

# Splunk4Ninjas - SPL Best Practices

EMET Technologies

splunk>



# Forward-Looking Statements

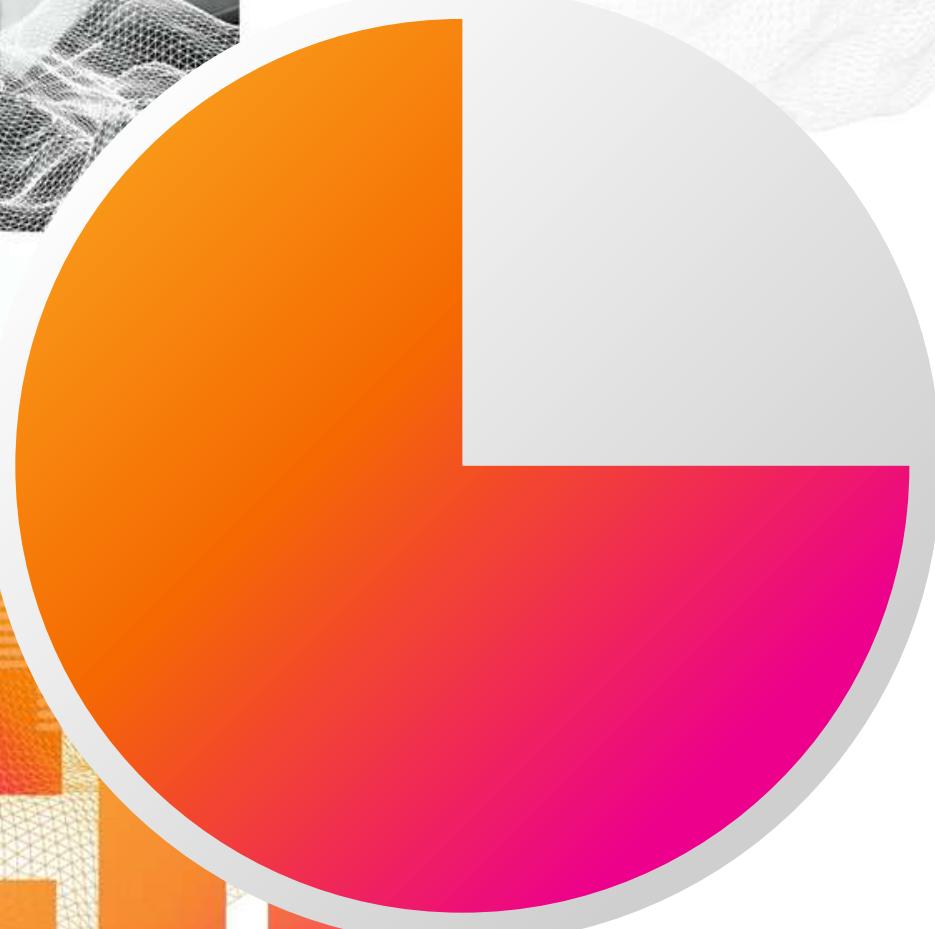


This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's 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](http://www.investors.splunk.com) or the SEC's website at [www.sec.gov](http://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, Splunk> and Turn Data Into Doing are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names or trademarks belong to their respective owners. © 2023 Splunk Inc. All rights reserved.



# Yaron Eilat

Presale Engineer

# Please introduce Yourself!

Name

Company/organisation

Role

What is your SPL experience?

What are your expectations?



# Agenda for Today's Workshop

- ✓ Recap: Splunk's Search Processing Language (SPL)
- ✓ Tips & Tricks for Writing More Efficient Searches
- ✓ SPL 201 > Getting Little more Advanced
- ✓ Tackling Difficult Data > Nested JSON
- ✓ Exploring Multi-Value Fields



