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)

input_values	output_values	test1	test2	test3	total	counter
	/ *appendpipe [...]*	1	10	100	0	0
----------	----------	1	20	500	521	3
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT					
----------	----------					
Target Field = test1	Target Field = test1					
counter = 0	counter = 1					
test1 = 1	test1 = 1					
test2 = 10	test2 = 20					
test3 = 100	test3 = 100					
total = 0	total = 1					
total_str =	total_str = 0 + 1 (total + test1)					
----------	----------					
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT					
----------	----------					
Target Field = test2	Target Field = test2					
counter = 1	counter = 2					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 100	test3 = 500					
total = 1	total = 21					
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)					
----------	----------					
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT					
----------	----------					
Target Field = test3	Target Field = test3					
counter = 2	counter = 3					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 500	test3 = 500					
total = 21	total = 521					
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)					

How can I show what's happening?

input_values	output_values	test1	test2	test3	total	counter
		1	10	100	0	0
------------------------------	------------------------------	1	20	500	521	3
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT					
------------------------------	------------------------------					
Target Field = test1	Target Field = test1					
counter = 0	counter = 1					
test1 = 1	test1 = 1					
test2 = 10	test2 = 20					
test3 = 100	test3 = 100					
total = 0	total = 1					
total_str =	total_str = 0 + 1 (total + test1)					
------------------------------	------------------------------					
ITERATION 2 - INPUT	ITERATION 2 - OUTPUT					
------------------------------	------------------------------					
Target Field = test2	Target Field = test2					
counter = 1	counter = 2					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 100	test3 = 500					
total = 1	total = 21					
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)					
------------------------------	------------------------------					
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT					
------------------------------	------------------------------					
Target Field = test3	Target Field = test3					
counter = 2	counter = 3					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 500	test3 = 500					
total = 21	total = 521					
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)					

How can I show what's happening?

Compilation using foreach!

Behavior Established

Level 0 achieved.

Time to Level Up!

splunk> .conf23

Usecases

Level 1

1

Convert | eval's to a Template Approach

Clean up your SPL by merging repetitive eval logic.

Level 1 - Overview of Implementation

| foreach - Convert eval's to a Template Approach

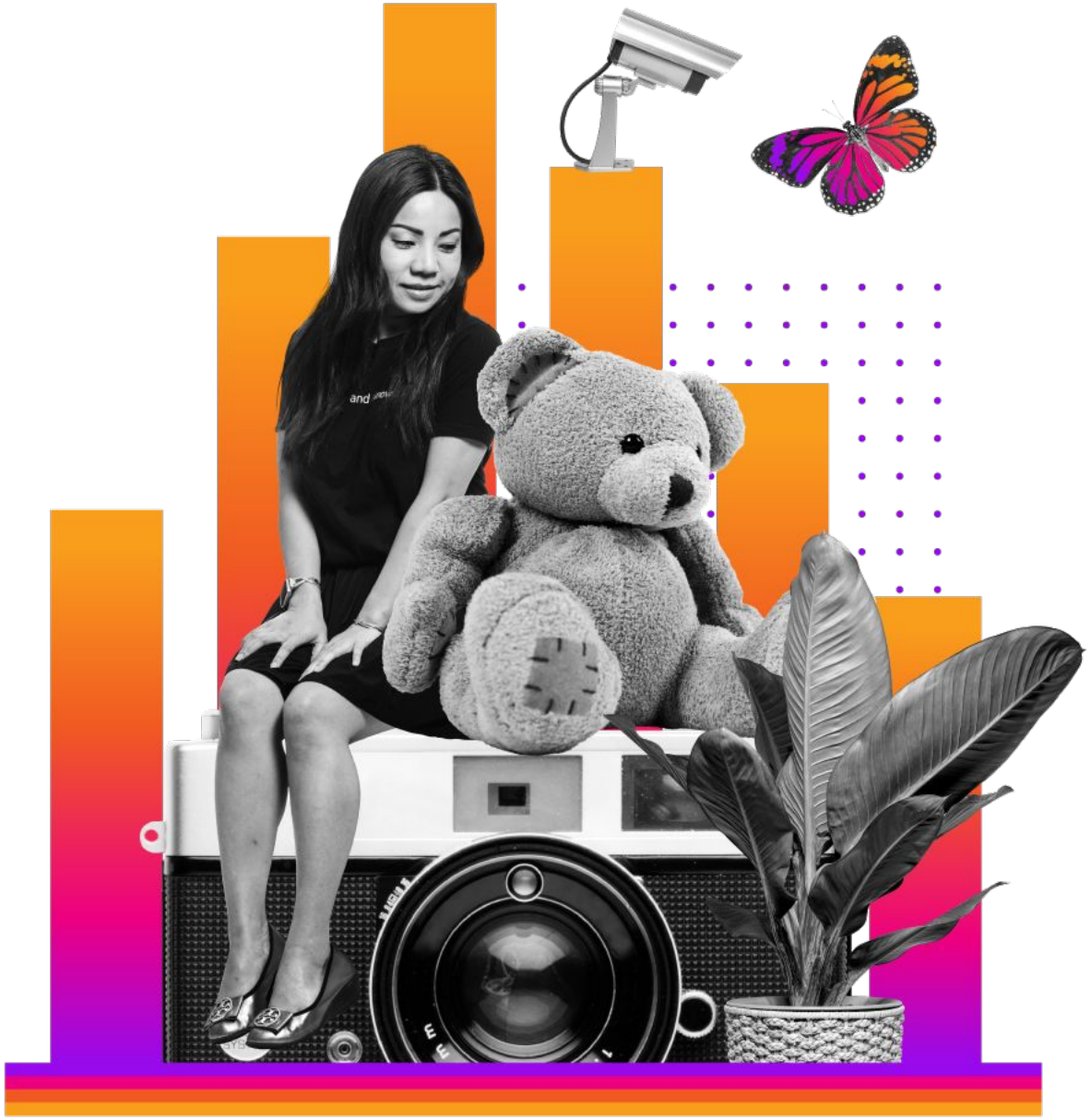
1. Look at your searches to identify eval patterns that are repeated across multiple fields.

2. Merge those evals into a foreach statement, specifying the original fields as the target.

Utility/Benefits

Level 1
Convert | eval's to a Template Approach

- Reduced length of SPL™ queries while maintaining readability
- Before/After are equivalent in terms of processing and resource cost
- Ease of expanding common logic to additional fields if needed



Level 1 Examples

splunk> .conf23

| eval Consolidation

Level 1 - Simplify that SPL™!

Long, Repetitive SPL

```
| eval host = lower(host)
| eval hostname = lower(hostname)
| eval user = lower(user)

| eval kbps = round(kbps, 0)
| eval eps = round(eps, 0)

| eval datetime = strptime(datetime, "%Y-%m-%dT%H:%M:%S")
| eval start = strptime(start, "%Y-%m-%dT%H:%M:%S")
| eval end = strptime(end, "%Y-%m-%dT%H:%M:%S")

| eval message = substr(message, 1, 151)
| eval event_message = substr(event_message, 1, 151)
| eval description = substr(description, 1, 151)
```

Concise, Templatized

```
| foreach host, hostname, user
    [| eval <<FIELD>> = lower('<<FIELD>>')]
```

```
| foreach kbps, eps
    [| eval <<FIELD>> = round('<<FIELD>>', 0)]
```

```
| foreach datetime, start, end
    [| eval <<FIELD>> = strptime('<<FIELD>>', "%Y-%m-%dT%H:%M:%S")]
```

```
| foreach message, event_message, description
    [| eval <<FIELD>> = substr('<<FIELD>>', 1, 151)]
```

| eval Consolidation

Level 1 - Simplify that SPL™!

Long, Repetitive SPL

```
| eval host = lower(host)
| eval hostname = lower(hostname)
| eval user = lower(user)

| eval kbps = round(kbps, 0)
| eval eps = round(eps, 0)

| eval datetime = strptime(datetime, "%Y-%m-%dT%H:%M:%S")
| eval start = strptime(start, "%Y-%m-%dT%H:%M:%S")
| eval end = strptime(end, "%Y-%m-%dT%H:%M:%S")

| eval message = substr(message, 1, 151)
| eval event_message = substr(event_message, 1, 151)
| eval description = substr(description, 1, 151)
```

Concise, Templatized

```
| foreach host, hostname, user
    [| eval <<FIELD>> = lower('<<FIELD>>')]

| foreach kbps, eps
    [| eval <<FIELD>> = round('<<FIELD>>', 0)]

| foreach datetime, start, end
    [| eval <<FIELD>> = strptime('<<FIELD>>', "%Y-%m-%dT%H:%M:%S")]

| foreach message, event_message, description
    [| eval <<FIELD>> = substr('<<FIELD>>', 1, 151)]
```

?

splunk> .conf23

~CRITICAL TIP~

Wrap your <<FIELD>> in single quotes!

(on the right side of the eval '=')

- Better to get in the habit of **always** wrapping your '<<FIELD>>' and '<<ITEM>>' references

- Helps differentiate between referencing the **field *name*** ("<<FIELD>>") vs **field *value*** ('<<FIELD>>')

```
| foreach host, hostname, user
    [| eval <<FIELD>> = lower('<<FIELD>>')]

| foreach kbps, eps
    [| eval <<FIELD>> = round('<<FIELD>>', 0)]

| foreach datetime, start, end
    [| eval <<FIELD>> = strptime('<<FIELD>>', "%Y-%m-%dT%H:%M:%S")]

| foreach message, event_message, description
    [| eval <<FIELD>> = substr('<<FIELD>>', 1, 151)]
```

Field References Within | eval

'Left Side'

"This is part of the field name"

=

'Right Side'

"This is referencing a field"

New Search

```
1  | makeresults | fields - _time
2  | eval 'user.fullname' = "ryan wood"
3  | eval user = "admin123"
4  | eval user.fullname = "Ryan Wood"
5  | eval "user.fullname" = "Wood Ryan"
6  | eval user_concat_1 = user . user.fullname
7  | eval user_concat_2 = user . 'user.fullname'
```

field	value
'user.fullname'	ryan wood
user	admin123
user.fullname	Wood Ryan
user_concat_2	admin123Wood Ryan

splunk> .conf23

Field References Within | eval

'Left Side'

"This is part of the field name"

=

'Right Side'

"This is referencing a field"

New Search

```
1  | makeresults | fields - _time
2  | eval 'user.fullname' = "ryan wood"
3  | eval user = "admin123"
4  | eval user.fullname = "Ryan Wood"
5  | eval "user.fullname" = "Wood Ryan"
6  | eval user_concat_1 = user . user.fullname
7  | eval user_concat_2 = user . 'user.fullname'
```

field	value
'user.fullname'	ryan wood
user	admin123
user.fullname	Wood Ryan
user_concat_2	admin123Wood Ryan

Field References Within | eval

'Left Side'

"This is part of the field name"

'Right Side'

"This is referencing a field"

New Search

```
1  | makeresults | fields - _time
2  | eval 'user.fullname' = "ryan wood"
3  | eval user = "admin123"
4  | eval user.fullname = "Ryan Wood"
5  | eval "user.fullname" = "Wood Ryan"
6  | eval user_concat_1 = user . user.fullname
7  | eval user_concat_2 = user . 'user.fullname'
```

Overwrites

field	value
'user.fullname'	ryan wood
user	admin123
user.fullname	Wood Ryan
user_concat_2	admin123Wood Ryan

| foreach - Starting Point

Conversion without modification is the simplest usage of foreach.
Multiple evals can be included in the consolidation.

Long, Repetitive SPL™

```
| eval comment = if( true(), null(), "
Preserve the readable timestamps in equivalent *_strf fields,
then convert original to epoch.")
| rename *_time AS *_time_strf
| eval close_time = strptime(close_time_strf, "%Y-%m-%dT%H:%M:%S.%6N"),
   create_time = strptime(create_time_strf, "%Y-%m-%dT%H:%M:%S.%6N"),
   due_time = strptime(due_time_strf, "%Y-%m-%dT%H:%M:%S.%6N"),
   end_time = strptime(end_time_strf, "%Y-%m-%dT%H:%M:%S.%6N"),
   start_time = strptime(start_time_strf, "%Y-%m-%dT%H:%M:%S.%6N"),
   update_time=strptime(update_time_strf, "%Y-%m-%dT%H:%M:%S.%6N")
```

Concise, Templatized

```
| eval comment = if( true(), null(), "
Preserve the readable timestamps in equivalent *_strf fields,
then convert original to epoch.")
| foreach *_time
   [| eval <<FIELD>>_strf = '<<FIELD>>'
   | eval <<FIELD>> = strptime('<<FIELD>>', "%Y-%m-%dT%H:%M:%S.%6N")]
```

Usecases

Level 2

2

Field Validation & Batch Transformations

Normalize your fields to ensure your logic works like it should.

Wildcard Refs

```
| foreach "*"
```

- Wildcard references allow us to build logic that be applied to all of the available fields - enabling consistent, predictable behavior.

- Can have significant impact when run on large input dataset - reduce input dataset size where possible:
  - specify specific fields instead of "*" wildcard, OR
  - reduce fields before the foreach command runs via transforming command or `| fields` command

Wildcard foreach Usage - Overview of Implementation

| foreach

1) Identify common transformations that need to be applied across a number of fields.
   a) Tip - Use Level 3 Methodologies
2) Generate/compile sample data representing “Before” state to build and test logic on.
   a) Tip - `| loadjob` is your best friend!
   b) Tip - Summary indexing via `| collect`
3) Build non-foreach “verbose” eval SPL to serve as a baseline - this will help identify if foreach approach is significantly slower.
   a) Reference Utility '*Generate SPL using SPL'* for an easy way to compile this verbose approach
4) Build foreach SPL with the identified transformations from Step 1 implemented.
5) Run both “verbose” and “foreach” SPL queries on same input data - **Validate results are consistent between the two approaches.**
   a) Reference Utility ‘*Check two sets of search results are the same*’ for a way to do this per-row.
6) Run both “verbose” and “foreach” SPL queries on same input data to confirm impact from foreach usage. **AFTER confirming results are consistent between the two queries!**
   a) `Job performance properties` can help understand performance cost of foreach vs eval:

```
| rest /servicesNS/-/-/search/jobs/<sid>
| fields sid, title, published, runDuration, performance.command.eval*,
performance.command.foreach*
```

Utility/Benefits

Level 2
Field Validation & Batch Transformations

- Build Confidence
  - Ensure that fields are the format/data type expected.
- Transform fields at scale
- Batch regex replacements
- Multivalue field operations

Scenario

Two events, same fields and values.

```
{
  "first_name":"Ryan",
  "last_name":"Wood",
  "company":"GuidePoint Security",
  "age": "",
  "hobbies": ["SPL", "Biking"]
}
```

```
first_name="Ryan", last_name="Wood",
company="GuidePoint Security", age="",
hobbies="SPL", hobbies="Biking"
```

Scenario

```
isnull(age) OR isnotnull(age)?
```

```
{
  "first_name":"Ryan",
  "last_name":"Wood",
  "company":"GuidePoint Security",
  "age": "",
  "hobbies": ["SPL", "Biking"]
}
```

```
first_name="Ryan", last_name="Wood",
company="GuidePoint Security", age="",
hobbies="SPL", hobbies="Biking"
```

* Key Point *

Highlight: Handling Zero-Length Fields

NULL(ish) in Splunk

```
...
| where isnotnull(age)
```

_raw	age	company	first_name	hobbies	last_name
{ "time":"1979-09-16T00:00:00", "first_name":"Ryan", "last_name":"Wood", "company":"GuidePoint Security", "age": "", "hobbies": ["SPL", "Biking"] }	?	GuidePoint Security	Ryan	SPL Biking	Wood
time="1979-09-16T00:00:00", first_name="Ryan", last_name="Wood", company="GuidePoint Security", age="", hobbies="SPL", hobbies="Biking"	?	GuidePoint Security	Ryan	SPL Biking	Wood

Solution!

Templatized handling of "" zero-length field values.

| foreach *field*, set true null()


```
...
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' ) ]
```

For each field:

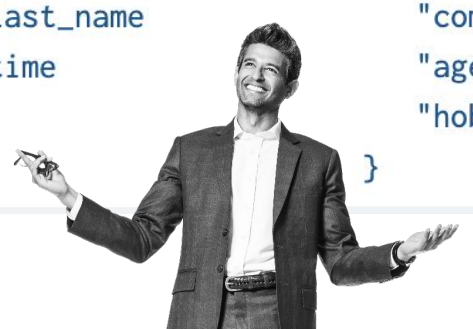
- 1) Check if the 'value' length is less than 1
- 2) Set value to `null()` if true.
- 3) Preserve field value in all other cases.

| foreach *field*, set true null()


```
...
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' ) ]
```

No more zero-length headaches!

_raw	null_fields	notnull_fields
{ "time":"1979-09-16T00:00:00", "first_name":"Ryan", "last_name":"Wood", "company":"GuidePoint Security", "age": "", "hobbies": ["SPL", "Biking"] }		age company first_name hobbies last_name time



_raw	null_fields	notnull_fields
{ "time":"1979-09-16T00:00:00", "first_name":"Ryan", "last_name":"Wood", "company":"GuidePoint Security", "age": "", "hobbies": ["SPL", "Biking"] }	age	company first_name hobbies last_name time

| foreach *field*, set true null()


```
...
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' ) ]
| where isnotnull(age)
```

No results found. Try expanding the time range.

| foreach *field*, ~~set true null()~~ set "string"

~CRITICAL TIP~

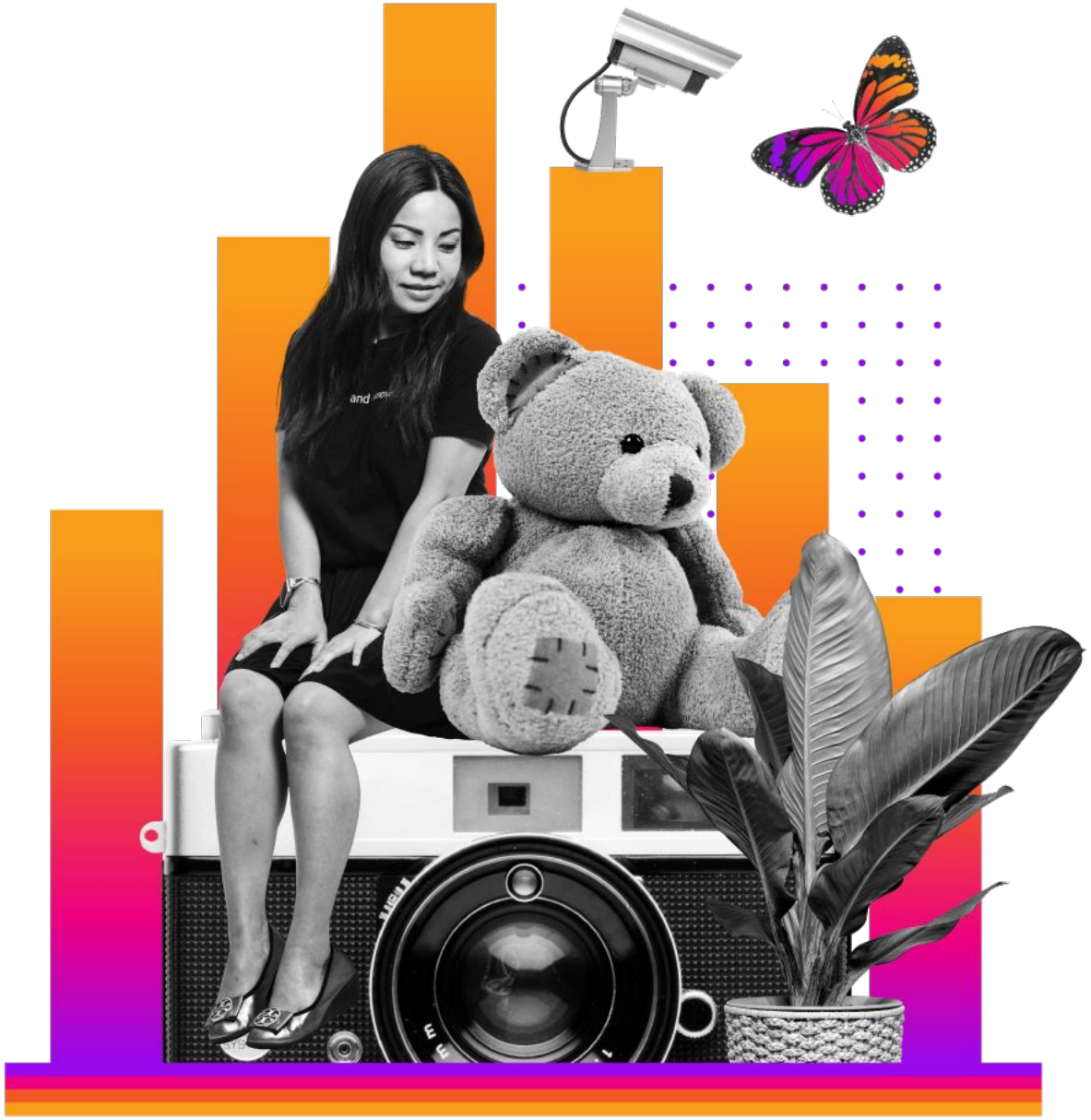
length() of null() is still null()!

To insert a "string" instead of setting null(), use the **top** pattern instead:

```
| foreach *
   [| eval <<FIELD>> = if( len('<<FIELD>>') < 1 OR isnull('<<FIELD>>'), "unknown", '<<FIELD>>' ) ]
```

```
| foreach *
   [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, "unknown", '<<FIELD>>' ) ]
```

^ *This bottom pattern **will not insert a "value" into null fields.***



2

Field Validation & Batch Transformations

Normalize your fields to ensure your logic works like it should.

Level 2 Examples

What else can we do?

splunk> .conf23

Examples
Level 2

1) Field Transformations
   - 1.1) Handle zero-length fields
   - 1.2) Prefix field values with field name

More references in addendum

1) Field Transformations
(non-regex)

1.1) Handle zero-length fields

```
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' ) ]
```

1.2) Prefix field *values* with field *name* – `field="value"`

```
| foreach *
    [| eval <<FIELD>> = "<<FIELD>>=\"" . '<<FIELD>>' . "\""]
```

hostname	src_ip
SPL-SH01	10.10.0.10
SPL-IDX01	10.10.0.11
SPL-IDX02	10.10.0.12

hostname	src_ip
hostname="SPL-SH01"	src_ip="10.10.0.10"
hostname="SPL-IDX01"	src_ip="10.10.0.11"
hostname="SPL-IDX02"	src_ip="10.10.0.12"

Examples
Level 2

1) Field Transformations
- 1.1) Handle zero-length fields
- 1.2) Prefix field values with field name

2) Batch Regex Replacements
- 2.1) Remove all non-ascii characters
- 2.2) Add escape backslash to special characters

More references in addendum

2) Batch Regex Replacements

2.1) Remove all non-ascii characters

```
| foreach *
    [| eval <<FIELD>> = if( isnotnull('<<FIELD>>'), replace('<<FIELD>>', "[^[:ascii:]]+", ""),  null() )]
```

2.2) Add escape backslash to special characters *(if not already escaped)*
- using `| rex mode=sed` OR `| eval replace`

```
| foreach *
    [| rex mode=sed field="<<FIELD>>" "s/([^\\\](?!\\\))(\W)/\1\\\\\2/g"
  OR | eval <<FIELD>> = replace('<<FIELD>>', "([^\\\](?!\\\))(\W)", "\1\\\\\2")
    ]
```

splunk_major_breakers	string	string2
\<>[](){\\}!?;,"&	Escapes are important.	But\ careful\ doubling\ up!
\<\>\[\]\(\)\{\\}\!\?\;\,\"\&	Escapes\ are\ important\.	But\ careful\ doubling\ up\!\

splunk> .conf23

Examples
Level 2

1) Field Transformations
   - 1.1) Handle zero-length fields
   - 1.2) Prefix field values with field name

2) Batch Regex Replacements
   - 2.1) Remove all non-ascii characters
   - 2.2) Add escape backslash to special characters

3) Multivalue Field Handling
   - 3.1) Concatenate multivalued items together, Split single-value string into multivalue list
   - 3.2) Limit to first 5 values in multivalue list (addendum)
   - 3.3) Prep Results for CSV Export/Summary Index (addendum)

More references in addendum

3) Multivalue Field Handling

3.1) Concatenate multivalued items together / Split single-value string into a multivalue list

```
| foreach *
    [| eval <<FIELD>> = if( mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', "::delim::"), '<<FIELD>>' )]

| foreach *
    [| eval <<FIELD>> = if( match('<<FIELD>>', "::delim::"), split('<<FIELD>>', "::delim::"), '<<FIELD>>')]
```

hostname	src_ip
SPL-IDX01	10.10.0.10
SPL-IDX02	10.10.0.11
SPL-SH01	10.10.0.12

hostname	src_ip
SPL-IDX01::delim::SPL-IDX02::delim::SPL-SH01	10.10.0.10::delim::10.10.0.11::delim::10.10.0.12

hostname	src_ip
SPL-IDX01	10.10.0.10
SPL-IDX02	10.10.0.11
SPL-SH01	10.10.0.12

splunk> .conf23

#Content - Technical Topics Reminder

Setting the stage

Complexity

1
Convert | eval's to a Template Approach
Clean up your SPL by merging repetitive eval logic.

2
Field Validation & Batch Transformations
Normalize your fields to ensure your logic works like it should.

3
Present Your Data through Aggregation
Compile field values for review or to output in a specific format

4
Dynamic References & Adaptive Design
Versatile Logic for Dynamic Thresholding & Advanced Insights

Base Layer
The simplest form of using this command.

Lay the Foundations
Ensure your assumptions are accurate.

Review & Communicate
Analyze what you have available.
Make it understandable.
Identify opportunities.

Usecases

Level 3

3

Present Your Data through Aggregation

Compile field values for review or to output in a specific format

Utility/Benefits

Level 3
Present Your Data through Aggregation

- Aggregate & present large amounts of information into readable, concise columns
  - Enable Investigation
  - Clean up your dashboard panels
  - Add enrichment to your alert output
- Identify fields that match `<condition>`
  - Validation at scale
- Output data in desired formats

* Key Point *

Highlight - Compiling Fields for Review

…to save time & build better understanding.

input_values	output_values	test1	test2	test3	total	counter
		1	10	100	0	0
----------	----------	1	20	500	521	3
ITERATION 1 - INPUT	ITERATION 1 - OUTPUT					
----------	----------					
Target Field = test1	Target Field = test1					
counter = 0	counter =					
test1 = 1	test1 = 1					
test2 = 10	test2 = 2					
test3 = 100	test3 = 1					
total = 0	total = 1					
total_str =	total_str					
----------	----------					
ITERATION 2 - INPUT	ITERATION					
----------	----------					
Target Field = test2	Target Field = test2					
counter = 1	counter = 2					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 100	test3 = 500					
total = 1	total = 21					
total_str = 0 + 1 (total + test1)	total_str = 1 + 20 (total + test2)					
----------	----------					
ITERATION 3 - INPUT	ITERATION 3 - OUTPUT					
----------	----------					
Target Field = test3	Target Field = test3					
counter = 2	counter = 3					
test1 = 1	test1 = 1					
test2 = 20	test2 = 20					
test3 = 500	test3 = 500					
total = 21	total = 521					
total_str = 1 + 20 (total + test2)	total_str = 21 + 500 (total + test3)					

```
...
| foreach * fieldstr="#field#"
    [| eval input_values =
        if( isnotnull('#field#'),
            mvappend(input_values, "#field# = " . '#field#'),
            input_values )
    ]
```

splunk> .conf23

Compilation of Fields

2

```
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' ) ]
```

3

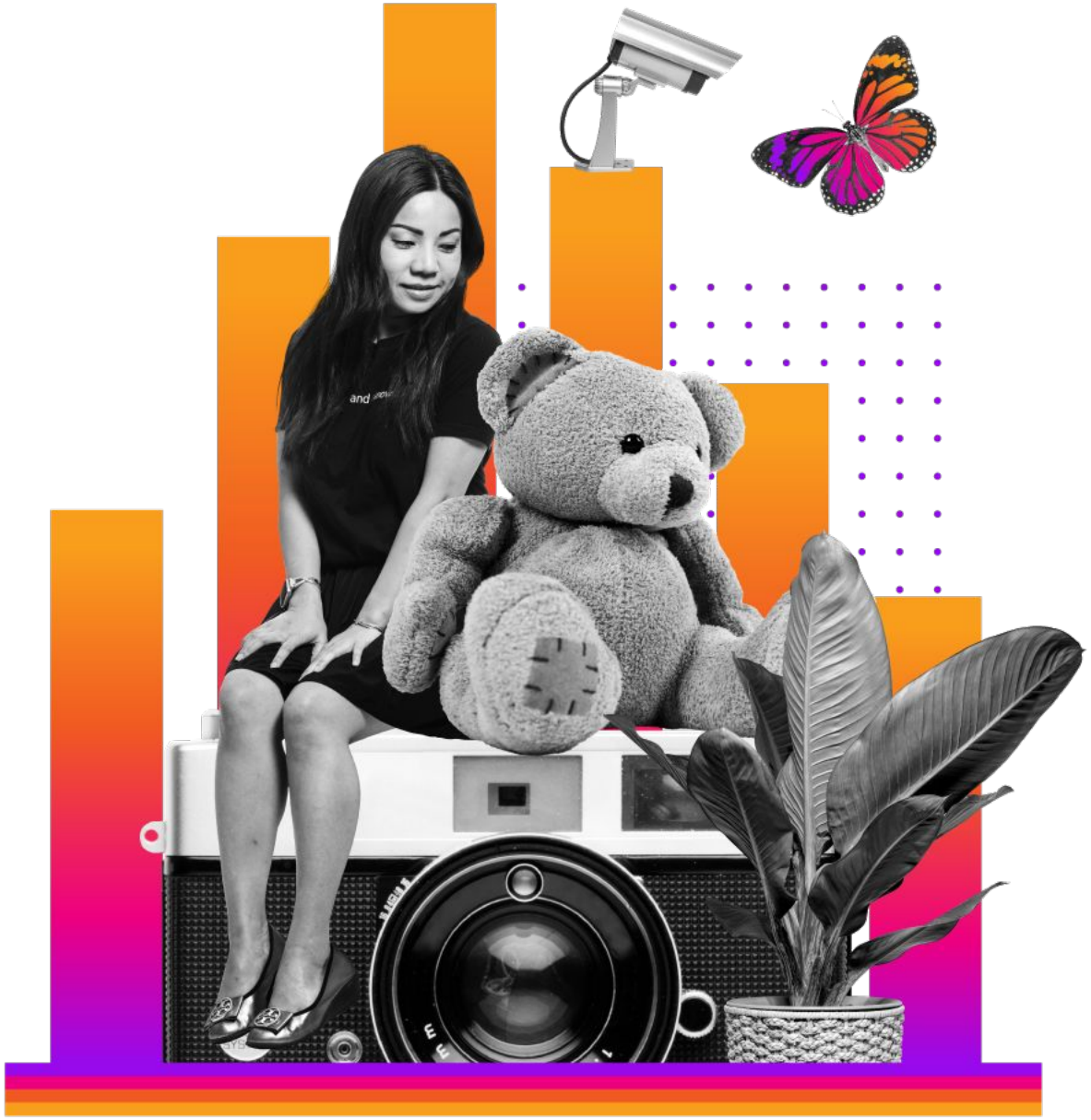
```
| foreach *
  [| eval null_fields =
            if( isnull('<<FIELD>>'), mvappend(null_fields, "<<FIELD>>"), null_fields)
  | eval notnull_fields =
            if( isnotnull('<<FIELD>>'), mvappend(notnull_fields, "<<FIELD>>"), notnull_fields)]
```

2

3

_raw	null_fields	notnull_fields
{ "time":"1979-09-16T00:00:00", "first_name":"Ryan", "last_name":"Wood", "company":"GuidePoint Security", "age": "", "hobbies": ["SPL", "Biking"] }		age company first_name hobbies last_name time

_raw	null_fields	notnull_fields
{ "time":"1979-09-16T00:00:00", "first_name":"Ryan", "last_name":"Wood", "company":"GuidePoint Security", "age": "", "hobbies": ["SPL", "Biking"] }	age	company first_name hobbies last_name time



3

Present Your Data through Aggregation

Compile field values for review or to output in a specific format

Level 3 Examples

Compilation in action.

splunk> .conf23

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

More references in addendum

1) Which fields…

1.1) Which fields **are Not Null?**

```
| foreach *
  [| eval notnull_fields =
          if( isnotnull('<<FIELD>>'), mvappend(notnull_fields, "<<FIELD>>"), notnull_fields)]
```

1.2) Which fields **are over a certain length?**

```
| foreach *
   [| eval long_fields = if( len('<<FIELD>>') > 499, mvappend(long_fields, "<<FIELD>>"), long_fields)]
```

1.3) Which fields **are the longest?**

- Overwrite field value with field length, then run calculations BY sourcetype

```
| foreach *
   [| eval <<FIELD>> =
        if( IN("<<FIELD>>", "sourcetype"), '<<FIELD>>', len(mvjoin('<<FIELD>>', "")) )
   ]
| stats max(*) AS max_len-* BY sourcetype
```

sourcetype	max_len-file	max_len-method	max_len-user
splunk_app_infrastructure		24	
splunk_web_access	16	3	5
splunkd	51		18
splunkd_access	33	4	18
splunkd_ui_access	27	6	5

splunk> .conf23

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

More references in addendum

1) Which fields…

1.4) Which fields **contain values matching a reference list of values?**

- *fieldToCheck* - Table of values in data to check against our supplied list
- *ComparisonList* - delimited set of strings to check against our *fieldToCheck* values.
- *ValueRegex* - Can be any regex pattern to ensure expected results for each "match" check.
- *flag_IsInList* - Uses *mvfind()* function to check each of the *ComparisonList* values for the Regex.

```
| makeresults | fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";") | mvexpand fieldToCheck
| foreach fieldToCheck
  [| eval ComparisonList = split("LIST::OF::VALUES", "::")
  | eval ValueRegex = "^" . '<<FIELD>>' . "$"
  | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )
  | fields - ComparisonList, ValueRegex]
```

fieldToCheck	flag_IsInList
IN	false
LIST	true
OF	true
MY	false
VALUES?	true

Notice "VALUES?" returns true because the '?' is read as part of the regex!

splunk> .conf23

Addendum
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values? **(Newline Alternative)**
- 1.5) differ between two groups?

1) Which fields…

1.4) Which fields *contain values matching a reference list of values?*

Newline-based Alternative Approach for Spreadsheets

Alternative approach allowing you to paste a newline-delimited set of values to check for in *ComparisonList,* like if it was copied out of a spreadsheet.

- Utilizes the | `makemv tokenizer=` functionality to build a regex capture group on everything that isn't a new line.

```
| makeresults
| fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";")
| mvexpand fieldToCheck
| foreach fieldToCheck
    [| eval ComparisonList = "LIST
OF
NEWLINE
VALUES"
    | makemv ComparisonList tokenizer="([^\n]+)"
    | eval ValueRegex = "^" . '<<FIELD>>' . "$"
    | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )
    | fields - ComparisonList, ValueRegex]
```

fieldToCheck	flag_IsInList
IN	false
LIST	true
OF	true
MY	false
VALUES?	true

splunk> .conf23

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

More references in addendum

1) Which fields…

1.5) Which fields **differ between two groups? (What differs within a group?)**

```
| eventstats dc(*) AS *_dc BY group_key
| foreach *_dc
   [| eval field_diffs = if( '<<FIELD>>' > 1, mvappend(field_diffs, "<<MATCHSTR>>"), field_diffs)]
| table group_key, field_diffs, *
```

group_key	field_diffs	hostname	hostname_dc	owner	owner_dc	src_ip	src_ip_dc
splunk	owner src_ip	SPL01	1	Wonder Mike	2	127.0.0.1	2
splunk	owner src_ip	SPL01	1	Ryan Wood	2	192.168.0.1	2
endpoint		EP01	1	Ryan Wood	1	1.1.1.1	1

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

2) Standard Format Output (JSON)

2) Standard Format Output (JSON)
Format each field, then compile it into the desired format

type	hostname	location	dest_ip	src_ip
Search Head	SPL-SH01	US	192.168.0.1	10.10.0.10
Indexer	SPL-IDX01	APAC	192.168.0.2	10.10.0.11
Indexer	SPL-IDX02	US	192.168.0.3	10.10.0.12

```
| foreach *
   [| eval <<FIELD>>_json = if( len('<<FIELD>>') < 1, null(), "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\"" )]
```

type_json	hostname_json	location_json	dest_ip_json	src_ip_json
"type":"Search Head"	"hostname":"SPL-SH01"	"location":"US"	"dest_ip":"192.168.0.1"	"src_ip":"10.10.0.10"
"type":"Indexer"	"hostname":"SPL-IDX01"	"location":"APAC"	"dest_ip":"192.168.0.2"	"src_ip":"10.10.0.11"
"type":"Indexer"	"hostname":"SPL-IDX02"	"location":"US"	"dest_ip":"192.168.0.3"	"src_ip":"10.10.0.12"

```
| foreach *_json
   [| eval json_full = mvappend(json_full, '<<FIELD>>')]
| eval json_full = "{" . mvjoin(json_full, ",") . "}"
```

json_full
{"dest_ip":"192.168.0.1", "hostname":"SPL-SH01", "location":"US", "src_ip":"10.10.0.10", "type":"Search Head"}
{"dest_ip":"192.168.0.2", "hostname":"SPL-IDX01", "location":"APAC", "src_ip":"10.10.0.11", "type":"Indexer"}
{"dest_ip":"192.168.0.3", "hostname":"SPL-IDX02", "location":"US", "src_ip":"10.10.0.12", "type":"Indexer"}

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

2) Standard Format Output (JSON)

3) REST - Consolidating columns

More references in addendum

3) REST - Consolidating Columns

Compiling the many savedsearches.conf fields into one for readability.

```
| rest /servicesNS/-/-/saved/searches splunk_server=* count=0
|  fields title, description, disabled, allow_skew, is_visible, max_concurrent, realtime_schedule,
request.*, run_*, schedule_*, author, federated.provider, workload_pool, indicator_name, related_dashboard
| foreach allow_skew, is_visible, max_concurrent, realtime_schedule, request.*, run_*, schedule_*, author,
federated.provider, workload_pool, indicator_name, related_dashboard
   [| eval misc_props_mv = if(isnotnull('<<FIELD>>'),
                mvappend(misc_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), misc_props_mv)]
| fields - allow_skew, is_visible, max_concurrent, realtime_schedule, request.*, run_*, schedule_*, author,
federated.provider, workload_pool, indicator_name, related_dashboard
| table title, description, disabled, misc_props_mv
```

title	description	disabled	misc_props_mv
DMC Alert - Abnormal State of Indexer Processor	One or more of your indexers is reporting an abnormal state.	1	"allow_skew":"0" "is_visible":"1" "max_concurrent":"1" "realtime_schedule":"1" "request.ui_dispatch_app":"" "request.ui_dispatch_view":"" "run_n_times":"0" "run_on_startup":"0" "schedule_as":"auto" "schedule_priority":"default" "schedule_window":"0" "author":"nobody" "workload_pool":""

Full Utility Version of this Example in Utility Addendum:
REST - Savedsearches Properties Generation

With great power…

With great power…

Comes an Out-Of-Memory Risk

Search Performance Risk

Consider these three similar searches

```
index=_internal
| foreach user
    [| eval <<FIELD>> = if( len('<<FIELD>>') > 0, '<<FIELD>>', null() )]
| stats count by user
```

4 results by scanning **930,954** events in **8.944** seconds

```
index=_internal
| fields user
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') > 0, '<<FIELD>>', null() )]
| stats count by user
```

4 results by scanning **931,289** events in **11.295** seconds

```
index=_internal
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') > 0, '<<FIELD>>', null() )]
| stats count by user
```

4 results by scanning **930,129** events in **126.097** seconds

Reduce Memory Cost

Large Input = High Memory Consumption

- Be mindful of the size of the input dataset you're running foreach on
- Seek to reduce the input size where you can
  - `| fields` command is your best friend.
- Drop fields you don't need within the foreach search

```
| makeresults | fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";") | mvexpand fieldToCheck
| foreach fieldToCheck
  [| eval ComparisonList = split("LIST::OF::VALUES", "::")
   | eval ValueRegex = "^" . '<<FIELD>>' . "$"
   | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )
   | fields - ComparisonList, ValueRegex]
```

fieldToCheck	flag_IsInList
IN	false
LIST	true
OF	true
MY	false
VALUES?	true

```
| makeresults | fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";") | mvexpand fieldToCheck
| eval ComparisonList = split("LIST::OF::VALUES", "::")
| foreach fieldToCheck
 [| eval ValueRegex = "^" . '<<FIELD>>' . "$"
  | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )]
```

ComparisonList	ValueRegex	fieldToCheck	flag_IsInList
LIST OF VALUES	^IN$	IN	false
LIST OF VALUES	^LIST$	LIST	true
LIST OF VALUES	^OF$	OF	true

splunk> .conf23

Usecases

Level 4

4

Dynamic References & Adaptive Design

Versatile Logic for Dynamic Thresholding & Advanced Insights

Utility/Benefits

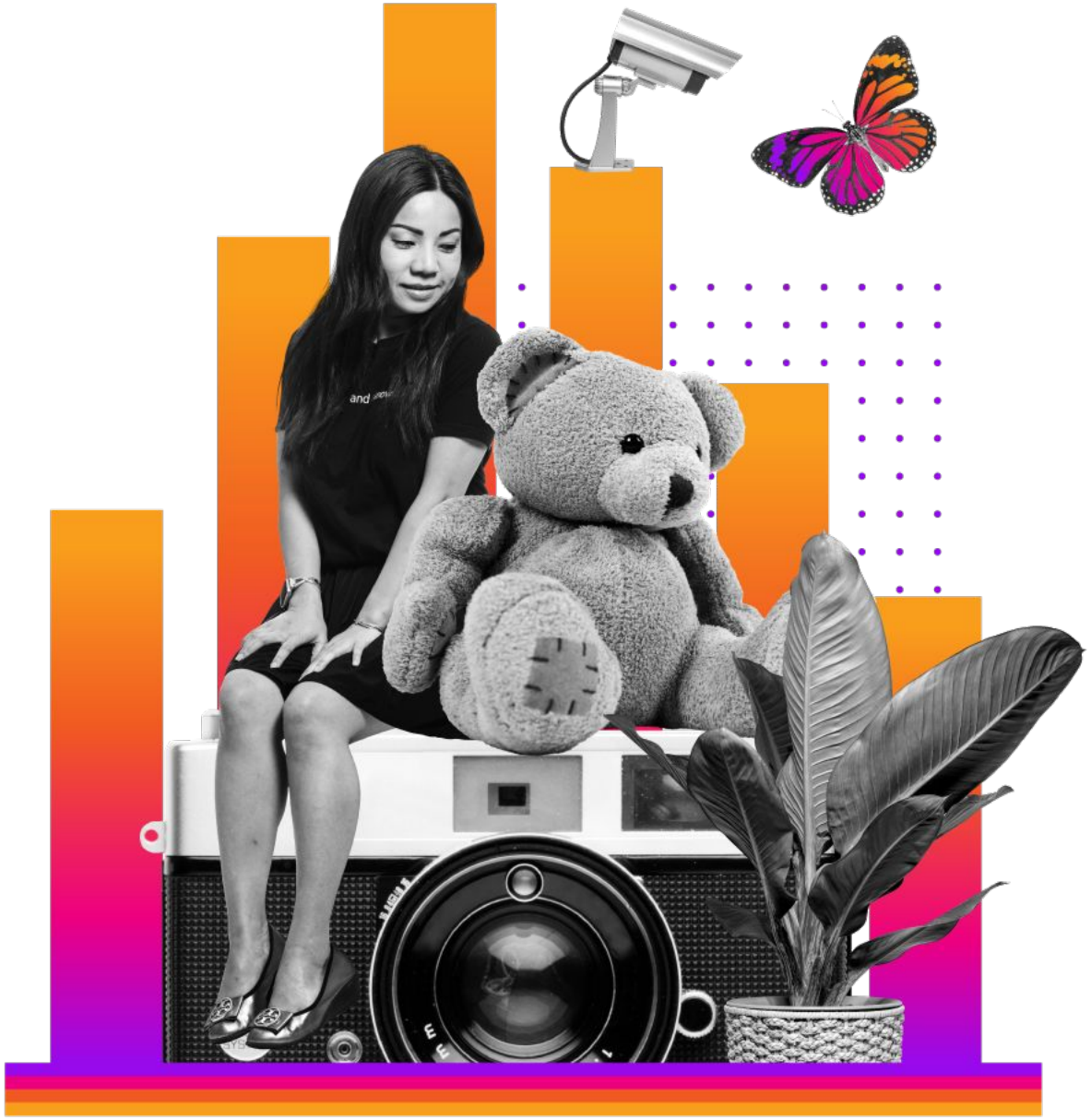
Level 4
Dynamic References & Adaptive Design

- Complex, adaptive logic that can't easily be achieved without foreach
- Usage of Advanced options to dynamically reference segments of wildcard
  - <<MATCHSTR>>
  - <<MATCHSEG#>>

<<MATCH*>>

Level 4
Dynamic References & Adaptive Design

- The '<<MATCH*>>' references allow accessing the segment of the wildcard pattern passed to foreach to operate on in our logic.
  - '<<MATCHSTR>>' - The entire segment represented by the wildcard "*"
  - '<<MATCHSEG#>>' - The specific wildcard segment when multiple wildcards are used - "command.*.*" creates 2 segments.