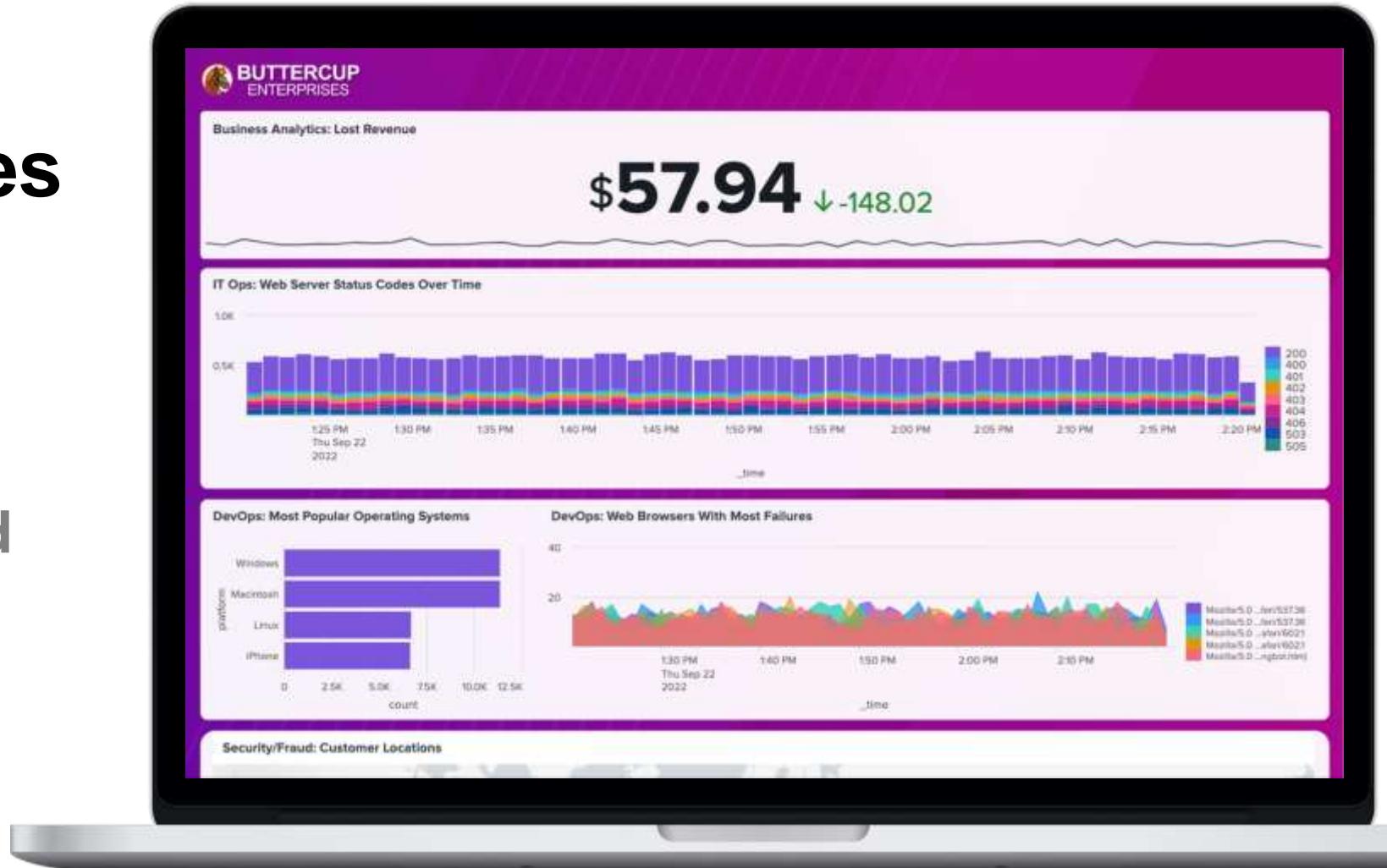
# Disclaimer

Please be advised that the environment we'll be using today is running on **Splunk 9.0**. Therefore some **features may not be available** for those of you still using **older versions of Splunk**.