4

Dynamic References & Adaptive Design

Versatile Logic for Dynamic Thresholding & Advanced Insights

Level 4 Examples

Dynamic References

splunk> .conf23

Examples
Level 4

Generates epoch times for 7 days **before/after** beginning of month across the last 3 months.

- Uses <<MATCHSTR>> to reference the base_* fields
- Helpful for running historical searches, correlation boundaries

Generate Start/End Timestamps Using <<MATCHSTR>>

```
| makeresults
| eval base_Month1 = relative_time(now(), "@mon")
| eval base_Month2 = relative_time(now(), "-1mon@mon")
| eval base_Month3 = relative_time(now(), "-2mon@mon")
| foreach base_*
   [| eval <<FIELD>> = round('<<FIELD>>', 0)
   | eval <<MATCHSTR>>_earliest = round(<<FIELD>> - 604800, 0)
   | eval <<MATCHSTR>>_latest = round(<<FIELD>> + 604800, 0)]
```

base_Month1	base_Month2	base_Month3
1685577600	1682899200	1680307200
2023-06-01	2023-05-01	2023-04-01

Month1_earliest	Month1_latest	Month2_earliest	Month2_latest	Month3_earliest	Month3_latest
1684972800	1686182400	1682294400	1683504000	1679702400	1680912000
2023-05-25	2023-06-08	2023-04-24	2023-05-08	2023-03-25	2023-04-08

Examples
Level 4

Establish avg() and p90() BY group-by, then compare each field to those thresholds

Using Dynamically Generated Dataset Thresholds
Comparing field values to the p90(), avg() of the group-by

```
| makeresults count=50
| eval num1 = (random() % 100) + 1, num2 = (random() % 100) + 1,
       num3 = (random() % 100) + 1, num4 = (random() % 100) + 1
| streamstats reset_after="(group > 1)" count AS group
| eventstats avg(*) AS *_avg, p90(*) AS *_p90 BY group
| foreach *
  [| eval less_than_avg=if('<<FIELD>>' < '<<FIELD>>_avg',
            mvappend(less_than_avg, "<<FIELD>>"), less_than_avg)
   | eval less_than_p90=if('<<FIELD>>' < '<<FIELD>>_p90',
            mvappend(less_than_p90, "<<FIELD>>"), less_than_p90)]
```

group	less_than_avg	less_than_p90	num1	num1_avg	num1_p90	num2	num2_avg	num2_p90
1	num2 num4	num1 num2 num4	89.0	54.1	89.6	4.0	46.9	77.6
2	num3	num2 num3 num4	99.0	55.0	85.2	81.0	54.2	86.2
1	num1	num1 num2 num4	34.0	54.1	89.6	48.0	46.9	77.6
2	num1 num4	num1 num2 num3 num4	36.0	55.0	85.2	82.0	54.2	86.2

zScore Calculation

PLA1159C - Lesser Known Search Commands Part 1

Courtesy of Kyle Smith

```
index=_internal sourcetype=splunkd component=Metrics group=per_sourcetype_thruput
| eval series = replace(series, ":", "_")
| timechart span=60m avg(kbps) AS avg_kbps BY series useother=f
| streamstats window=720 mean(*) AS MEAN*, stdev(*) AS STDEV*
| foreach *
   [| eval Z_<<FIELD>> = (( '<<FIELD>>' - 'MEAN<<MATCHSTR>>') / 'STDEV<<MATCHSTR>>')]
| fields _time, Z*
| eval upper=3, lower=-3
```

This **foreach** command analyzes sourcetype throughput, to detect for anomalies. A **timechart** with the **avg_kbps** of each sourcetype leads into a **streamstats** to get the mean and standard deviation for the rolling 720 window.

The **foreach** command iterates over any of the found fields (sourcetypes), and calculates the Z-Score.

We made it!

splunk> .conf23

Wrap-up

#Content - What we Covered

A look back at the progression

Complexity

1

Convert | eval's to a Template Approach

Clean up your SPL by merging repetitive eval logic.

```
| foreach host, hostname, user
   [| eval <<FIELD>> = lower('<<FIELD>>')]
```

2

Field Validation & Batch Transformations

Normalize your fields to ensure your logic works like it should.

```
| foreach *
   [| eval <<FIELD>>=if(len('<<FIELD>>') < 1, null(), '<<FIELD>>')]
```

3

Present Your Data through Aggregation

Compile field values for review or to output in a specific format

```
| foreach messages.*
   [| eval all_messages_mv=mvdedup(mvappend(messages, '<<FIELD>>'))]
```

4

Dynamic References & Adaptive Design

Versatile Logic for Dynamic Thresholding & Advanced Insights

```
| eventstats mean(*) AS MEAN*, stdev(*) AS STDEV* BY group
| foreach *
[| eval Z_<<FIELD>>=(('<<FIELD>>'-'MEAN<<MATCHSTR>>') / 'STDEV<<MATCHSTR>>')]
```

Key Takeaways

- Foreach is a cool command that can do really interesting things to provide confidence.
- This is like a new platform to see how you can put the puzzle pieces together.
  - See what you come up with.
- Be mindful of your search performance, reduce the input size where you can.
- Go get the slides!
  GitHub:
  https://github.com/TheWoodRanger/presentation-splunk_foreach

splunk> .conf23

Call to Action

- **Go get the latest version of the slides! The current version of the deck you're viewing is v2.0**
  GitHub: https://github.com/TheWoodRanger/presentation-splunk_foreach

- **Put those references to practice! Lots of wonderfully interesting puzzle pieces to fit together in whatever ways you can come up with!**

- **Reach out!**
  - Splunk UserGroups Slack: **TheWoodRanger**
  - Email: ryan.wood@guidepointsecurity.com
  - LinkedIn: https://www.linkedin.com/in/ryanwoodranger/

Q&A

splunk> .conf23

More Resources

- **Github for this presentation's materials and updates:**
  https://github.com/TheWoodRanger/presentation-splunk_foreach
- Splunk Field Analysis for the Advanced Engineer
  - https://github.com/TheWoodRanger/splunk_fields_analysis_presentation
  - Advanced SPL deep dive on the | `fieldsummary` command.
- Clara-Fication Blog Posts
  (Clara Merriman's Presentations are A+)

More awesome presentations listed on the github.

Other Resources and Questions

- https://docs.splunk.com
- https://answers.splunk.com
- https://lantern.splunk.com
- Slack!

  https://splk.it/slack
- Join a User Group!

  https://usergroups.splunk.com

  https://www.splunk.community

Thank You

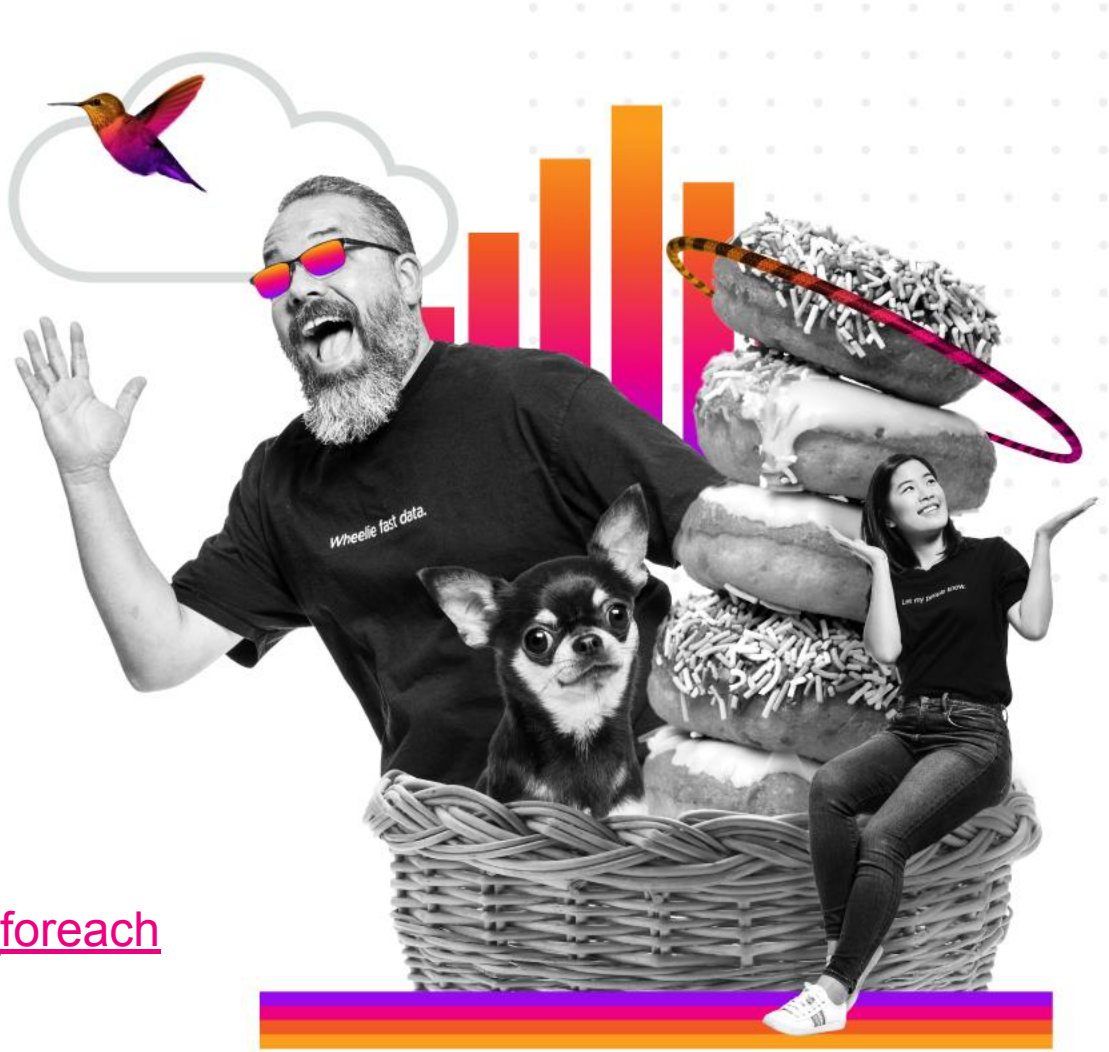
splunk> .conf23

Addendum Slides

More SPL & Information

Notes, Updates, etc. on Github:

https://github.com/TheWoodRanger/presentation-splunk_foreach



splunk> .conf23

Addendum Outline

Content Overview

1) Foreach Usage Warnings

- Do's and Don'ts
- Notes on Reducing Performance Risks

2) Tips & Tricks

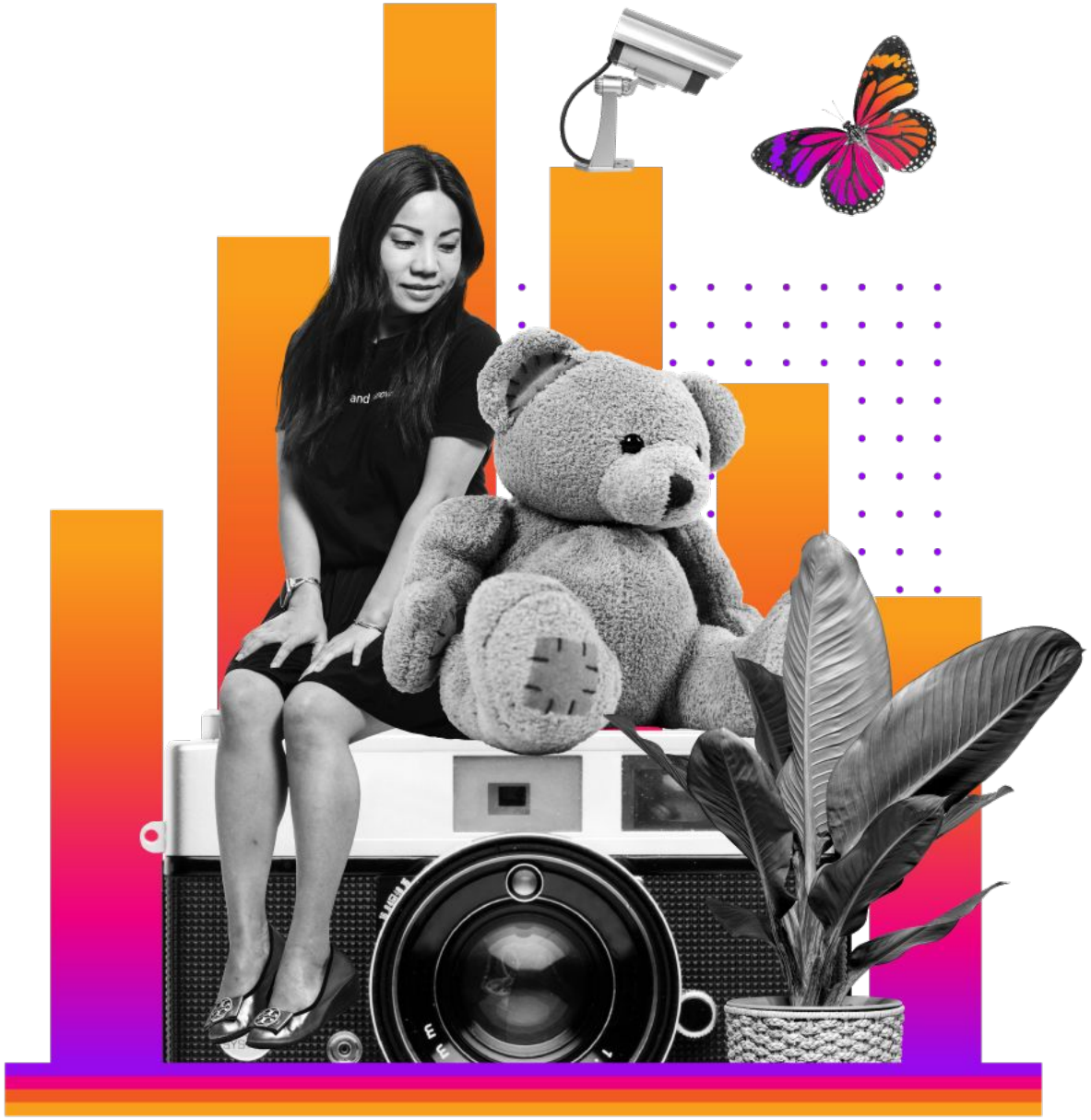
- Methods to Preserve Specific Field Values within Foreach
- Optimization Strategies (TBA)
- Output Formatting Methods (TBA)

3) Level 2-4 Addendum Usecases/References

- Expansions to Examples slides within main presentation
- Addendum Slides for Additional Examples (marked Addendum)

4) SPL Utilities/Reports

- Full Utility report queries meant to make your life a bit easier
- Paste-and-Run SPL, output screenshots.



Foreach Usage Warnings

Addendum

splunk> .conf23

| foreach Do's and Don'ts

Do's and Don'ts

Do…

- Always wrap the <<FIELD>> reference in 'single quotes' on the right side of the eval.
- Reduce the size of the data input before | foreach
  - `| fields` command is your best friend.
- Consider what fields need to be evaluated and limit the scope of foreach to those fields.
- Reduce the data within the foreach subsearch where the opportunity is obvious. (Readability vs Optimization) Examples:
  - If you are compiling field values into a new field and don't need the original fields after the foreach command, drop the original fields before the end of the foreach subsearch: `| fields` command
  - If you create any new fields within the subsearch and don't need them on the final foreach output, drop them before the end of the foreach subsearch: `| fields` command

Don't…

- Run | foreach on raw event searches
  - If you need to, reduce the fields before foreach by utilizing the `| fields` command
- Use the <<MATCH*>> references without an understanding of exactly what they do
  - Always validate the output is what you expect it to be. It's easy to make mistakes with these references.

Reducing Performance Risk

1) Reduce the number of fields iterated on!
   Utilize | `fields` command or pass more restrictive match pattern to | `foreach`

2) Drop fields within the foreach if you don't need them in the output. Use `fields` or `table` command

3) Take extreme care when using `{field}` references, as the risk of recursive explosion increases. A search demonstrating this risk (safely):

```
| makeresults count=5 | eval random = random() % 100
| foreach * [ eval {<<FIELD>>}=<<FIELD>> ] | format
```

4) Avoid embedding a foreach command within another foreach command

5) Avoid putting a foreach within a subsearch in the initial search filter line of your SPL
   * this search has **actually** crashed a SH (do not run):

```
index=* [ | search index=* | foreach * [ eval {<>}=<> ] ]
```

Reduce Memory Cost

Large Input = High Memory Consumption

- Be mindful of the size of the input dataset you're running foreach on
- Seek to reduce the input size where you can
  - `| fields` command is your best friend.
- Drop fields you don't need within the foreach search

```
| makeresults | fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";") | mvexpand fieldToCheck
| foreach fieldToCheck
  [| eval ComparisonList = split("LIST::OF::VALUES", "::")
   | eval ValueRegex = "^" . '<<FIELD>>' . "$"
   | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )
   | fields - ComparisonList, ValueRegex]
```

fieldToCheck	flag_IsInList
IN	false
LIST	true
OF	true
MY	false
VALUES?	true

```
| makeresults | fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";") | mvexpand fieldToCheck
| eval ComparisonList = split("LIST::OF::VALUES", "::")
| foreach fieldToCheck
 [| eval ValueRegex = "^" . '<<FIELD>>' . "$"
  | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )]
```

ComparisonList	ValueRegex	fieldToCheck	flag_IsInList
LIST OF VALUES	^IN$	IN	false
LIST OF VALUES	^LIST$	LIST	true
LIST OF VALUES	^OF$	OF	true

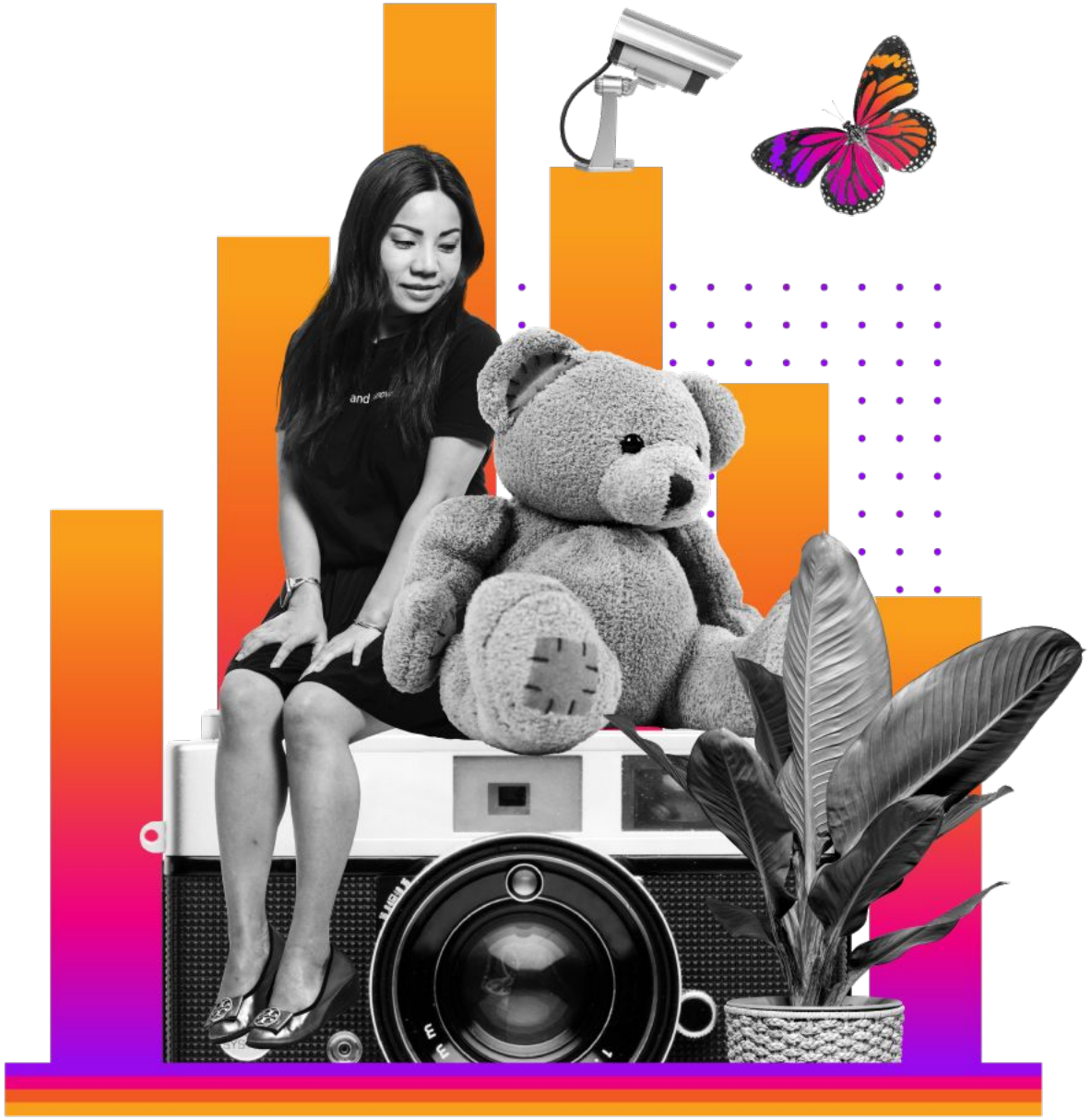
splunk> .conf23

Wildcard foreach Usage - Overview of Implementation

| foreach

1) Identify common transformations that need to be applied across a number of fields.
   a) Tip - Use Level 3 Methodologies
2) Generate/compile sample data representing “Before” state to build and test logic on.
   a) Tip - `| loadjob` is your best friend!
   b) Tip - Summary indexing via `| collect`
3) Build non-foreach “verbose” eval SPL to serve as a baseline - this will help identify if foreach approach is significantly slower.
   a) Reference Utility '*Generate SPL using SPL'* for an easy way to compile this verbose approach
4) Build foreach SPL with the identified transformations from Step 1 implemented.
5) Run both “verbose” and “foreach” SPL queries on same input data - **Validate results are consistent between the two approaches.**
   a) Reference Utility ‘*Check two sets of search results are the same*’ for a way to do this per-row.
6) Run both “verbose” and “foreach” SPL queries on same input data to confirm impact from foreach usage. *AFTER confirming results are consistent between the two queries!*
   a) **`Job performance properties` can help understand performance cost of foreach vs eval:**

```
| rest /servicesNS/-/-/search/jobs/<sid>
| fields sid, title, published, runDuration, performance.command.eval*,
performance.command.foreach*
```



Tips & Tricks

Miscellaneous bits to help you on the journey.

Note: This section will continue expanding as I get feedback/questions/time to build more refs.

splunk> .conf23

Tips & Tricks

Methods to Preserve Specific Field Values when Using Foreach

1) Direct String Comparisons
- 1.1) Equals Name
- 1.2) eval IN Name
- 1.3) Compare to Value

2) Regex Comparison
- 2.1) Regex match on Field Name
- 2.2) Comparison to Value using Regex

1) Direct String Comparison

(non-regex)

1.1) Equals Name

- Direct comparison against field name using equals.

If *field name is "sourcetype",* preserve value, otherwise set to length of value.

```
| foreach *
   [| eval <<FIELD>> = if( "<<FIELD>>"="sourcetype", '<<FIELD>>', len(mvjoin('<<FIELD>>', "")) )]
```

1.2) eval IN Name

- Direct comparison against field name but accepts list of values rather than needing OR.

If *field name is "sourcetype", "index", "host",* preserve value, otherwise set to length of value.

```
| foreach *
   [| eval <<FIELD>> =
        if( IN("<<FIELD>>", "sourcetype", "index", "host"),
            '<<FIELD>>',
            len(mvjoin('<<FIELD>>', ""))
          )]
```

1.3) Compare to Value

- Comparison against field *value* to determine whether to preserve.

If *numeric* field value *is greater than 10*, preserve, else set value to null so it doesn't affect distribution. Avoid overwriting non-numeric fields by using the correct assertion.

```
| foreach *
    [| eval <<FIELD>> = if( isnum('<<FIELD>>') AND '<<FIELD>>' < 11, null(), '<<FIELD>>')]
```

Tips & Tricks

Methods to Preserve Specific Field Values when Using Foreach

1) Direct String Comparisons
- 1.1) Equals Name
- 1.2) eval IN Name
- 1.3) Compare to Value

2) Regex Comparison
- 2.1) Regex match on Field Name
- 2.2) Comparison to Value using Regex

2) Regex-based Comparison

2.1) Regex match() on Field Name

- Use regex match() function to compare against field name using any pattern.

If field name matches any regex pattern in following: *stanza, id, appName, splunk_server,* do not add field value to aggregated *stanza_properties* field. Otherwise append to aggregated column.

```
| foreach *
    [| eval stanza_properties = case( match("<<FIELD>>", "^(stanza|id|appName|splunk_server)"), stanza_properties,
                                      isnotnull('<<FIELD>>'), mvappend(stanza_properties, "<<FIELD>>:::" . '<<FIELD>>'),
                                      true(), stanza_properties)]
```

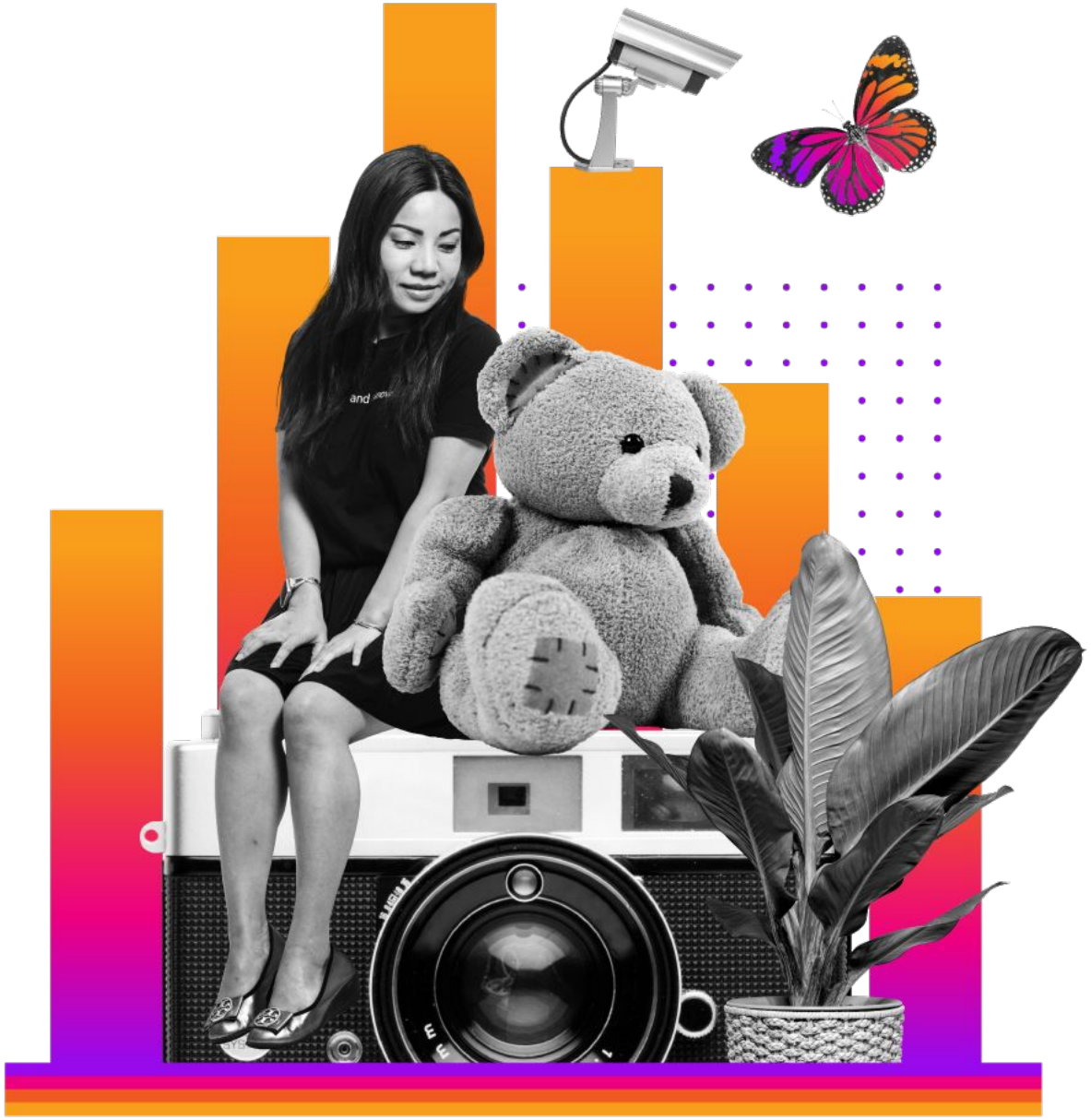
2.2) Comparison to Value Using Regex

- Comparison against field *value* to determine whether to preserve.

- If field value matches `ipv4:port` syntax, preserve value, otherwise concatenate IP and Port fields.
- Use coalesce to pass multiple potential Port fields.
- End with `true()` to ensure other fields are preserved.

```
| foreach *
    [| eval <<FIELD>> =
        case( match('<<FIELD>>', "\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}:\d+"), '<<FIELD>>',
        match('<<FIELD>>', "\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$"), '<<FIELD>>' . ":" . coalesce(port, Port, http_port, "null"),
        true(), '<<FIELD>>' )
    ]
```

splunk> .conf23



Additional Use Cases & SPL References

Addendum snippets and sample code

Note: Example slides from earlier in deck are copied here for ease of reference.

splunk> .conf23

Regarding Addendum Use Cases & Examples

There was a massive amount of content to review and compile for this presentation. Please refer to the github repo for this talk to obtain any updates or critical aspects that were not included in the original presentation.

Updated versions of the Slides will be posted here as well.

https://github.com/TheWoodRanger/presentation-splunk_foreach

| foreach *field*, set true null()


```
...
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' ) ]
```

For each field:

- 1) Check if the 'value' length is less than 1
- 2) Set value to `null()` if true.
- 3) Preserve field value in all other cases.

| foreach *field*, ~~set true null()~~ set "string"

~CRITICAL TIP~

length() of null() is still null()!

To insert a "string" instead of setting null(), use the **top** pattern instead:

```
| foreach *
   [| eval <<FIELD>> = if( len('<<FIELD>>') < 1 OR isnull('<<FIELD>>'), "unknown", '<<FIELD>>' ) ]
```

```
| foreach *
   [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, "unknown", '<<FIELD>>' ) ]
```

^ *This bottom pattern **will not insert a "value" into null fields.***

Examples
Level 2

1) Field Transformations
- 1.1) Handle zero-length fields
- 1.2) Prefix field values with field name

More references in addendum

1) Field Transformations
(non-regex)

1.1) Handle zero-length fields

```
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' ) ]
```

1.2) Prefix field *values* with field *name* – `field="value"`

```
| foreach *
    [| eval <<FIELD>> = "<<FIELD>>=\"" . '<<FIELD>>' . "\""]
```

hostname	src_ip
SPL-SH01	10.10.0.10
SPL-IDX01	10.10.0.11
SPL-IDX02	10.10.0.12

hostname	src_ip
hostname="SPL-SH01"	src_ip="10.10.0.10"
hostname="SPL-IDX01"	src_ip="10.10.0.11"
hostname="SPL-IDX02"	src_ip="10.10.0.12"

Examples
Level 2

1) Field Transformations
- 1.1) Handle zero-length fields
- 1.2) Prefix field values with field name

2) Batch Regex Replacements
- 2.1) Remove all non-ascii characters
- 2.2) Add escape backslash to special characters

More references in addendum

2) Batch Regex Replacements

2.1) Remove all non-ascii characters

```
| foreach *
    [| eval <<FIELD>> = if( isnotnull('<<FIELD>>'), replace('<<FIELD>>', "[^[:ascii:]]+", ""),  null() )]
```

2.2) Add escape backslash to special characters *(if not already escaped)*
- using | rex mode=sed OR | eval replace

```
| foreach *
    [| rex mode=sed field="<<FIELD>>" "s/([^\\\](?!\\\))(\W)/\1\\\\\2/g"
  OR | eval <<FIELD>> = replace('<<FIELD>>', "([^\\\](?!\\\))(\W)", "\1\\\\\2")
    ]
```

splunk_major_breakers	string	string2
\<>[](){\\}!?;,"&	Escapes are important.	But\ careful\ doubling\ up!
\<\>\[\]\(\)\{\\}\!\?\;\,\"\&	Escapes\ are\ important\.	But\ careful\ doubling\ up\!\

splunk> .conf23

Examples
Level 2

1) Field Transformations
   - 1.1) Handle zero-length fields
   - 1.2) Prefix field values with field name

2) Batch Regex Replacements
   - 2.1) Remove all non-ascii characters
   - 2.2) Add escape backslash to special characters

3) Multivalue Field Handling
   - 3.1) Concatenate multivalued items together, Split single-value string into multivalue list

More references in addendum

3) Multivalue Field Handling

3.1) Concatenate multivalued items together / Split single-value string into a multivalue list

```
| foreach *
    [| eval <<FIELD>> = if( mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', "::delim::"), '<<FIELD>>' )]

| foreach *
    [| eval <<FIELD>> = if( match('<<FIELD>>', "::delim::"), split('<<FIELD>>', "::delim::"), '<<FIELD>>')]
```

hostname	src_ip
SPL-IDX01	10.10.0.10
SPL-IDX02	10.10.0.11
SPL-SH01	10.10.0.12

hostname	src_ip
SPL-IDX01::delim::SPL-IDX02::delim::SPL-SH01	10.10.0.10::delim::10.10.0.11::delim::10.10.0.12

hostname	src_ip
SPL-IDX01	10.10.0.10
SPL-IDX02	10.10.0.11
SPL-SH01	10.10.0.12

splunk> .conf23

Addendum
Level 2

1) Field Transformations
- 1.1) Handle zero-length fields
- 1.2) Prefix field values with field name

2) Batch Regex Replacements
- 2.1) Remove all non-ascii characters
- 2.2) Add escape backslash to special characters

3) Multivalue Field Handling
- 3.1) Concatenate multivalued items together, Split single-value string into multivalue list
- 3.2) Limit to first 5 values in multivalue list
- 3.3) Prep Results for CSV Export/Summary Index

3) Multivalue Field Handling (Addendum)

3.2) Limit to first 5 values in multivalued list

```
...
| foreach *
    [| eval <<FIELD>> = if(mvcount('<<FIELD>>') > 5,
                           mvappend(mvindex('<<FIELD>>', 0, 4), "[trimmed values]"), '<<FIELD>>')
    ]
```

numbers	numbers_trimmed
47	47
58	58
95	95
1	1
77	77
60	[trimmed values]
29	
71	
75	
51	
74	
95	
80	
23	
87	
56	
41	
82	

splunk> .conf23

Addendum
Level 2

1) Field Transformations
- 1.1) Handle zero-length fields
- 1.2) Prefix field values with field name

2) Batch Regex Replacements
- 2.1) Remove all non-ascii characters
- 2.2) Add escape backslash to special characters

3) Multivalue Field Handling
- 3.1) Concatenate multivalued items together, Split single-value string into multivalue list
- 3.2) Limit to first 5 values in multivalue list
- 3.3) Prep Results for CSV Export

3) Multivalue Field Handling (Addendum)

3.3) Prep Results for CSV Export to maintain list-of-values format

- The literal newline in doublequotes here is inserted using mvjoin
- Inserting a newline this way maintains the appearance of a multivalue list when exporting.
- This can be very useful for custom formatting when building MV fields for readability. See below for demonstration.

```
| eval comment=if(true(), null(), "For CSV exporting: if any multivalued fields, convert to single value field with newline as delimiter. The mvjoin() function reads literally, so the explicit newline here is important." )
| foreach *
    [| eval <<FIELD>> = if( mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', "
"), '<<FIELD>>') ]
```

```
| makeresults | fields - _time
| eval colors_concat = "red,blue,purple,green"
| eval colors_mv = split(colors_concat, ",")
| eval colors_explicitNewLine = mvjoin(colors_mv, "
")
| eval colors_withHeader_mv = mvappend("-----", "This is a list of colors:", colors_mv, "~~ And That's all! ~~")
| eval colors_withHeader_explicitNewLine = mvjoin(colors_withHeader_mv, "
")
| foreach colors*
    [| eval colors_fields_mvcount = mvappend(colors_fields_mvcount, "<<FIELD>> has " . mvcount('<<FIELD>>') . " multivalued items.")]
| table colors_concat, colors_mv, colors_explicitNewLine, colors_withHeader_mv, colors_withHeader_explicitNewLine, colors_fields_mvcount
```

Last 24 hour

✓ 1 result (7/31/23 1:00:00.000 AM to 8/1/23 1:29:12.000 AM) No Event Sampling ▾ Job ▾

Events Patterns **Statistics (1)** Visualization

10 Per Page ▾ Format Preview ▾

colors_concat	colors_mv	colors_explicitNewLine	colors_withHeader_mv	colors_withHeader_explicitNewLine	colors_fields_mvcount
red,blue,purple,green	red blue purple green	red blue purple green	----- This is a list of colors: red blue purple green ~~ And That's all! ~~	----- This is a list of colors: red blue purple green ~~ And That's all! ~~	colors_concat has 1 multivalued items. colors_explicitNewLine has 1 multivalued items. colors_mv has 4 multivalued items. colors_withHeader_explicitNewLine has 1 multival colors_withHeader_mv has 7 multivalued items.

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

More references in addendum

1) Which fields…

1.1) Which fields **are Not Null?**

```
| foreach *
  [| eval notnull_fields =
          if( isnotnull('<<FIELD>>'), mvappend(notnull_fields, "<<FIELD>>"), notnull_fields)]
```

1.2) Which fields **are over a certain length?**

```
| foreach *
   [| eval long_fields = if( len('<<FIELD>>') > 499, mvappend(long_fields, "<<FIELD>>"), long_fields)]
```

1.3) Which fields **are the longest?**

- Overwrite field value with field length, then run calculations BY sourcetype

```
| foreach *
   [| eval <<FIELD>> =
        if( IN("<<FIELD>>", "sourcetype"), '<<FIELD>>', len(mvjoin('<<FIELD>>', "")) )
   ]
| stats max(*) AS max_len-* BY sourcetype
```

sourcetype	max_len-file	max_len-method	max_len-user
splunk_app_infrastructure		24	
splunk_web_access	16	3	5
splunkd	51		18
splunkd_access	33	4	18
splunkd_ui_access	27	6	5

splunk> .conf23

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

More references in addendum

1) Which fields…

1.4) Which fields **contain values matching a reference list of values?**

- *fieldToCheck* - Table of values in data to check against our supplied list
- *ComparisonList* - delimited set of strings to check against our *fieldToCheck* values.
- *ValueRegex* - Can be any regex pattern to ensure expected results for each "match" check.
- *flag_IsInList* - Uses *mvfind()* function to check each of the *ComparisonList* values for the Regex.

```
| makeresults | fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";") | mvexpand fieldToCheck
| foreach fieldToCheck
  [| eval ComparisonList = split("LIST::OF::VALUES", "::")
  | eval ValueRegex = "^" . '<<FIELD>>' . "$"
  | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )
  | fields - ComparisonList, ValueRegex]
```

fieldToCheck	flag_IsInList
IN	false
LIST	true
OF	true
MY	false
VALUES?	true

Notice "VALUES?" returns true because the '?' is read as part of the regex!

splunk> .conf23

Addendum
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values? **(Newline Alternative)**
- 1.5) differ between two groups?

1) Which fields…

1.4) Which fields *contain values matching a reference list of values?*

Newline-based Alternative Approach for Spreadsheets

Alternative approach allowing you to paste a newline-delimited set of values to check for in *ComparisonList,* like if it was copied out of a spreadsheet.

- Utilizes the | `makemv tokenizer=` functionality to build a regex capture group on everything that isn't a new line.

```
| makeresults
| fields - _time
| eval fieldToCheck = split("IN;LIST;OF;MY;VALUES?", ";")
| mvexpand fieldToCheck
| foreach fieldToCheck
    [| eval ComparisonList = "LIST
OF
NEWLINE
VALUES"
    | makemv ComparisonList tokenizer="([^\n]+)"
    | eval ValueRegex = "^" . '<<FIELD>>' . "$"
    | eval flag_IsInList = if( isnum(mvfind(ComparisonList, ValueRegex)), "true", "false" )
    | fields - ComparisonList, ValueRegex]
```

fieldToCheck	flag_IsInList
IN	false
LIST	true
OF	true
MY	false
VALUES?	true

splunk> .conf23

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

More references in addendum

1) Which fields…

1.5) Which fields **differ between two groups?**

```
| eventstats dc(*) AS *_dc BY group_key
| foreach *_dc
   [| eval field_diffs = if( '<<FIELD>>' > 1, mvappend(field_diffs, "<<MATCHSTR>>"), field_diffs)]
| table group_key, field_diffs, *
```

group_key	field_diffs	hostname	hostname_dc	owner	owner_dc	src_ip	src_ip_dc
splunk	owner src_ip	SPL01	1	Wonder Mike	2	127.0.0.1	2
splunk	owner src_ip	SPL01	1	Ryan Wood	2	192.168.0.1	2
endpoint		EP01	1	Ryan Wood	1	1.1.1.1	1

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

2) Standard Format Output (JSON)

2) Standard Format Output (JSON)
Format each field, then compile it into the desired format

type	hostname	location	dest_ip	src_ip
Search Head	SPL-SH01	US	192.168.0.1	10.10.0.10
Indexer	SPL-IDX01	APAC	192.168.0.2	10.10.0.11
Indexer	SPL-IDX02	US	192.168.0.3	10.10.0.12

```
| foreach *
   [| eval <<FIELD>>_json = if( len('<<FIELD>>') < 1, null(), "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\"" )]
```

type_json	hostname_json	location_json	dest_ip_json	src_ip_json
"type":"Search Head"	"hostname":"SPL-SH01"	"location":"US"	"dest_ip":"192.168.0.1"	"src_ip":"10.10.0.10"
"type":"Indexer"	"hostname":"SPL-IDX01"	"location":"APAC"	"dest_ip":"192.168.0.2"	"src_ip":"10.10.0.11"
"type":"Indexer"	"hostname":"SPL-IDX02"	"location":"US"	"dest_ip":"192.168.0.3"	"src_ip":"10.10.0.12"

```
| foreach *_json
   [| eval json_full = mvappend(json_full, '<<FIELD>>')]
| eval json_full = "{" . mvjoin(json_full, ",") . "}"
```

json_full
{"dest_ip":"192.168.0.1", "hostname":"SPL-SH01", "location":"US", "src_ip":"10.10.0.10", "type":"Search Head"}
{"dest_ip":"192.168.0.2", "hostname":"SPL-IDX01", "location":"APAC", "src_ip":"10.10.0.11", "type":"Indexer"}
{"dest_ip":"192.168.0.3", "hostname":"SPL-IDX02", "location":"US", "src_ip":"10.10.0.12", "type":"Indexer"}

Examples
Level 3

1) Which fields…
- 1.1) are Not Null?
- 1.2) are over a certain length?
- 1.3) are the longest?
- 1.4) contain values in this list of values?
- 1.5) differ between two groups?

2) Standard Format Output (JSON)

3) REST - Consolidating columns

More references in addendum

3) REST - Consolidating Columns

Compiling the many savedsearches.conf fields into one for readability.

```
| rest /servicesNS/-/-/saved/searches splunk_server=* count=0
|  fields title, description, disabled, allow_skew, is_visible, max_concurrent, realtime_schedule,
request.*, run_*, schedule_*, author, federated.provider, workload_pool, indicator_name, related_dashboard
| foreach allow_skew, is_visible, max_concurrent, realtime_schedule, request.*, run_*, schedule_*, author,
federated.provider, workload_pool, indicator_name, related_dashboard
   [| eval misc_props_mv = if(isnotnull('<<FIELD>>'),
                mvappend(misc_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), misc_props_mv)]
| fields - allow_skew, is_visible, max_concurrent, realtime_schedule, request.*, run_*, schedule_*, author,
federated.provider, workload_pool, indicator_name, related_dashboard
| table title, description, disabled, misc_props_mv
```

title	description	disabled	misc_props_mv
DMC Alert - Abnormal State of Indexer Processor	One or more of your indexers is reporting an abnormal state.	1	"allow_skew":"0" "is_visible":"1" "max_concurrent":"1" "realtime_schedule":"1" "request.ui_dispatch_app":"" "request.ui_dispatch_view":"" "run_n_times":"0" "run_on_startup":"0" "schedule_as":"auto" "schedule_priority":"default" "schedule_window":"0" "author":"nobody" "workload_pool":""

Full Utility Version of this Example in Utility Addendum:

REST - Savedsearches Properties Generation

Addendum Level 3

REST - props.conf Compilation of Columns

Props has a lot of fields, so instead of viewing them all in columns, we compile them together into a single consolidated column for review.

```
| rest /servicesNS/-/-/configs/conf-props splunk_server=*
| eval comment = if( true(), null(), "
Note: You may need to aggregate results BY id - if you don't want to consider each splunk_server separately. Might
have issues scaling depending on size of your deployment.")
| rename title AS stanza, eai:acl.app AS appName, eai:acl.sharing AS acl_sharing
| fields - eai*
| eval comment = if( true(), null(), "
Set all empty string field values to true null.")
| foreach *
   [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>')]
| eval comment = if( true(), null(), "
Compile multivalue field of all the properties (fields) returned for each stanza. We concatenate the key and value
using quadruple-colons because quad colons is extremely unlikely to appear in any other context, goal being to
extract the key and value into fields on the other side of the stats transformation.")
| foreach *
   [| eval stanza_properties = case( match("<<FIELD>>", "^(stanza|id|appName|splunk_server)"), stanza_properties,
   isnotnull('<<FIELD>>'), mvappend(stanza_properties, "<<FIELD>>::::" . '<<FIELD>>'),
   true(), stanza_properties)]
| table stanza, appName, stanza_properties, splunk_server
```

splunk> .conf23

Addendum Level 3

REST - Compile Multiple Endpoints Together for Display

Merges different API endpoint return values together to normalize across multiple endpoints. Creates common fields to unify the different data sources.

```
| makeresults | where false()
| append
   [| rest splunk_server=local /servicesNS/-/-/data/props/calcfields
   | dedup id
   | rename field.name AS field_name, eai:acl.* AS acl_*, "*'*'*" AS "***", "*'*" AS "**"
   | table dataSource, title, field_name, stanza, type, attribute, acl_app, acl_owner, acl_sharing, value
   | eval dataSource = "props-calcfields"]
| append
   [| rest splunk_server=local /servicesNS/-/-/data/props/extractions
   | dedup id
   | rename eai:acl.* AS acl_*, "*'*'*" AS "***", "*'*" AS "**"
   | rex field=attribute "^[^-]+-(?<field_name>.+)"
   | table dataSource, title, field_name, stanza, type, attribute, acl_app, acl_owner, acl_sharing, value
   | eval dataSource = "props-extractions"]
| append
   [| rest splunk_server=local /servicesNS/-/-/data/props/fieldaliases
   | dedup id
   | rename eai:acl.* AS acl_*, "*'*'*" AS "***", "*'*" AS "**"
   | foreach alias.*
      [| eval conf_specific_properties_mv = if( len('<<FIELD>>') > 0, mvappend(conf_specific_properties_mv,
"<<FIELD>>::::" . '<<FIELD>>'), conf_specific_properties_mv )]
   | rex field=value "\s[Aa][Ss]\s(?<field_name>.+)$"
   | table dataSource, title, field_name, stanza, type, attribute, acl_app, acl_owner, acl_sharing, value,
conf_specific_properties_mv
   | eval dataSource = "props-fieldaliases"]
```

Full Utility Version of this Example in Utility Addendum:

Field Object Derivations - View Field Objects

Addendum Level 3

Add copies of result data with adjusted timestamps for testing, data generation, event generation

Use appendpipe to duplicate the data at time it's called, adjusting the timestamps as desired.
Easy way to build "test" data off of live data by duplicating it and modifying the fields as needed.

In this case, we're modifying the time fields by a static amount and adjusting the count fields by a multiplier, effectively creating test data from the single starting row.

The *appendpipe_marker* field here helps illustrate how each appendpipe duplicates the data that existed when appendpipe was called.

```
index=_audit sourcetype=audittrail
| stats latest(_time) AS _time, latest(search_startup_time) AS search_startup_time, latest(event_count)
as event_count, latest(scan_count) as scan_count
BY savedsearch_name
| eval appendpipe_marker = 0
| appendpipe
    [| foreach _time, *_time, *_Time
        [| eval <<FIELD>> = '<<FIELD>>' + 88762]
    | foreach *_count
        [| eval <<FIELD>> = round('<<FIELD>>' * 1.7, 0)]
    | eval appendpipe_marker = "1"]
| appendpipe
    [| foreach _time, *_time, *_Time
        [| eval <<FIELD>> = '<<FIELD>>' + 129571]
    | foreach *_count
        [| eval <<FIELD>> = round('<<FIELD>>' * 0.8, 0)]
    | eval appendpipe_marker = "2"]
| appendpipe
    [| foreach _time, *_time, *_Time
        [| eval <<FIELD>> = '<<FIELD>>' + 175125]
    | foreach *_count
        [| eval <<FIELD>> = round('<<FIELD>>' * 0.1, 0)]
    | eval appendpipe_marker = "3"]
```

Addendum Level 3

Add copies of result data with adjusted timestamps for testing, data generation, event generation

Use appendpipe to duplicate the data at time it's called, adjusting the timestamps as desired.
Easy way to build "test" data off of live data by duplicating it and modifying the fields as needed.

In this case, we're modifying the time fields by a static amount and adjusting the count fields by a multiplier, effectively creating test data from the single starting row.

The *appendpipe_marker* field here helps illustrate how each appendpipe duplicates the data that existed when appendpipe was called.

savedsearch_name	_time	search_startup_time	event_count	scan_count	appendpipe_marker
scma_splunk_instance_info	2023-08-01 01:55:00.253	1195	748	24	0
scma_splunk_instance_info	2023-08-02 02:34:22.253	89957	1272	41	1
scma_splunk_instance_info	2023-08-02 13:54:31.253	130766	598	19	2
scma_splunk_instance_info	2023-08-03 14:33:53.253	219528	1018	33	2
scma_splunk_instance_info	2023-08-03 02:33:45.253	176320	75	2	3
scma_splunk_instance_info	2023-08-04 03:13:07.253	265082	127	4	3
scma_splunk_instance_info	2023-08-04 14:33:16.253	305891	60	2	3
scma_splunk_instance_info	2023-08-05 15:12:38.253	394653	102	3	3

Addendum Level 3

Parse Raw Event CSV field="value" Events Manually

Extracts the Key and Value pairs using regex to display the "summary" of data manually compiled from key="value",key2="value2" CSV-delimited doublequote-wrapped events, like ServiceNow data.

Note: Updated from .Conf version 1.0

```
index=keyvalueCSVDelimevents
| eval hash = md5(_raw)
| eval RawField = replace(_raw, "^[^=]+\s+([^=]+=)", "\1")
| table RawField, hash
| eval _raw = RawField
| foreach RawField
    [| makemv RawField tokenizer=",?([^\"]+\"[^\"]+\")"
| mvexpand RawField
| rex field=RawField "(?<Field>^[^=]+)=\"(?<Value>[^\"]+(\"[^\"]+\"[^\"]+)*)\"$"
| eval Value = if(len(Value) > 50, substr(Value, 0, 50)."...[".len(Value)."]", Value)]
| eval Field = ltrim(Field, " ")
| stats values(Field) AS Fields BY hash, Value
| eval mvcount = mvcount(Fields)
```

hash	RawField
535398b28a8fb5862c1c5919102d1ca9	feature="System Check", color="green", due_to_stanza="feature:wlm_system_check", node_type="feature", node_path="splunkd.workload_management.system_check"
37c6996b835ef642cf15f868689be955	feature="Configuration Check", color="green", due_to_stanza="feature:wlm_configuration_check", node_type="feature", node_path="splunkd.workload_management.configuration_check"

hash	Fields	Value	mvcount
37c6996b835ef642cf15f868689be955	feature	Configuration Check	1
37c6996b835ef642cf15f868689be955	node_type	feature	1
37c6996b835ef642cf15f868689be955	due_to_stanza	feature:wlm_configuration_check	1
37c6996b835ef642cf15f868689be955	color	green	1
37c6996b835ef642cf15f868689be955	node_path	splunkd.workload_management.configuration_check	1
535398b28a8fb5862c1c5919102d1ca9	feature	System Check	1
535398b28a8fb5862c1c5919102d1ca9	node_type	feature	1
535398b28a8fb5862c1c5919102d1ca9	due_to_stanza	feature:wlm_system_check	1
535398b28a8fb5862c1c5919102d1ca9	color	green	1
535398b28a8fb5862c1c5919102d1ca9	node_path	splunkd.workload_management.system_check	1

Addendum Level 3

Numeric Field Value Increase/Decrease Over Rows BY Group

Method to view how the values of a field change over time - noting whether the given field increased or decreased.

In this case, pass a single SID in the initial filter to identify whether the introspection PerProcess metrics increased or decreased over time.

This will calculate that positive/negative/equal change across however many fields there are.

```
index=_introspection
sourcetype=splunk_resource_usage component=PerProcess NOT source IN ("*/splunkforwarder/*", "*\SplunkUniversalForwarder\*")
data.process_type IN (search, search_launcher)
"SID"
| fields - "date_*", linecount, punct, "splunk_server*", timestartpos, timeendpos, timestamp
| rename data.* AS data_*
| rename data_search_props.* AS data_search_props_*
| sort 0 data_search_props_sid, host, -_time
| streamstats reset_on_change=t current=f last(*) AS *_next BY data_search_props_sid, host
| foreach *_next
    [| eval <<MATCHSTR>>_change = case(('<<FIELD>>' - '<<MATCHSTR>>') > 0, "positive",
                                       ('<<FIELD>>' - '<<MATCHSTR>>') < 0, "negative",
                                       ('<<FIELD>>' - '<<MATCHSTR>>') = 0, "equal")

    ]
| table _time, host, *
```

Addendum Level 3

Numeric Field Value Increase/Decrease Over Rows BY Group

Method to view how the values of a field change over time - noting whether the given field increased or decreased.

In this case, pass a single SID in the initial filter to identify whether the introspection PerProcess metrics increased or decreased over time.

This will calculate that positive/negative/equal change across however many fields there are.

data_elapsed	data_elapsed_change	data_elapsed_next	data_fd_used	data_fd_used_change	data_fd_used_next	data_mem_used	data_mem_used_change	data_mem_used_next
1.2300	positive	11.2600	22	positive	29	70.254	positive	88.648
11.2600	positive	21.2600	29	equal	29	88.648	equal	88.648
21.2600	positive	31.2600	29	equal	29	88.648	equal	88.648
31.2600	positive	41.2700	29	equal	29	88.648	equal	88.648
41.2700	positive	51.5400	29	equal	29	88.648	equal	88.648
51.5400	positive	61.5700	29	equal	29	88.648	equal	88.648
61.5700	positive	71.5700	29	equal	29	88.648	equal	88.648

Examples
Level 4

Generates epoch times for 7 days **before/after** beginning of month across the last 3 months.

- Uses <<MATCHSTR>> to reference the base_* fields
- Helpful for running historical searches, correlation boundaries

Generate Start/End Timestamps Using <<MATCHSTR>>

```
| makeresults
| eval base_Month1 = relative_time(now(), "@mon")
| eval base_Month2 = relative_time(now(), "-1mon@mon")
| eval base_Month3 = relative_time(now(), "-2mon@mon")
| foreach base_*
   [| eval <<FIELD>> = round('<<FIELD>>', 0)
   | eval <<MATCHSTR>>_earliest = round(<<FIELD>> - 604800, 0)
   | eval <<MATCHSTR>>_latest = round(<<FIELD>> + 604800, 0)]
```

base_Month1	base_Month2	base_Month3
1685577600	1682899200	1680307200
2023-06-01	2023-05-01	2023-04-01

Month1_earliest	Month1_latest	Month2_earliest	Month2_latest	Month3_earliest	Month3_latest
1684972800	1686182400	1682294400	1683504000	1679702400	1680912000
2023-05-25	2023-06-08	2023-04-24	2023-05-08	2023-03-25	2023-04-08

Examples
Level 4

Establish avg() and p90() BY group-by, then compare each field to those thresholds

Using Dynamically Generated Dataset Thresholds
Comparing field values to the p90(), avg() of the group-by

```
| makeresults count=50
| eval num1 = (random() % 100) + 1, num2 = (random() % 100) + 1,
       num3 = (random() % 100) + 1, num4 = (random() % 100) + 1
| streamstats reset_after="(group > 1)" count AS group
| eventstats avg(*) AS *_avg, p90(*) AS *_p90 BY group
| foreach *
   [| eval less_than_avg=if('<<FIELD>>' < '<<FIELD>>_avg',
              mvappend(less_than_avg, "<<FIELD>>"), less_than_avg)
    | eval less_than_p90=if('<<FIELD>>' < '<<FIELD>>_p90',
              mvappend(less_than_p90, "<<FIELD>>"), less_than_p90)]
```

group	less_than_avg	less_than_p90	num1	num1_avg	num1_p90	num2	num2_avg	num2_p90
1	num2 num4	num1 num2 num4	89.0	54.1	89.6	4.0	46.9	77.6
2	num3	num2 num3 num4	99.0	55.0	85.2	81.0	54.2	86.2
1	num1	num1 num2 num4	34.0	54.1	89.6	48.0	46.9	77.6
2	num1 num4	num1 num2 num3 num4	36.0	55.0	85.2	82.0	54.2	86.2

zScore Calculation

(for anomaly detection)

PLA1159C - Lesser Known Search Commands Part 1

Courtesy of Kyle Smith


```
index=_internal sourcetype=splunkd component=Metrics group=per_sourcetype_thruput
| eval series = replace(series, ":", "_")
| timechart span=60m avg(kbps) AS avg_kbps BY series useother=f
| streamstats window=720 mean(*) AS MEAN*, stdev(*) AS STDEV*
| foreach *
   [| eval Z_<<FIELD>> = (( '<<FIELD>>' - 'MEAN<<MATCHSTR>>') / 'STDEV<<MATCHSTR>>')]
| fields _time, Z*
| eval upper=3, lower=-3
```

This **foreach** command analyzes sourcetype throughput, to detect for anomalies. A **timechart** with the **avg_kbps** of each sourcetype leads into a **streamstats** to get the mean and standard deviation for the rolling 720 window.

The **foreach** command iterates over any of the found fields (sourcetypes), and calculates the Z-Score.

splunk> .conf23

Addendum Level 4

{field} Token

(also Utility)

REST - Splunk instrumentation Health Monitor - {field} Example

Query utilizing foreach and {field} referencing with transpose for viewing health messages.
- Included in Level 4 Addendum as it's an advanced method for dynamic references
- This query is a standalone Utility for viewing health messages

{field} References insert the value of the field within the curly braces, creating a new column for every unique value in the field.

```
| rest services/server/health/splunkd/details splunk_server=local
| fields features.*, health
| rename health AS features.health
| fields - *.reasons.* *.messages.*
| foreach features.*
    [| eval newname = "splunkd." . replace(lower("<<MATCHSTR>>"), " ", "_")
| rex field=newname mode=sed "s/features\.|\.health//g"
| eval {newname} = '<<FIELD>>'
| eval newname_vals = mvappend(newname_vals, "newname=" . newname)
| eval matchstr_vals = mvappend(matchstr_vals, "MATCHSTR=\"<<MATCHSTR>>\"")]
| fields - features.*, newname

| transpose column_name="features"
| rename "row 1" as current_color
```

features	current_color
splunkd	red
splunkd.file_monitor_input	green
splunkd.file_monitor_input.batchreader-0	green
splunkd.file_monitor_input.ingestion_latency	green
splunkd.file_monitor_input.tailreader-0	green
splunkd.index_processor	green
splunkd.index_processor.buckets	green
splunkd.index_processor.disk_space	green
splunkd.index_processor.index_optimization	green
splunkd.resource_usage	red
splunkd.resource_usage.iowait	red

Addendum Level 4

Advanced - Foreach to Clean Fields of Special Characters, Handle Preparation of LDAP Records for Summary Index

Pattern for deep cleaning of field values using step-by-step regex replacements to normalize the output data in a clean format.

Note: Change fields in table command to match LDAP attribute field names
- *flag_IsValidJSON* - field to validate output is proper JSON format.

```
~ Compilation - Compile LDAP Field Properties into JSON
| table fullname, lastchanged, manager, memberof, name, result, sid, telephonenumber, title, user, userprincipalname, cn, domain, company, createdon, department,
departmentnumber, description, division, emailaddress, employeetype
| foreach *
   [| eval <<FIELD>> = if( mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', ":::::"), '<<FIELD>>' )
| eval comment = if(true(), null(), "
1. Fix all field values to fit JSON Schema. ORDER OF OPERATION MATTERS.
   1a. Handle any escape characters. '\\' becomes '\' due to KV_mode=json behavior in Splunk v9.0.2, which leads to invalid JSON schema.
   1b. Add two backslashes to escape non-ascii characters if they do not already have a backslash in front of them: (?<!\\\\) - must be done to pass UTF-16 in valid JSON.
   1c. Escape any doublequotes within each field value.
   1d. Replace any newlines, carriage returns, or tabs with regex pattern. These are read as their underlying representation when KV_mode=json in Splunk v9.0.2, which
creates invalid JSON if we pass the values without correcting it.
Note: The number of backslashes is what worked in 9.0.2, no guarantees out into the future.
2. Roll up all the field values into a JSON object using mvappend. The explicit newline included in eval userproperties mvappend IS INTENDED and included in the
doublequote value, do not remove.")
| foreach *
   [| eval <<FIELD>> = if( match('<<FIELD>>', "\\\\"), replace('<<FIELD>>', "\\\\", "\\\\\\"), '<<FIELD>>' )
   | eval <<FIELD>> = if( match('<<FIELD>>', "(?<!\\\\)[^[:ascii:]]"), replace('<<FIELD>>', "(?<!\\\\)([^[:ascii:]]+)", "\\\\\\\\\1"), '<<FIELD>>' )
   | eval <<FIELD>> = if( match('<<FIELD>>', "(?<!\\\\)\""), replace('<<FIELD>>', "(?<!\\\\)\"", "\\\""), '<<FIELD>>' )
   | eval <<FIELD>> = if( match('<<FIELD>>', "\r"), replace('<<FIELD>>', "\r", "\r"), '<<FIELD>>' )
   | eval <<FIELD>> = if( match('<<FIELD>>', "\n"), replace('<<FIELD>>', "\n", "\n"), '<<FIELD>>' )
   | eval <<FIELD>> = if( match('<<FIELD>>', "\t"), replace('<<FIELD>>', "\t", "\t"), '<<FIELD>>' )
   ]
| foreach *
   [| eval userproperties = if( len('<<FIELD>>') < 1, userproperties, mvappend(userproperties, "  \"<<FIELD>>\":\"" . '<<FIELD>>' . "\"") )]
| eval userproperties = mvappend("{", mvjoin(userproperties, ",
"), "}")
| nomv userproperties
| table domain, user, sid, userproperties
| eval flag_IsValidJSON = if( json_valid(userproperties), null(), "ERROR - userproperties is not valid JSON!")
```

Addendum Level 4

Advanced - Using Foreach to Perform Masking of Data

Uses a series of regex patterns and sequences to create masked values that maintain cardinality of original data without being reversible through the *random()* function and *mvindex()*

- Generates random masked value for each input value
- Passes first generated value for each key/value pair into rest of dataset to maintain cardinality
- Outputs Exception information if length of original/masked values is not consistent.

```
| table key, value
| eval comment = if(true(), null(), "
For universal masking and segment-based masking, iterate through values to replace groups of characters. This avoids passing sensitive data and retains basic semblance to original
values as far as letters/number distribution.
   * Technique uses random() so that the letter selected for replacements varies with each run, while retaining the original distribution.
   * Each row will replace with a randomly selected character from the list, which prevents the data from being reconstructed and constitutes full masking.
   NOTE: Since multiple unique replaced values will appear for each input, to maintain cardinality distribution we pass the first value down to the rest of the rows in streamstats.")
| eval fieldsToMask_values = value
| foreach fieldsToMask_values
   [| eval <<FIELD>> = replace('<<FIELD>>', "[a-d]", mvindex(split("a,b,c,d", ","), (random() % 4)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[e-j]", mvindex(split("e,f,g,h,i,j", ","), (random() % 6)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[k-o]", mvindex(split("k,l,m,n,o", ","), (random() % 5)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[p-t]", mvindex(split("p,q,r,s,t", ","), (random() % 5)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[u-z]", mvindex(split("u,v,w,x,y,z", ","), (random() % 6)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[A-D]", mvindex(split("A,B,C,D", ","), (random() % 4)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[E-J]", mvindex(split("E,F,G,H,I,J", ","), (random() % 6)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[K-O]", mvindex(split("K,L,M,N,O", ","), (random() % 5)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[P-T]", mvindex(split("P,Q,R,S,T", ","), (random() % 5)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[U-Z]", mvindex(split("U,V,W,X,Y,Z", ","), (random() % 6)))
   | eval <<FIELD>> = replace('<<FIELD>>', "[5-9]", tostring(random() % 4 + 1))
   | eval <<FIELD>> = replace('<<FIELD>>', "[0-4]", tostring(random() % 5 + 5))
   ]
| eval comment = if(true(), null(), "
To retain cardinality distribution from input data in the output data, we use streamstats to pass the first randomly generated output value for each unique input value to ensure
overall distinct counts remain consistent.
   * Streamstats avoids limits built into eventstats. Uses sort to reduce memory usage for tracking groups as it runs.")
| sort 0 value
| streamstats first(fieldsToMask_values) AS fieldsToMask_values_masked BY key, value
| eval comment = if(true(), null(), "
Remove fields line for visibility into behavior.")
| fields - fieldsToMask_values
| eval value_masked = coalesce( fieldsToMask_values_masked, anyOtherOptions, value )
| eval Exception = case( len(value_masked)!=len(value) OR (isnull(value_masked) AND isnotnull(value)),
                        "ERROR - Difference between masked value and original value, add following SPL to investigate: | where isnotnull(Exception) ")
```

Addendum Utility References

Reports meant to make your life a bit easier.

Full queries rather than snippets, paste and run.

Utility

Generate SPL Using SPL (1)

Generating SPL is as straightforward as formatting, then concatenating fields however you want the output SPL to return. These are a collection of methods showing some examples of things you can do.

Create a | where match() or | search key="value" condition dynamically using newline-delimited input.

Check whether *ip_address="value"* OR whether the value matches the beginning of the *ip_port* field.

```
| makeresults | fields - _time
| eval input = "192.168.0.1
10.10.10.10
255.255.255.0"
| makemv input tokenizer="([^\n]+)"
| mvexpand input
| eval input_regex = "\"^" . replace(input, "\.", "\\.") . "\""
| eval match_pattern = "match(ip_port, " . input_regex . ")"
| eval search_pattern = "ip_address=\"" . input . "\""


| stats values(match_pattern) AS match_pattern_spl,
        values(search_pattern) AS search_pattern_spl
| eval match_pattern_spl = "| where " . mvjoin(match_pattern_spl, " OR ")
| eval search_pattern_spl = "| search " . mvjoin(search_pattern_spl, " OR ")
```

input	input_regex	match_pattern	search_pattern
192.168.0.1	"^192\.168\.0\.1"	match(ip_port, "^192\.168\.0\.1")	ip_address="192.168.0.1"
10.10.10.10	"^10\.10\.10\.10"	match(ip_port, "^10\.10\.10\.10")	ip_address="10.10.10.10"
255.255.255.0	"^255\.255\.255\.0"	match(ip_port, "^255\.255\.255\.0")	ip_address="255.255.255.0"

match_pattern_spl	search_pattern_spl
| where match(ip_port, "^10\.10\.10\.10") OR match(ip_port, "^192\.168\.0\.1") OR match(ip_port, "^255\.255\.255\.0")	| search ip_address="10.10.10.10" OR ip_address="192.168.0.1" OR ip_address="255.255.255.0"

splunk> .conf23

Utility

Generate SPL Using SPL (2)

Using "::" delimited field-value pairs as input, create eval lines for those values.

Use mvindex to split the string inside the logic, concatenating the segments to create the final | eval line:

```
| makeresults | fields - _time
| eval string = "field1::value1
field2::value2"
| makemv string tokenizer="([^\n]+)"
| mvexpand string
| eval SPL = "| eval " . mvindex(split(string, "::"), 0) . " = \"" . mvindex(split(string, "::"), 1) . "\""
```

string	SPL
field1::value1	| eval field1 = "value1"
field2::value2	| eval field2 = "value2"

Then use stats to aggregate to make copying it easier:

```
...
| stats list(SPL) AS SPL
```

SPL
| eval field1 = "value1" | eval field2 = "value2"

Utility

Generate SPL Using SPL (3)

Create a `TERM()` filter condition for more efficiently searching raw events for IPv4 strings.

Highly recommended sessions for how to use TERM():
PLA1258C (Conf 2023), PLA1089C (Conf 2020)

If you're working with a multivalue field, concatenate with mvjoin:

```
...
| stats values(ip_address) AS list_of_ips_mv
| eval SPL = "( TERM(" . mvjoin(list_of_ips_mv, ") OR TERM(") . ") )"
```

list_of_ips_mv	SPL
10.10.10.10	(TERM(10.10.10.10) OR
192.168.0.1	TERM(192.168.0.1) OR
192.168.1.1	TERM(192.168.1.1) OR
255.255.255.0	TERM(255.255.255.0))

OR

If you're working with a table of single-value fields, you can format then aggregate:

```
...
| table ip_address
| eval ip_formatted = "TERM(" . ip_address . ")"
| stats values(ip_formatted) AS SPL
| eval SPL = "( " . mvjoin(SPL, " OR ") . " )"
```

ip_address	ip_formatted
192.168.0.1	TERM(192.168.0.1)
192.168.1.1	TERM(192.168.1.1)
10.10.10.10	TERM(10.10.10.10)
255.255.255.0	TERM(255.255.255.0)

SPL
(TERM(10.10.10.10) OR TERM(192.168.0.1) OR TERM(192.168.1.1) OR TERM(255.255.255.0))

Utility

Generate SPL Using SPL (4)

Create a `search IN ()` condition dynamically within a subsearch to filter an outer search.

In this case, generate a list of search IDs (SIDs) from *_internal* to pass as a filter to *_introspection*.
This enables seeing introspection *splunk_resource_usage* data for specific savedsearch names, like the Network Traffic DMA.

```
index=_introspection sourcetype=splunk_resource_usage component=PerProcess
data.search_props.role="head" AND data.search_props.sid::*
  [ search index=_internal sourcetype=scheduler status=* sid=* savedsearch_name="_ACCELERATE_DM_Splunk_SA_CIM_Network_Traffic_ACCELERATE_"
    | eval sid = "TERM(" . sid . ")"
    | stats values(sid) AS sid
    | eval search = "data.search_props.sid IN (" . mvjoin(sid, ", ") . ")"
    | table search]
| stats max(data.pct_cpu) AS max_cpu, max(data.mem_used) AS max_mem by sourcetype, data.search_props.sid
```

Here, the inner search is returning a filter string of the SID values wrapped in TERM() to the outer introspection search, allowing calculation.

✓ 11 events (7/31/23 8:58:00.000 PM to 7/31/23 9:58:45.000 PM) No Event Sampling

Events Patterns Statistics (1) Visualization

10 Per Page Format No Preview

search

```
data.search_props.sid IN (TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690837260_1765),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690837560_1795),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690837860_1829),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690838160_1860),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690838460_1902),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690838760_1935),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839060_1973),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839360_2006),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839660_2043),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839960_2086),
TERM(scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690840260_2110))
```

Events Patterns Statistics (11) Visualization

10 Per Page Format No Preview

sourcetype	data.search_props.sid	max_cpu	max_mem
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690837260_1765	98.50	342.680
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690837560_1795		248.918
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690837860_1829		141.730
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690838160_1860	110.59	344.094
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690838460_1902	96.51	249.375
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690838760_1935	119.55	331.375
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839060_1973	106.79	340.105
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839360_2006	103.92	250.340
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839660_2043	110.12	343.059
splunk_resource_usage	scheduler__nobody_U3BsdW5rX1NBX0NJTQ__RMD5dbc859cde34b3bc6_at_1690839960_2086	105.83	290.508

Utility

Check Datamodel Fields for Null Distribution

Run search against datamodel, creating sum count for each field if the value is "unknown" or null to identify DMs/fields that might need investigation and remediation - breakout by *sourcetype*.

Note: This can be an expensive query. Run during off-hours.

```
| datamodel Authentication Authentication flat
| eval dataSource = mvindex(split(sourcetype, ":"), 0)
| eval category = split(category, ";")
| foreach *
   [| eval <<FIELD>>_populatedSum = case(match('<<FIELD>>', "^unknown$") OR isnull('<<FIELD>>'), 0, isnotnull('<<FIELD>>'), 1)]
| fields - dataSource_populatedSum
| stats count as TotalEvents, sum(*_populatedSum) AS * BY dataSource
| transpose 0 column_name="FieldCount" header_field="dataSource"
```

FieldCount	linux_secure	syslog
TotalEvents	20	23
action	15	18
app	20	23
dest	20	23
dest_bunit	0	0
dest_category	0	0
dest_priority	0	0
host	20	23
is_Default_Authentication	20	23
is_Failed_Authentication	20	23

Utility

Check if two sets of search results are the same

row-by-row

1) First, add this SPL to your "A" results to generate a hashed value representing all the fields in that set of results

```
| foreach * [| eval <<FIELD>> = if(len('<<FIELD>>') < 1, null(), '<<FIELD>>') ]
| foreach * [| eval <<FIELD>> = if(mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', "::::"), '<<FIELD>>') ]
| foreach *
   [| eval all_fields_hash = if( isnotnull('<<FIELD>>'), mvjoin(mvappend(all_fields_hash, '<<FIELD>>'),
":::"), all_fields_hash)]
| eval all_fields_hash = sha1(all_fields_hash)
```

2) Get the SID of the first job, then run the same compilation on the "B" second set of results and filter out the set of hash values from "A". Paste the SID from step 1 into the | loadjob command

```
| foreach * [| eval <<FIELD>> = if(len('<<FIELD>>') < 1, null(), '<<FIELD>>') ]
| foreach * [| eval <<FIELD>> = if(mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', "::::"), '<<FIELD>>') ]
| foreach *
   [| eval all_fields_hash = if( isnotnull('<<FIELD>>'), mvjoin(mvappend(all_fields_hash, '<<FIELD>>'),
":::"), all_fields_hash)]
| eval all_fields_hash = sha1(all_fields_hash)

| search NOT all_fields_hash IN
   [| loadjob 1685208748.53268
   | stats values(all_fields_hash) AS search
   | eval search = "(" . mvjoin(search, ", ") . ")"]
```

splunk> .conf23

Utility

Note: Updated from .Conf version 1.0 to move to Utility section.

REST - Compile Search Job Messages BY Search ID

Compiles MV fields of job information messages from current search artifacts in the dispatch directory.

```
| rest splunk_server=local /services/search/jobs
| rex field=id "/jobs/(?<sid>.*)"
| foreach messages.*
    [| eval all_messages_mv=mvdedup(mvappend(messages, '<<FIELD>>'))]
| table sid, all_messages_mv, messages.*
| where mvcount(all_messages_mv) > 1
```

sid	all_messages_mv	messages.error	messages.fatal	messages.info
scheduler__admin_U0EtRW5kcG9pbnRQcm90ZWN0aW9u__RMD5147a7530bec00016_at_1690855800_3724	Unable to find tag virtual Found no results to append to collection 'useraccounts_tracker'.			
scheduler__admin_U0EtRW5kcG9pbnRQcm90ZWN0aW9u__RMD511358f153a5e2956_at_1690852500_3353	Unable to find tag malware Unable to find tag operations			No results. Retaining existing lookup file 'malware_operations_tracker'. The specified search will not match any events Successfully read lookup file '/opt/splunk/etc/apps/SA-EndpointProtection/lookups/malware_operations_tracker.cs
scheduler__admin_U0EtRW5kcG9pbnRQcm90ZWN0aW9u__RMD5147a7530bec00016_at_1690852200_3316	Unable to find tag virtual Found no results to append to collection 'useraccounts_tracker'.			
scheduler__admin_U0EtRW5kcG9pbnRQcm90ZWN0aW9u__RMD511358f153a5e2956_at_1690838100_1842	Unable to find tag malware Unable to find tag operations			No results. Retaining existing lookup file 'malware_operations_tracker'. The specified search will not match any events Successfully read lookup file '/opt/splunk/etc/apps/SA-EndpointProtection/lookups/malware_operations_tracker.cs

Utility

{field} Token

REST - Splunk instrumentation Health Monitor - {field} Example

Query utilizing foreach and {field} referencing with transpose for viewing health messages.
- Included in Level 4 Addendum as it's an advanced method for dynamic references
- This query is a standalone Utility for viewing health messages

{field} References insert the value of the field within the curly braces, creating a new column for every unique value in the field.

```
| rest services/server/health/splunkd/details splunk_server=local
| fields features.*, health
| rename health AS features.health
| fields - *.reasons.* *.messages.*
| foreach features.*
    [| eval newname = "splunkd." . replace(lower("<<MATCHSTR>>"), " ", "_")
| rex field=newname mode=sed "s/features\.|\.health//g"
| eval {newname} = '<<FIELD>>'
| eval newname_vals = mvappend(newname_vals, "newname=" . newname)
| eval matchstr_vals = mvappend(matchstr_vals, "MATCHSTR=\"<<MATCHSTR>>\"")]
| fields - features.*, newname

| transpose column_name="features"
| rename "row 1" as current_color
```

splunk> .conf23

Utility

{field} Token

REST - Splunk instrumentation Health Monitor - {field} Example

Query utilizing foreach and {field} referencing with transpose for viewing health messages.
- Included in Level 4 Addendum as it's an advanced method for dynamic references
- This query is a standalone Utility for viewing health messages

{field} References insert the value of the field within the curly braces, creating a new column for every unique value in the field.
Add the transpose lines to make readout easier.

matchstr_vals	newname_vals	splunkd	splunkd.file_monitor_input	splunkd.file_monitor_input.batchreader-0	splunkd.file_monitor
MATCHSTR="File Monitor Input.features.BatchReader-0.health"	newname=splunkd.file_monitor_input.batchreader-0 newname=splunkd.file_monitor_input.ingestion_latency newname=splunkd.file_monitor_input.tailreader-0 newname=splunkd.file_monitor_input newname=splunkd.index_processor.buckets newname=splunkd.index_processor.disk_space newname=splunkd.index_processor.index_optimization newname=splunkd.index_processor newname=splunkd.resource_usage.iowait newname=splunkd.resource_usage.iowait.messages newname=splunkd.resource_usage newname=splunkd.search_scheduler.search_lag newname=splunkd.search_scheduler.searches_delayed newname=splunkd.search_scheduler.searches_skipped newname=splunkd.search_scheduler newname=splunkd.workload_management.disabled newname=splunkd.workload_management newname=splunkd	red	green	green	green
MATCHSTR="File Monitor Input.features.Ingestion Latency.health"					
MATCHSTR="File Monitor Input.features.TailReader-0.health"					
MATCHSTR="File Monitor Input.health"					
MATCHSTR="Index Processor.features.Buckets.health"					
MATCHSTR="Index Processor.features.Disk Space.health"					
MATCHSTR="Index Processor.features.Index Optimization.health"					

features	current_color
splunkd	red
splunkd.file_monitor_input	green
splunkd.file_monitor_input.batchreader-0	green
splunkd.file_monitor_input.ingestion_latency	green
splunkd.file_monitor_input.tailreader-0	green
splunkd.index_processor	green
splunkd.index_processor.buckets	green
splunkd.index_processor.disk_space	green
splunkd.index_processor.index_optimization	green
splunkd.resource_usage	red
splunkd.resource_usage.iowait	red

Utility

REST - KVstore Collections Config Info

Displays KVstore collection configuration information from REST.
Note: related to collections.conf, does not pick up transforms.conf settings.

- Compiles field type, field acceleration, profiling, and replication settings into unique columns.
- May want to set Wrap Results to "false" in table view.

```
| rest /servicesNS/-/-/storage/collections/config splunk_server=*
| rename eai:acl.app AS appContext
| fields - id, eai:*, author, published, type, updated
| eval comment = if( true(), null(), "
Universal multivalue field handling. Ensure our other compilation logic doesn't break on multivalued fields. Simpler than every foreach having multivalue condition handlers.")
| foreach *
    [| eval <<FIELD>> = if( mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', ":::"), '<<FIELD>>')]
| eval comment = if( true(), null(), "
Compile kvstore collection properties together.")
| foreach accelerated_fields*
    [| eval props_acceleratedField = if( isnull('<<FIELD>>') OR len('<<FIELD>>') < 1, props_acceleratedField, mvappend(props_acceleratedField, "<<FIELD>> = " . '<<FIELD>>') )]
| foreach field.*
    [| eval props_fieldType = if( isnull('<<FIELD>>') OR len('<<FIELD>>') < 1, props_fieldType, mvappend(props_fieldType, "<<FIELD>> = " . '<<FIELD>>') )]
| foreach replicat*
    [| eval props_replication = if( isnull('<<FIELD>>') OR len('<<FIELD>>') < 1, props_replication, mvappend(props_replication, "<<FIELD>> = " . '<<FIELD>>') )]
| foreach profiling*
    [| eval props_profiling = if( isnull('<<FIELD>>') OR len('<<FIELD>>') < 1, props_profiling, mvappend(props_profiling, "<<FIELD>> = " . '<<FIELD>>') )]
| fields - accelerated_fields*, field.*, profiling*, replicat*
| table appContext, title, disabled, props_fieldType, enforceTypes, props_acceleratedField, props_profiling, props_replication, *, splunk_server
| rename title AS collection, props_* AS *Properties
```

Note: Updated from .Conf version 1.0 to address SPL syntax issues and enrich output.

splunk> .conf23

Utility

REST - KVstore Collections Config Info

Displays KVstore collection configuration information from REST.
Note: related to collections.conf, does not pick up transforms.conf settings.

- Compiles field type, field acceleration, profiling, and replication settings into unique columns.
- May want to set Wrap Results to "false" in table view.

appContext	collection	disabled	fieldTypeProperties	enforceTypes	acceleratedFieldProperties
Splunk_SA_CIM	cam_queue	0	field.action_name = string field.info = string field.settings = string field.sid = string field.time = time field.worker = string		accelerated_fields.default = {"worker": 1} accelerated_fields.sid = {"sid": 1}
splunk_secure_gateway	subscriptions	0	field.device_id = string field.expired_time = string field.last_update_time = string field.session_key = string field.session_key_type = string field.shard_id = string field.subscription_key = string field.subscription_type = string field.ttl_seconds = string field.user = string field.version = number field.visualization_id = string		accelerated_fields.device_id_accel = {"device_id": 1} accelerated_fields.last_update_time_accel = {"last_update_time": 1} accelerated_fields.subscription_accel = {"subscription_key": 1} accelerated_fields.subscription_type_accel = {"subscription_type": 1}
splunk_secure_gateway	device_role_mapping	0	field.device_id = string field.role = string field.timestamp = number		accelerated_fields.device_role_accel = {"role": 1}

Utility

REST - Serverclass DS Assignment Breakout

Displays consolidated serverclass assignment information for DS instance, showing app deployment settings and whitelist/blacklist settings.

- Uses | `fields` - to drop fields, combined with table wildcard to ensure any new fields appear

```
| rest /services/deployment/server/serverclasses splunk_server=*
| rename eai:acl.app AS appContext
| fields - id, repositoryList*, eai:acl.*, author, published
| eval comment = if( true(), null(), "
Universal multivalue field handling. Ensure our other compilation logic doesn't break on multivalued fields since serverclass syntax has been changing recently.")
| foreach *
    [| eval <<FIELD>> = if( mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', "::::"), '<<FIELD>>')]
| eval comment = if( true(), null(), "
Note: whitelist/blacklist references will be updated to allow/deny in future Splunk version.")
| foreach whitelist.*
    [| eval whitelist_values = if( isnull('<<FIELD>>') OR len('<<FIELD>>') < 1, whitelist_values, mvappend(whitelist_values, "\"<<FIELD>>\" : \"" . '<<FIELD>>' . "\"") )]
| foreach blacklist.*
    [| eval blacklist_values = if( isnull('<<FIELD>>') OR len('<<FIELD>>') < 1, blacklist_values, mvappend(blacklist_values, "\"<<FIELD>>\" : \"" . '<<FIELD>>' . "\"") )]
| fields - whitelist.*, blacklist.*
| eval comment = if( true(), null(), "
Compile other app deployment settings for better readability.")
| foreach restart*, excludeFromUpdate, issueReload, stateOnClient
    [| eval app_properties = if( isnull('<<FIELD>>') OR len('<<FIELD>>') < 1, app_properties, mvappend(app_properties, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\"") )]
| fields - restart*, excludeFromUpdate, issueReload, stateOnClient
| eval comment = if( true(), null(), "
Wildcard at end of table will display any new fields that may appear in newer versions.")
| table appContext, title, loadTime, currentDownloads, whitelist-size, blacklist-size, whitelist_values, blacklist_values, machineTypesFilter, app_properties, *
| rename title AS serverclass_name, updated AS webui_updated
| eval loadTime = strftime(loadTime, "%Y-%m-%d %X %z")
| fields - filterType, continueMatching, endpoint, repositoryLocation, targetRepositoryLocation, tmpFolder
```

Note: Updated from .Conf version 1.0 to address SPL syntax issues and enrich output.

splunk> .conf23

Utility

REST - Serverclass DS Assignment Breakout

Displays consolidated serverclass assignment information for DS instance, showing app deployment settings and whitelist/blacklist settings.

- Uses | `fields` - to drop fields, combined with table wildcard to ensure any new fields appear

appContext	serverclass_name	loadTime	currentDownloads	whitelist-size	blacklist-size	whitelist_values	blacklist_values	machineTypesFilter	app_properties
system	all_uf_inputs_nix	2023-07-31 18:30:27 +0000	0	1	0	"whitelist.0" : "*NIX*"		linux-x86_64	"restartIfNeeded":"0" "restartSplunkWeb":"0" "restartSplunkd":"0" "issueReload":"0" "stateOnClient":"enabled"
system	all_uf_inputs_windows	2023-07-31 18:30:27 +0000	0	2	1	"whitelist.0" : "LAB01*" "whitelist.1" : "12.12.1.12"	"blacklist.0" : "*NIX*"	windows-*	"restartIfNeeded":"0" "restartSplunkWeb":"0" "restartSplunkd":"0" "issueReload":"0" "stateOnClient":"enabled"
system	Linux_Phantom_Inputs	2023-07-31 18:30:26 +0000	0	1	0	"whitelist.0" : "10.10.10.15"			"restartIfNeeded":"0" "restartSplunkWeb":"0" "restartSplunkd":"0" "issueReload":"0" "stateOnClient":"enabled"

Utility

REST - DS Deployment Client App Assignment Breakout

Displays app deployment information for any DS instance the SH is a peer of.
Ideally run this on DS or DMC.

- Compiles application/serverclass into consolidated column *client_app_sc_list* for each *hostname*
- Deduplicates using `mvdedup`
- `mvexpand` then `rex` to output 1 row per unique *hostname* + *serverclass* + *app*

```
| rest /services/deployment/server/clients splunk_server=*
| fields hostname, instanceName, title, dns, ip, splunkVersion, utsname, applications.*.serverclasses, averagePhoneHomeInterval,
lastPhoneHomeTime, build, mgmt, package, updated, splunk_server
| foreach applications.*.serverclasses
    [| eval client_app_sc_list = if( isnotnull('<<FIELD>>'),
                  mvdedup(mvappend(client_app_sc_list, "<<MATCHSEG1>>:::".'<<FIELD>>')), client_app_sc_list )
    ]
| fields - applications.*.serverclasses
| rename title AS client_guid, dns AS host_dns, ip AS ip_address, utsname AS machineType, build AS splunk_build, mgmt AS
mgmt_port, updated AS webui_updated
| table hostname, instanceName, host_dns, ip_address, splunkVersion, machineType, package, client_app_sc_list, lastPhoneHomeTime,
app, serverclass, webui_updated, splunk_server
| mvexpand client_app_sc_list
| rex field=client_app_sc_list "^(?<app>.+):::(?<serverclass>.+)"
| fields - client_app_sc_list
```

Note: Updated from .Conf version 1.0 to address SPL syntax issues and enrich output.

splunk> .conf23

Utility

REST - DeploymentClient App Assignment Breakout

Displays app deployment information for any DS instance the SH is a peer of.
Ideally run this on DS or DMC.

- Compiles application/serverclass into consolidated column *client_app_sc_list* for each *hostname*
- Deduplicates using `mvdedup`
- `mvexpand` then `rex` to output 1 row per unique *hostname* + *serverclass* + *app*

instanceName	host_dns	ip_address	splunkVersion	machineType	package	lastPhoneHomeTime	app	serverclass
splunk02.lab.gps	lab02.spl.gps	10.10.10.10	9.0.2	linux-x86_64	universal_forwarder	1690825060	Lab_TA_DeploymentClient	All_UniversalForwarder_Deploy
splunk02.lab.gps	lab02.spl.gps	10.10.10.10	9.0.2	linux-x86_64	universal_forwarder	1690825060	Lab_TA_Outputs	All_UniversalForwarder_Output
splunk02.lab.gps	lab02.spl.gps	10.10.10.10	9.0.2	linux-x86_64	universal_forwarder	1690825060	Splunk_TA_nix	Linux_OS_Inputs
splunk02.lab.gps	lab02.spl.gps	10.10.10.10	9.0.2	linux-x86_64	universal_forwarder	1690825060	TA-phantom_internal	Linux_Phantom_Inputs
splunk02.lab.gps	lab02.spl.gps	10.10.10.10	9.0.2	linux-x86_64	universal_forwarder	1690825060	uf_meta_fields_tracker	uf_meta_fields_tracker

Utility

REST - Savedsearches Properties Generation

Compiles majority of properties returned from savedsearches.conf API output and displays them in compiled columns.
Additionally uses subsearch *within eval if()* to mark applications as "core", meaning out of box.
Does not consider multiple SHs when deduplicating, check dedup line for logic used.

Note: Updated from .Conf version 1.0 to enrich output.

```
| rest /servicesNS/-/-/saved/searches splunk_server=local count=0
| eval updated_strptime = strptime(update, "%Y-%m-%dT%H:%M:%S%z")
| dedup id, title, eai:acl.app, search sortby -updated_strptime
| rename search AS searchQuery
| eval env = if(match(splunk_server, "splunkcloud\.com$"), "Cloud", "OnPrem")
| fields env, title, description, disabled, searchQuery, is_scheduled, cron_schedule, eai:acl.*, updated, dispatchAs, actions, action.*, alert_*, alert.*, auto_summarize.*, dispatch.*, durable.*, allow_skew,
is_visible, max_concurrent, realtime_schedule, request.*, run_*, schedule_*, author, federated.provider, workload_pool, id, indicator_name, related_dashboard
| foreach *
    [| eval <<FIELD>> = if( len('<<FIELD>>') < 1, null(), '<<FIELD>>' )
    | eval <<FIELD>> = if( mvcount('<<FIELD>>') > 1, mvjoin('<<FIELD>>', "::::"), '<<FIELD>>' )]
| eval app_name = 'eai:acl.app',
    acl_owner = 'eai:acl.owner',
    acl_sharing = 'eai:acl.sharing'
| eval comment = if(true(), null(), "
Compile diverse fields into aggregated '*_props_mv' multivalue fields.")
| foreach eai:acl.*
    [| eval acl_props_mv = if(isnotnull('<<FIELD>>') , mvappend(acl_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), acl_props_mv)]
| foreach action.*
    [| eval action_props_mv = if(isnotnull('<<FIELD>>'), mvappend(action_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), action_props_mv)]
| foreach alert.*
    [| eval alert_props_mv = if(isnotnull('<<FIELD>>'), mvappend(alert_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), alert_props_mv)]
| foreach auto_summarize*
    [| eval auto_summarize_props_mv = if(isnotnull('<<FIELD>>'), mvappend(auto_summarize_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), auto_summarize_props_mv)]
| foreach dispatch.*
    [| eval dispatch_props_mv = if(isnotnull('<<FIELD>>'), mvappend(dispatch_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), dispatch_props_mv)]
| foreach durable.*
    [| eval durable_props_mv = if(isnotnull('<<FIELD>>'), mvappend(durable_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), durable_props_mv)]
| eval comment = if(true(), null(), "
Aggregate all other interesting but not directly useful fields into a consolidated mv column for sake of auditing and integrity.")
| foreach allow_skew, is_visible, max_concurrent, realtime_schedule, request.*, run_*, schedule_*, author, federated.provider, workload_pool, indicator_name, related_dashboard
    [| eval misc_props_mv = if(isnotnull('<<FIELD>>'), mvappend(misc_props_mv, "\"<<FIELD>>\":\"" . '<<FIELD>>' . "\""), misc_props_mv)]
| rename dispatch.earliest_time AS dispatch_earliest_time, dispatch.latest_time AS dispatch_latest_time
| fields - eai:acl.*, action.*, dispatch.*, alert.*, display.*, auto_summarize*, durable.*, allow_skew, is_visible, max_concurrent, realtime_schedule, request.*, run_*, schedule_*, author, federated.provider,
workload_pool, indicator_name, related_dashboard
| rename dispatch_earliest_time AS dispatch.earliest_time, dispatch_latest_time AS dispatch.latest_time
| eval comment = if(true(), null(), "
Subsearch references apps endpoint to allow filtering of core apps. Passes the value back to the in() function to create the flag field.")
| eval flag_CoreApp = if( in(app_name,
                            [| rest splunk_server=local count=0 /servicesNS/-/-/apps/local search="core=1"
                            | stats values(title) AS search
                            | eval search = "\"" . mvjoin(search, "\", \"") . "\""
                            ]),
                        "true", "false" )
| table env, flag_CoreApp, app_name, title, disabled, description, updated, is_scheduled, acl_owner, cron_schedule, acl_sharing, searchQuery, dispatch.*, dispatchAs, alert_type, alert_condition, alert_comparator,
alert_threshold, alert_props_mv, actions, action*, *_mv, *
```

Utility

REST - Savedsearches Properties Generation

Compiles majority of properties returned from savedsearches.conf API output and displays them in compiled columns.
Additionally uses subsearch *within eval if()* to mark applications as "core", meaning out of box.
Does not consider multiple SHs when deduplicating, check dedup line for logic used.

env	flag_CoreApp	app_name	title	disabled	description	updated	is_scheduled	acl_owner	cron_schedule	acl_sharing	searchQuery
OnPrem	true	splunk_monitoring_console	DMC Alert - Abnormal State of Indexer Processor	1	One or more of your indexers is reporting an abnormal state.	1970-01-01T00:00:00+00:00	1	nobody	3,8,13,18,23,28,33,38,43,48,53,58 * * * *	app	| rest splunk_server_group=dmc_group_indexer /services/server/introspection/indexer | fields splunk_server, average_KBps, status, rea | where status != "normal" | eval average_KBps = round(average_KBps, 0) | eval status= if(status=="normal", status, statu ".reason) | fields - reason | rename splunk_server as Instance, average_KBps "Average KB/s (last 30s)", status as Status
OnPrem	true	splunk_monitoring_console	DMC Alert - Critical System Physical Memory Usage	1	One or more instances has exceeded 90% memory usage.	1970-01-01T00:00:00+00:00	1	nobody	3,8,13,18,23,28,33,38,43,48,53,58 * * * *	app	| rest splunk_server_group=dmc_group_* /services/server/status/resource-usage/hostwide | eval percentage=round(mem_used/mem,3)*100 | where percentage > 90 | fields splunk_server, percentage, mem_used, mem | rename splunk_server AS Instance, mem AS "Physi memory installed (MB)", percentage AS "Memory use mem_used AS "Memory used (MB)"
OnPrem	true	splunk_monitoring_console	DMC Alert - Expired and Soon To Expire Licenses	1	You have instances with licenses that	1970-01-01T00:00:00+00:00	1	nobody	3 0 * * *	app	| rest splunk_server_group=dmc_group_license_mast /services/licenser/licenses | join type=outer group_id splunk_server [rest splunk_server_group=dmc_group_license_ma /services/licenser/groups

Utility

Extract $token$ Token References from XML Dashboards

Identifies and extracts all token references (usage), token definitions (input), and search references defined within an XML dashboard. Second step identifies tokens that are referenced but not defined.

Note: Updated from .Conf version 1.0

```
| rest /servicesNS/-/-/data/ui/views splunk_server=local
| search title="*"
| appendpipe
   [| fields id, eai:data
   | rex field=eai:data max_match=0 "(?<tokenReferences>[^\$\>]\$[^\$\"\n]+\$)"
   | fields - eai:data
   | eval tokenReferences = mvfilter(!match(tokenReferences, "^\>"))
   | mvexpand tokenReferences
   | eval tokenReferences = replace(replace(replace(tokenReferences, "^[^\$]", ""), "\|\w\$", "$"), "\$", "")
   | search NOT tokenReferences IN ("row.*", "job.*", "click.*")
   | eval tokenReferences = replace(tokenReferences, "\.(earliest|latest)", "")
   | stats values(tokenReferences) AS tokenReferences BY id]
| stats values(*) AS * BY id
| rex field=eai:data max_match=0 "\<search.+id=\"(?<search_ids>[^\"]+)"
| rex field=eai:data max_match=0 "token=\"(?<tokenDefinitions>[^\"]+)"
| eval tokenDefinitions = mvmap(tokenDefinitions,
              if(match(tokenDefinitions, "^form\."), replace(tokenDefinitions, "^form\.", "") . "(form.)", tokenDefinitions))
| eval tokenReferences = mvmap(tokenReferences,
              if(match(tokenReferences, "^form\."), replace(tokenReferences, "^form\.", "") . "(form.)", tokenReferences))
| foreach tokenReferences, tokenDefinitions, search_ids
   [| eval <<FIELD>> = mvsort(mvdedup('<<FIELD>>'))]
| table title, tokenReferences, tokenDefinitions, search_ids, eai:data

~ Identify Tokens that are referenced but not defined in dashboard XML (Using the Extracted tokens to Identify Missing ones)
| fields - eai:data
| mvexpand tokenReferences
| foreach tokenReferences
    [| eval flag_match = case(isnotnull(flag_match), flag_match, isnum(mvfind(tokenDefinitions, "^".tokenReferences."$")), "match")]
| where isnull(flag_match) AND len(tokenReferences) > 1
| rename tokenReferences AS tokenReferencesWithoutDefinition
| stats values(*) AS * BY title
| table title, tokenReferencesWithoutDefinition, tokenDefinitions, search_ids
```

Utility

Extract $token$ Token References from XML Dashboards

Identifies and extracts all token XML dashboard references

title	tokenReferences	tokenDefinitions	search_ids	eai:data
forwarder_instance	forwarderStatusDrilldown funcVolume hostname snapshot	forwarderStatusDrilldown funcVolume hostname time	confRepPerfSearch	<form version="1.1" hideEdit="True" script="co <label>Forwarders: Instance</label> <fieldset autoRun="true" submitButton="false <input type="dropdown" searchWhenChanged=" <label>Instance:</label> <showClearButton>false</showClearButton> <populatingSearch fieldForLabel="hostnam | `dmc_get_forwarder_info` | stats count by hostname </populatingSearch> <selectFirstChoice>true</selectFirstChoi </input> <input type="time" searchWhenChanged="true <label>Time Range:</label> <default> <earliestTime>-4h@m</earliestTime> <latestTime>now</latestTime> </default> </input> </fieldset> <row id="forwarder_monitoring_extension"> <panel> <html> <h3>Forwarder Monitoring is <b>disable

splunk> .conf23

Utility

Extract $token$ Token References from XML Dashboards

Identifies and extracts all token XML dashboard references

```
~ Identify Tokens that are referenced but not defined in dashboard XML (Using the Extracted tokens to Identify Missing ones)
...
| table title, tokenReferences, tokenDefinitions, search_ids, eai:data
| fields - eai:data
| mvexpand tokenReferences
| foreach tokenReferences
    [| eval flag_match = case(isnotnull(flag_match), flag_match, isnum(mvfind(tokenDefinitions, "^".tokenReferences."$")), "match")]
| where isnull(flag_match) AND len(tokenReferences) > 1
| rename tokenReferences AS tokenReferencesWithoutDefinition
| stats values(*) AS * BY title
| table title, tokenReferencesWithoutDefinition, tokenDefinitions, search_ids
```

title	tokenReferencesWithoutDefinition	tokenDefinitions	search_ids
forwarder_deployment	snapshot	forwarderCountOverlayLabel forwarderCountSplitBy forwarderCountSplitByLabel forwarderNameFilter forwarderStatusFilter forwarder_count_including_internal_logs funcForwarderCountOverlay last_run_time show_instances_forwarding_logs time true_forwarder_count	forwarderReceiverCountBaseSearch forwarderReceiverExcludingInternalCountBaseSearch loadForwarderAssetsSearch loadForwarderExcludingInternalAssetsSearch
forwarder_instance	snapshot	forwarderStatusDrilldown funcVolume hostname time	confRepPerfSearch
http_event_collector_deployment	historical snapshot	activity_chart_label activity_chart_split_by_token all_errors_selected allow_split_by_token authentication_failure_selected	base_historical_search base_snapshot_search base_time_search

.conf23

Utility

Indexer Performance - Response times of Indexer Layer

Filtered to top contributors using weight determined by thresholds dynamically generated (Two Part Query)


```
index=_* sourcetype=splunkd_access sh_sid=*   "*ms*"
   host IN (<indexers>)
| bucket _time span=1m
| stats avg(spent) AS avg_respTimeMS, p50(spent) AS p50_respTimeMS, p66(spent) AS p66_respTimeMS, p75(spent) AS p75_respTimeMS, p90(spent) AS p90_respTimeMS, p95(spent) AS p95_respTimeMS, p99(spent) AS
p99_respTimeMS, max(spent) AS max_respTimeMS BY _time, host
| foreach *_respTimeMS
   [| eval <<FIELD>> = round('<<FIELD>>', 0)]
| join type=outer _time
   [ search index=_* sourcetype=splunkd_access
       "*ms*" sh_sid=*
       host IN (<indexers>)
   | timechart span=1m p40(spent) AS p40_spent
   | eval p40_spent = round(p40_spent, 0)
   | rename p40_spent AS threshold]
| foreach *_respTimeMS
   [| eval fieldsThatPassThreshold = case(
       isnull(fieldsThatPassThreshold) AND '<<FIELD>>' > threshold, "<<FIELD>>",
       isnotnull(flag_ValuesPassingThreshold) AND '<<FIELD>>' > threshold, mvappend(flag_ValuesPassingThreshold, "<<FIELD>>"),
       isnotnull(flag_ValuesPassingThreshold) AND '<<FIELD>>' < threshold, fieldsThatPassThreshold)]
| eval weightBasedOnThresholdViolations = mvcount(fieldsThatPassThreshold)
```

|loadjob of SID above

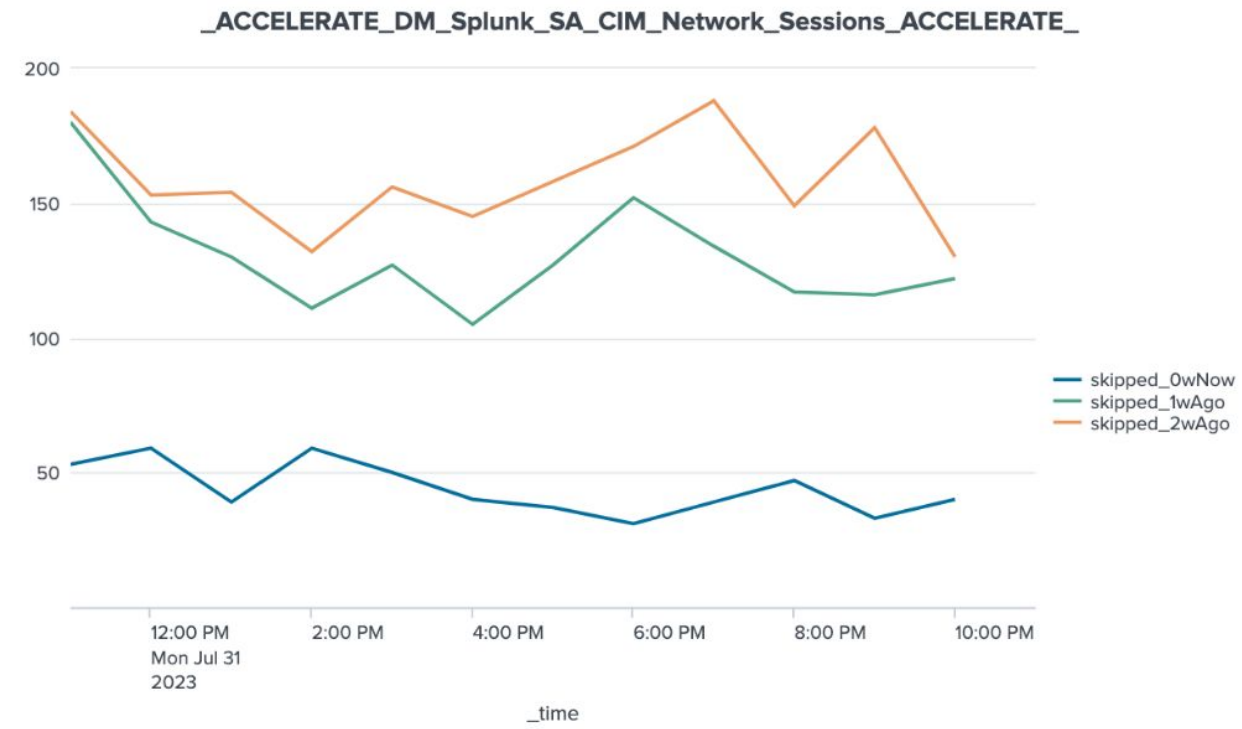
```
| loadjob <sid of above> | search
   [| loadjob <sid of above>
   | stats sum(weightBasedOnThresholdViolations) AS totalWeight BY host
   | sort 0 - totalWeight
   | head 15
   | return 15 host]
| fields - weightBasedOnThresholdViolations, fieldsThatPassThreshold
| rename threshold AS p40_threshold
| timechart span=1m limit=15 sum(avg_respTimeMS) AS avg_respTimeMS, sum(max_respTimeMS) AS max_respTimeMS, sum(p40_threshold) AS p40_threshold, sum(p50_respTimeMS) AS p50_respTimeMS, sum(p66_respTimeMS) AS
p66_respTimeMS, sum(p75_respTimeMS) AS p75_respTimeMS, sum(p90_respTimeMS) AS p90_respTimeMS, sum(p95_respTimeMS) AS p95_respTimeMS, sum(p99_respTimeMS) AS p99_respTimeMS BY host
```

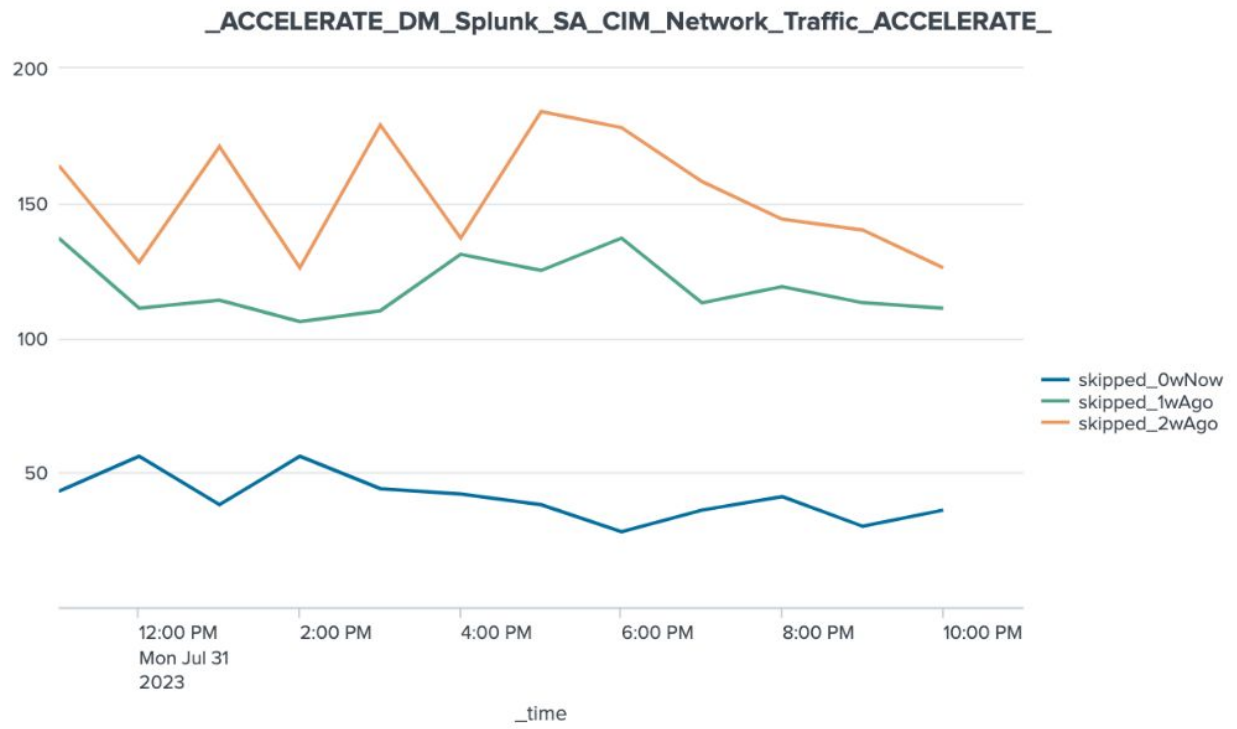
Utility

Search Scheduler Job Breakout Comparing Current Period to Same Period 1 week before and 2 weeks before

This comparison gives us a view into whether the current number of searches, number of results, and number of *host* values is aligned with those same metrics during previous historical periods for a given savedsearch + status.

Below is a comparison view of two Datamodel Acceleration searches over a 12 hour period restricted to *status="skipped"* - this shows us the number of skips per hour compared to the previous weeks during the same hours.





The next slide shows the base Step 1 SPL output that we used to generate these timecharts.

Utility

Search Scheduler Job Breakout Comparing Current Period to Same Period 1 week before and 2 weeks before

(Step 1 Output)

_time	savedsearch_name	status	hist_count_0w	hist_count_1w	hist_count_2w	hist_hosts_0w	hist_hosts_1w	hist_hosts_2w	hist_sid_0w
2023-07-31 19:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Sessions_ACCELERATE_	skipped	23	92	118	1	1	1	0
2023-07-31 19:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Traffic_ACCELERATE_	skipped	21	71	89	1	1	1	0
2023-07-31 20:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Sessions_ACCELERATE_	skipped	47	117	149	1	1	1	0
2023-07-31 20:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Traffic_ACCELERATE_	skipped	41	119	144	1	1	1	0
2023-07-31 21:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Sessions_ACCELERATE_	skipped	33	116	178	1	1	1	0
2023-07-31 21:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Traffic_ACCELERATE_	skipped	30	113	140	1	1	1	0
2023-07-31 22:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Sessions_ACCELERATE_	skipped	40	122	130	1	1	1	0
2023-07-31 22:00	_ACCELERATE_DM_Splunk_SA_CIM_Network_Traffic_ACCELERATE_	skipped	36	111	126	1	1	1	0

Here we can see the current "0w" period seems to have much less skips than previous weeks.

The next slide explains what this utility does and how, then the 2 SPL searches are given.

Utility

Search Scheduler Job Breakout Comparing Current Period to Same Period 1 week before and 2 weeks before

This comparison gives us a view into whether the current number of searches, number of results, and number of *host* values is aligned with those same metrics during previous historical periods for a given savedsearch + status.

We accomplish this by:

- Collecting the scheduler data from the current period of time alongside the "same" range of time from 1 week prior and 2 weeks prior BY savedsearch_name + status.
- Adjusting the timestamp values of the historical data so that it's aligned with the current data in 1 hour buckets, notated by "1w" and "2w" labels which we reference to do the adjustment.
- Comparing the adjusted values for each savedsearch + status to identify differences where, for example, a savedsearch that returned results in both historical periods did not return results in the current period.

This is best run as a two-step search, where in the first step we collect the base data and adjust the timestamps, then in the second step we render it as a visualization to see the changes in behavior.

The second step includes condition logic (*flag_Investigate)* to filter results to savedsearches where:

- the current count is less than both the 1w and 2w counts
- the number of failures/skipped searches is increased in the current counts

This *flag_Investigate* logic can be modified as needed. The last part of the SPL is a verbose timechart output using eval conditions to render specific aspects of the data, allowing us to compare the individual status types across each period we're comparing.

Note: To compare to other time periods, simply change "1w" or "2w" to the desired window everywhere it appears in the SPL queries.

This is the "universal" version which is *a lot* - if you restrict to a specific savedsearch name or a specific status, it becomes much more manageable.

Utility

Search Scheduler Job Breakout Comparing Current Period to Same Period 1 week before and 2 weeks before (Step 1 SPL)


```
| multisearch
    [ search index=_internal sourcetype=scheduler status=* savedsearch_name=*
    | fields _time, host, savedsearch_name, sid, status
    | bucket _time span=1h
    | eval comment = if(true(), null(), "
time_group_offset value should be 0w as this search is compiling the range to compare against historical periods.")
    | eval time_group_offset = "0w"]
    [ search index=_internal sourcetype=scheduler status=* savedsearch_name=*
        [| makeresults | addinfo
        | eval earliest = relative_time(info_min_time, "-1w")
        | eval latest = relative_time(info_max_time, "-1w")
        | eval search = "earliest=" . earliest . " latest=" . latest
        | table search]
    | fields _time, host, savedsearch_name, sid, status
    | bucket _time span=1h
    | eval comment = if(true(), null(), "
time_group_offset value should reflect the relative_time calculation adjustment used above.")
    | eval time_group_offset = "1w"]
    [ search index=_internal sourcetype=scheduler status=* savedsearch_name=*
        [| makeresults | addinfo
        | eval earliest = relative_time(info_min_time, "-2w")
        | eval latest = relative_time(info_max_time, "-2w")
        | eval search = "earliest=" . earliest . " latest=" . latest
        | table search]
    | fields _time, host, savedsearch_name, sid, status
    | bucket _time span=1h
    | eval comment = if(true(), null(), "
time_group_offset value should reflect the relative_time calculation adjustment used above.")
    | eval time_group_offset = "2w"]
| stats count AS agg_count, dc(sid) AS agg_sid, dc(host) AS agg_hosts by _time, savedsearch_name, status, time_group_offset
| eval comment = if(true(), null(), "
NOTE: You MUST round the output of relative_time() else the two _time values will not align due to subsecond addition.")
| eval _time = if( time_group_offset="0w", _time, round(relative_time(_time, "+".time_group_offset), 0) )
| eval comment = if(true(), null(), "
We use multisearch as it avoids the limitations of subsearches inherent with any append-type approach.
Then, we split each time_group_offset groupby into dedicated columns for all aggregated agg_* fields using foreach and a curly-brace reference to time_group_offset, which inserts the value
into the new field names notated as hist_<metric>_<timerange>.")
| foreach agg_*
    [| eval "hist_<<MATCHSTR>>_{time_group_offset}" = '<<FIELD>>']
| fields - agg_*
| stats values(*) AS * BY _time, savedsearch_name, status
```

Utility

Search Scheduler Job Breakout Comparing Current Period to Same Period 1 week before and 2 weeks before

(Step 1 Output)

_time	savedsearch_name	status	hist_count_0w	hist_count_1w	hist_count_2w	hist_hosts_0w	hist_hosts_1w	hist_hosts_2w	hist_sid_0w	hist_sid_1w	hist_sid_2w	time_group
2023-07-31 19:00	Access - Access App Tracker - Lookup Gen	success	2	2	2	1	1	1	2	2	2	0w 1w 2w
2023-07-31 20:00	Access - Access App Tracker - Lookup Gen	success	4	4	4	1	1	1	4	4	4	0w 1w 2w
2023-07-31 21:00	Access - Access App Tracker - Lookup Gen	success	4	4	4	1	1	1	4	4	4	0w 1w 2w
2023-07-31 22:00	Access - Access App Tracker - Lookup Gen	success	4	4	4	1	1	1	4	4	4	0w 1w 2w
2023-07-31 23:00	Access - Access App Tracker - Lookup Gen	success	2	2	2	1	1	1	2	2	2	0w 1w 2w
2023-07-31 20:00	Access - Authentication Tracker - Lookup Gen	success	1	1	1	1	1	1	1	1	1	0w 1w 2w

Utility

Search Scheduler Job Breakout Comparing Current Period to Same Period 1 week before and 2 weeks before

(Step 2 SPL)

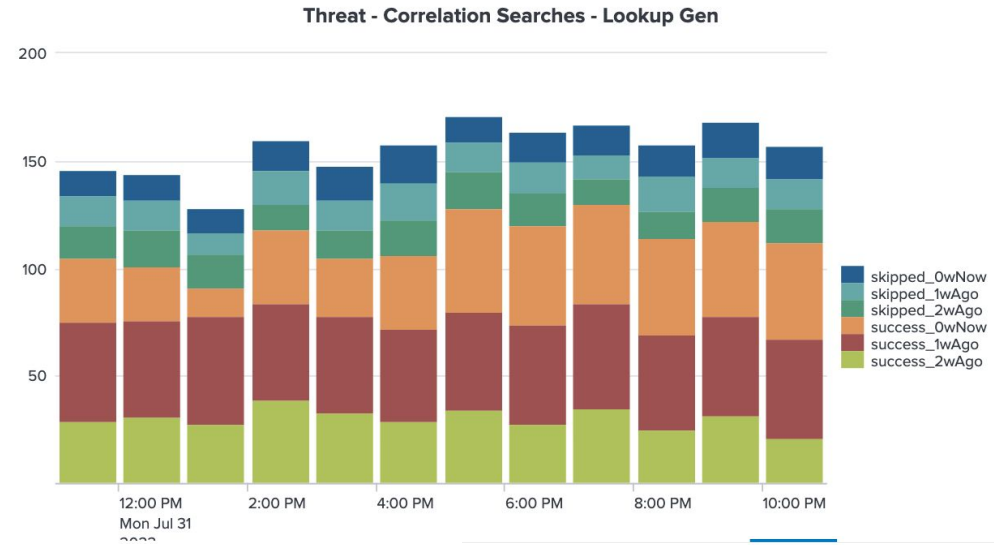
```
| loadjob <SID of Step 1>
| search NOT status IN ("continued", "delegated_remote", "delegated_remote_completion")
| eval comment = if(true(), null(), "
Set the condition for which savedsearch_name values to filter down to. Modify this eval logic to include/exclude other conditions in the final results.
In this case, filter to search names where current count is less than both 1w ago and 2w ago OR where number of failures/skipped searches is increased in current counts.")
| eval flag_Investigate = if( ( status="success" AND ((hist_count_0w < hist_count_1w) AND (hist_count_0w < hist_count_2w)) ) OR
                          ( (status="skipped" OR status="delegated_remote_error" OR status="failed") AND ((hist_count_0w >= hist_count_1w) OR
                           (hist_count_0w >= hist_count_2w)) ), "true", null() )
| eval comment = if(true(), null(), "
Use eventstats to compile a list of all savedsearch_names in the dataset where flag_Investigate is ever notnull.")
| eventstats values(eval(if(isnotnull(flag_Investigate), savedsearch_name, null()))) AS variant_search_names
| eval comment = if(true(), null(), "
Filter to search names that have varying numbers of values across different time periods by using mvfind(). Escape any special characters in the name to ensure the regex match behaves as intended.")
| where isnum(mvfind(variant_search_names, "^" . replace(savedsearch_name, "(?!\\\\)([^\w\s])", "\\\\\1") . "$"))
| fields - variant_search_names
| timechart limit=100 partial=f span=1h
    sum(eval(if(status="delegated_remote_error", hist_count_0w, null()))) AS delegated_remote_error_0wNow,
sum(eval(if(status="delegated_remote_error", hist_count_1w, null()))) AS delegated_remote_error_1wAgo,
sum(eval(if(status="delegated_remote_error", hist_count_2w, null()))) AS delegated_remote_error_2wAgo,
    sum(eval(if(status="failed", hist_count_0w, null()))) AS failed_0wNow, sum(eval(if(status="failed", hist_count_1w, null()))) AS
failed_1wAgo, sum(eval(if(status="failed", hist_count_2w, null()))) AS failed_2wAgo,
    sum(eval(if(status="skipped", hist_count_0w, null()))) AS skipped_0wNow, sum(eval(if(status="skipped", hist_count_1w, null()))) AS
skipped_1wAgo, sum(eval(if(status="skipped", hist_count_2w, null()))) AS skipped_2wAgo,
    sum(eval(if(status="success", hist_count_0w, null()))) AS success_0wNow, sum(eval(if(status="success", hist_count_1w, null()))) AS
success_1wAgo, sum(eval(if(status="success", hist_count_2w, null()))) AS success_2wAgo
    BY savedsearch_name
```

Utility

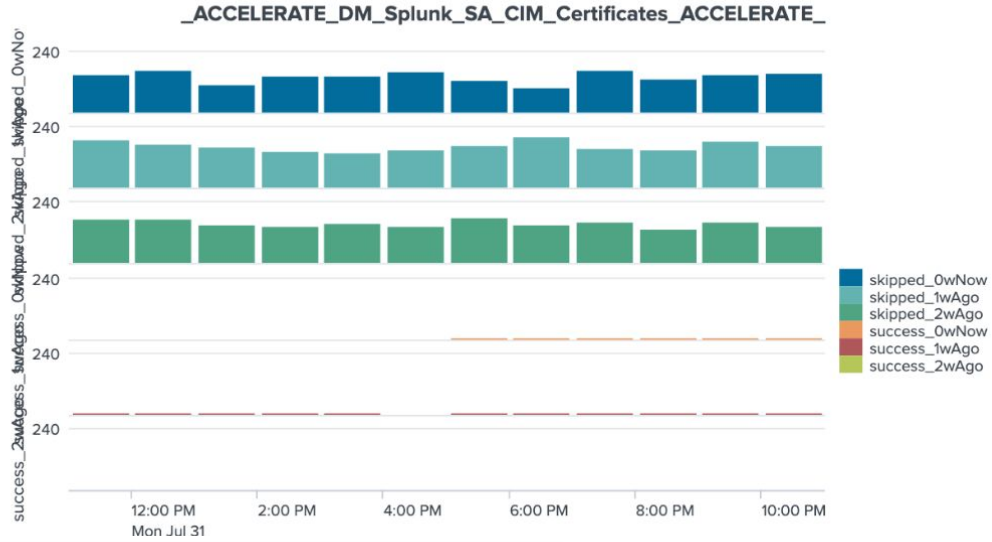
Search Scheduler Job Breakout Comparing Current Period to Same Period 1 week before and 2 weeks before (Step 2 Output)

Note - Recommended to set Visualization options to:
(1) `Column Chart, Stacked, Trellis enabled BY savedsearch_name`
If desired: **(2)** `Chart Overlay of a status` , or **(3)** `Multi-series Mode`

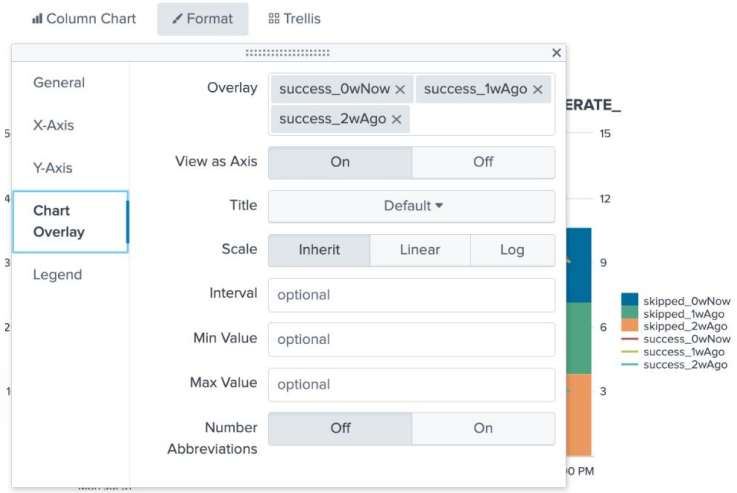
(1)

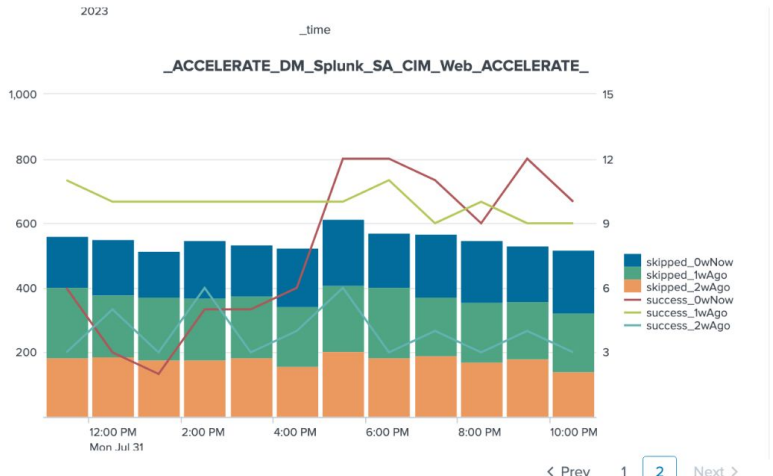


(2)



(3)





splunk> .conf23

Utility

FieldSummary/Data Analysis Presentation Utilities

These utilities are from another technical deep dive I gave -
What's in my Data? Field Analysis for the Advanced Engineer

Field Extraction Derivations - View All Field Object Definitions

Compile all field extraction objects information in environment by combining multiple REST endpoints into normalized, aggregated columns.

https://github.com/TheWoodRanger/splunk_fields_analysis_presentation#fields-objects-derivation-utility-query

Configured Datamodel Fields

Iterates across all configured datamodels to return all fields currently configured and their inheritance from datamodel hierarchy. Good way to check whether a field is configured within a datamodel or not.

https://github.com/TheWoodRanger/splunk_fields_analysis_presentation#configured-datamodel-fields

Fieldsummary Summarization of Data, Sampling and Top Values

Compile summarization information about field coverage, contents of the event data and sample values of each field. Batch analysis of data sources to better determine contents.

https://github.com/TheWoodRanger/splunk_fields_analysis_presentation#field-summarization-utility---no-group-by

Utility

Field Extraction Derivations - View All Field Object Definitions

Compile all field extraction objects information in environment by combining multiple REST endpoints into normalized, aggregated columns.
Note: query can take a long time to complete. Run the query and use | loadjob to analyze the results.

https://github.com/TheWoodRanger/splunk_fields_analysis_presentation#fields-objects-derivation-utility-query

dataSource	field_name	stanza	attribute	acl_app	acl_sharing	value	conf_specific_properties_mv
calcfields							
props-extractions	access	[access_combined]	REPORT-access	system	system	access-extractions	
props-extractions	access	[access_combined_wcookie]	REPORT-access	system	system	access-extractions	
props-fieldaliases	dvc	[aix_secure]	FIELDALIAS-dvc	Splunk_TA_nix	global	dest as dvc	alias.dest::::dvc
props-fieldaliases	dest	[audittrail]	FIELDALIAS-dest_for_splunk_access	Splunk_SA_CIM	global	host as dest	alias.host::::dest
props-lookups (automatic lookups)	action	[aix_secure]	LOOKUP-action_for_osx_secure	Splunk_TA_nix	global	nix_action_lookup vendor_action OUTPUTNEW action	overwrite::::0 transforms_stanza::::nix_action lookup_input::::vendor_action lookup_output::::action
props-lookups (automatic lookups)	machine search_group	[audittrail]	LOOKUP-dmc_add_instance_info	splunk_monitoring_console	app	dmc_assets host OUTPUTNEW machine search_group	overwrite::::0 transforms_stanza::::dmc_assets lookup_input::::host
transforms-extractions		[0365_account_microsoft_error_code]		Splunk_TA_microsoft-cloudservices	global	reason:\s\S+,\s(\{\"error\":\"([^"]+)\".*\})$	FORMAT::::error_info::$1 microsoft_error_code::$2 CAN_OPTIMIZE::::1 CLEAN_KEYS::::1 SOURCE_KEY::::_raw WRITE_META::::False KEEP_EMPTY_VALS::::0 LOOKAHEAD::::4096

Utility

Configured Datamodel Fields

Iterates across all configured datamodels to return all fields currently configured and their inheritance from datamodel hierarchy. Good way to check whether a field is configured within a datamodel or not.

https://github.com/TheWoodRanger/splunk_fields_analysis_presentation#configured-datamodel-fields

field	inheritance	datamodel	object
user	All_Changes.Endpoint_Changes	Change	Endpoint_Changes
vendor_product	All_Changes.Endpoint_Changes	Change	Endpoint_Changes
action	All_Changes.Endpoint_Changes	Change	Endpoint_Changes
_time	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
_raw	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
source	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
sourcetype	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
host	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
dest_bunit	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
dest_category	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
dest_priority	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
src_bunit	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
src_category	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts
src_priority	All_Changes.Endpoint_Changes.Endpoint_Restarts	Change	Endpoint_Restarts

Utility

Fieldsummary Summarization of Data, Sampling and Top Values

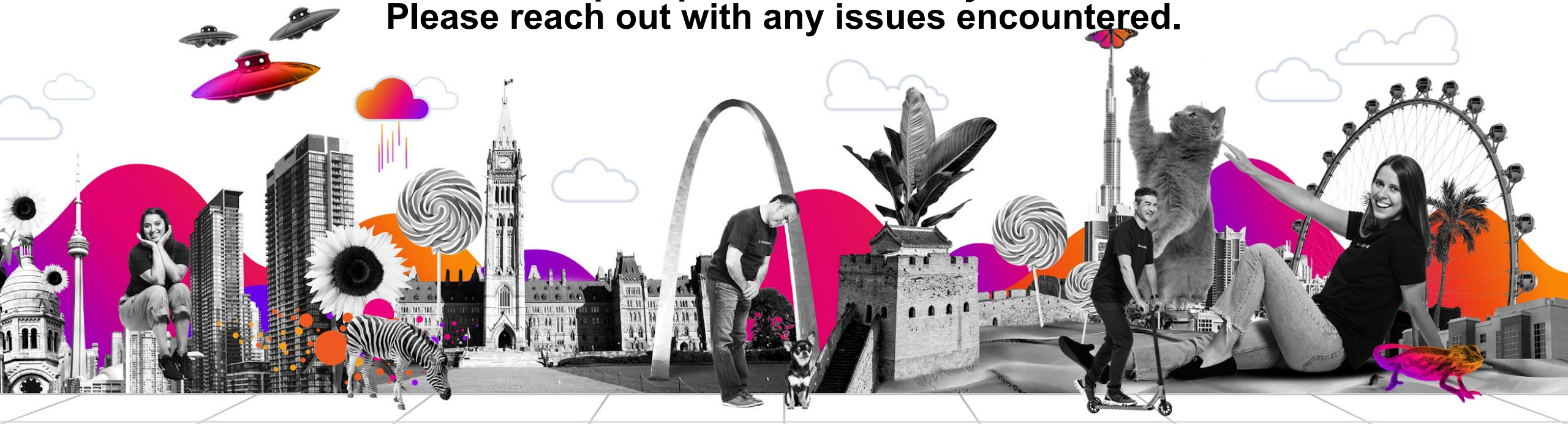
Compile summarization information about field coverage, contents of the event data and sample values of each field. Batch analysis of data sources to better determine contents. Multiple approaches are detailed in the presentation materials, showing the No Group-by method here.

https://github.com/TheWoodRanger/splunk_fields_analysis_presentation#field-summarization-utility---no-group-by

field	count	diffPerc	numeric_count	distinctValues	values
status	22512	98.7%	0	2 (Exact)	"success" :12796 "skipped" :9716
savedsearch_name	22512	98.7%	0	35 (Estimate)	"Threat - Correlation Searches - Lookup Gen" :1447 "_ACCELERATE_DM_SA-ThreatIntelligence_Incident_Management_ACCELERATE_" :964 "_ACCELERATE_DM_DA-ESS-ThreatIntelligence_Threat_Intelligence_ACCELERATE_" :961 "_ACCELERATE_DM_SA-ThreatIntelligence_Risk_ACCELERATE_" :955 "_ACCELERATE_DM_SA-NetworkProtection_Domain_Analysis_ACCELERATE_" :872 "_ACCELERATE_3E64B466-3A4F-475E-BC61-6C5DD2B2F161_InfoSec_App_for_Splunk_nobody_0ec400bee57dd68a_ACCELERATE_" :660 "_ACCELERATE_3E64B466-3A4F-475E-BC61-6C5DD2B2F161_InfoSec_App_for_Splunk_nobody_ff4c80c65c5958bd_ACCELERATE_" :645
priority	22512	98.7%	0	2 (Exact)	"highest" :12474 "default" :10038
search_type	22512	98.7%	0	3 (Exact)	"datamodel_acceleration" :16226 "scheduled" :4981 "report_acceleration" :1305
app	22512	98.7%	0	19 (Estimate)	"Splunk_SA_CIM" :12474 "SA-ThreatIntelligence" :3664 "DA-ESS-ThreatIntelligence" :1839 "InfoSec_App_for_Splunk" :1646 "SA-NetworkProtection" :1064 "TA-linux_auditd" :513 "SA-Utils" :371
user_watchlist	22512	98.7%	0	1 (Exact)	"false" :22512
user	22512	98.7%	0	2 (Exact)	"nobody" :18605 "admin" :3907

Thank You

For using the Addendum,
I hope it proves useful to you.
Please reach out with any issues encountered.



splunk> .conf23