# Splunk4Rookies Workshop >



Where we finished  
last time



# Recap: Search Processing Language (SPL)



# SPL (Search Processing Language) Refresher

Events are retrieved

Results passed linearly through SPL commands for processing

Search Terms

Commands

Clause

action=purchase | stats count by status | rename count as "number of events"

Pipe character: Output of left is input to right

e.g. action=purchase

Time	Event
15/09/2022 09:12:53.103	12.130.60.5 - - [15/Sep/2022:09:12:53:103] "GET /product.screen?product_id=MCR-5&SESSIONID=0245LJFT18DF14 HTTP/1.1" 401 1814 "http://www.buttercupenterprises.com/cart.do?action=purchase&item_id=EST-27&product_id=MCR-5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2956.16 Safari/537.36" 259 host = ip-172-31-99-95 source = /var/log/weblogs/noise_apache_2.log sourcetype = access_combined
15/09/2022 09:12:48.824	128.341.228.82 - - [15/Sep/2022:09:12:48:824] "GET /cart.do?action=purchase&item_id=EST-27&product_id=259-2&SESSIONID=0245LJFT18 HTTP/1.1" 404 2946 "http://www.buttercupenterprises.com/product.screen?product_id=259-2" "Mozilla/9.0 (iPhone; CPU iPhone OS 3_0 like Mac OS X) AppleWebKit/537.31 (KHTML, like Gecko) Version/9.0 Safari/9337.33 BingPreview/1.0" 661 host = ip-172-31-99-95 source = /var/log/weblogs/noise_apache_2.log sourcetype = access_combined
15/09/2022 09:12:42.794	141.148.8.86 - - [15/Sep/2022:09:12:42:794] "POST /cart.do?action=purchase&item_id=EST-19&product_id=MCR-5&SESSIONID=0245LJFT18DF19 HTTP/1.1" 305 3345 "http://www.buttercupenterprises.com/cart.do?action=purchase&item_id=EST-19&product_id=MCR-5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 Chrome/56.0.2956.16 Safari/537.36 OPR/43.0.2431.8 (Edition: developer)" 881 host = ip-172-31-99-95 source = /var/log/weblogs/noise_apache_1.log sourcetype = access_combined
15/09/2022 09:12:42.176	201.3.120.132 - - [15/Sep/2022:09:12:42:176] "POST /cart.do?action=purchase&item_id=EST-16&product_id=MCR-5&SESSIONID=0245LJFT18DF3 HTTP/1.1" 200 3542 "http://www.buttercupenterprises.com/product.screen?product_id=MCR-5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2956.16 Safari/537.36" 238 host = ip-172-31-99-95 source = /var/log/weblogs/noise_apache_1.log sourcetype = access_combined

Functions

| stats count by status

status	count
200	850
400	81
401	76
402	50
403	57

| rename count as "number of events"

status	number of events
200	850
400	81
401	76
402	50
403	57

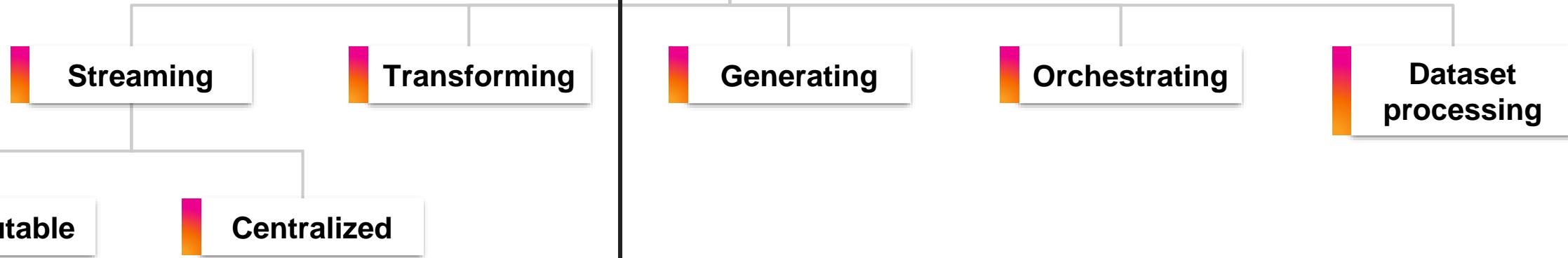
Want to know more? Check out:

Splunk Quick Reference Guide: <https://splk.it/SplunkQuickRef>

Search manual: <https://splk.it/SplunkSearchManual>

# SPL Command Types

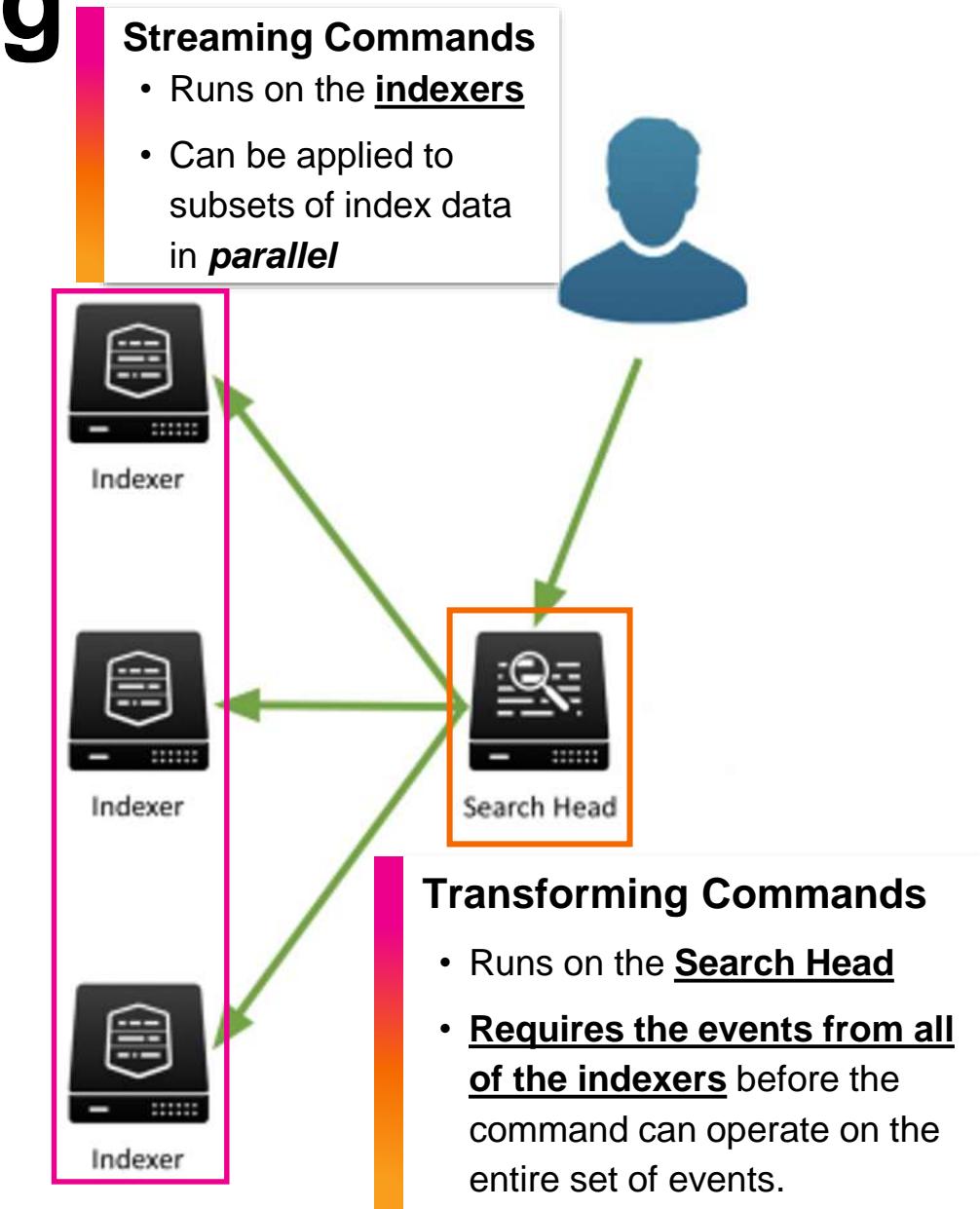
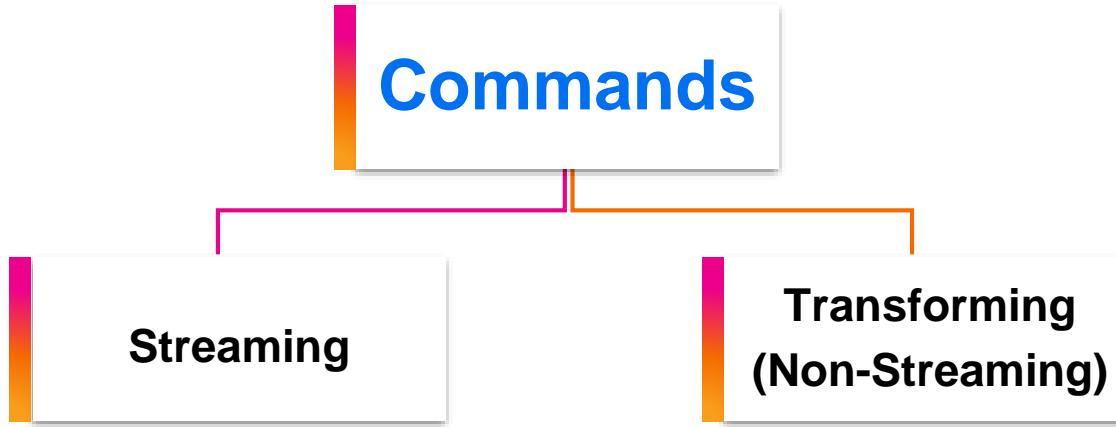
## Focus of Today's Workshop



Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/Search/Typesofcommands>

splunk>

# Streaming and Transforming Commands



# Comparison

## Streaming Commands

---

eval  
makemv  
rex  
spath

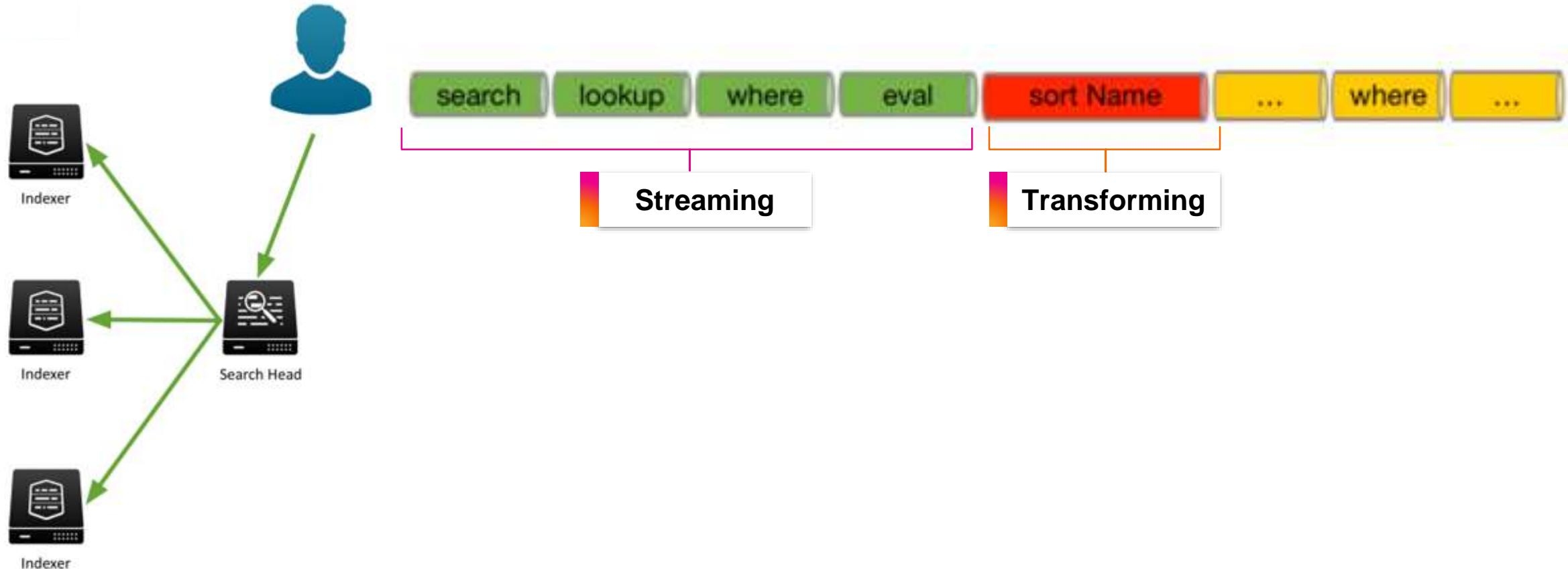
## Transforming Commands

---

chart  
timechart  
stats  
top  
rare

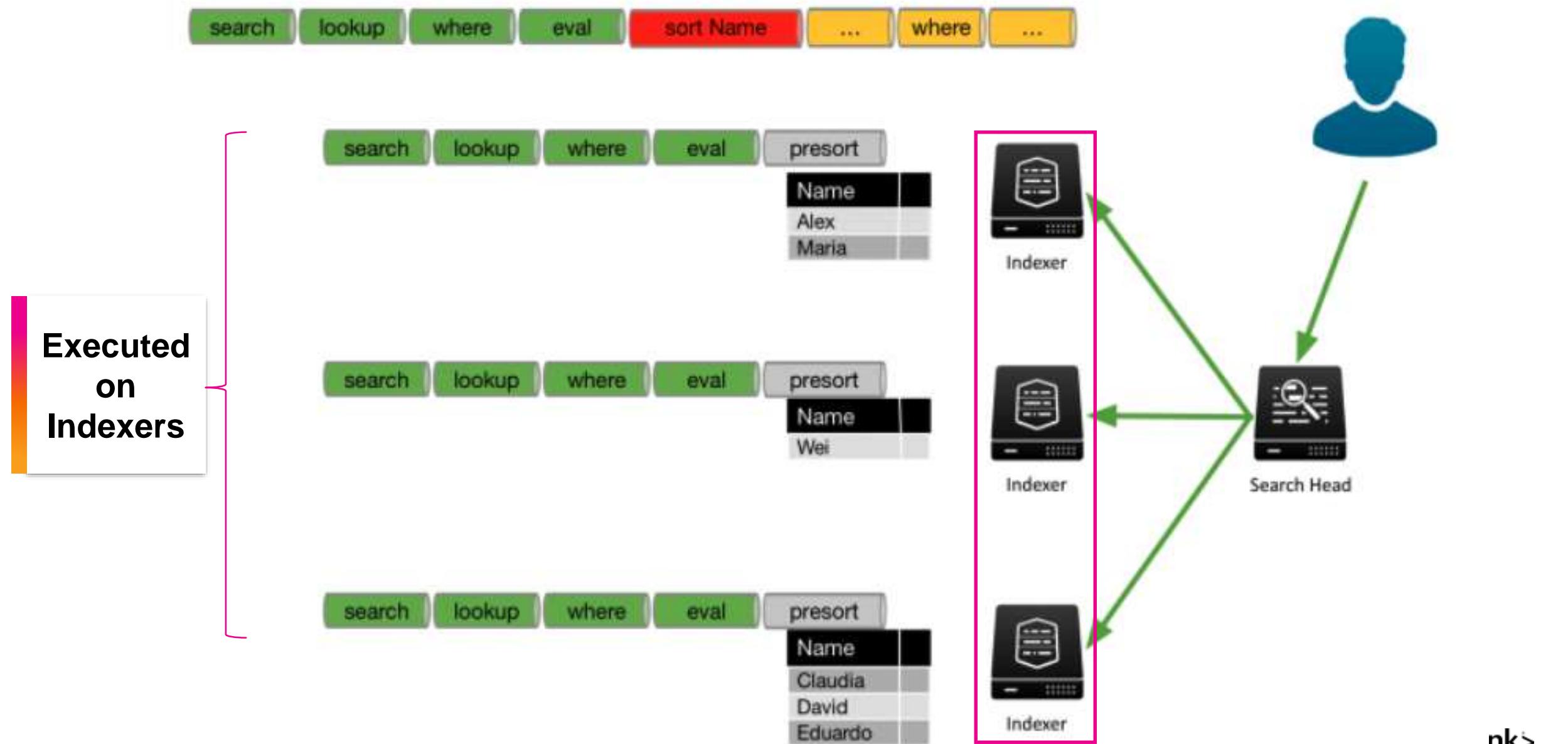
# Search Processing Example

User search > Search Head > Distributed to Indexers



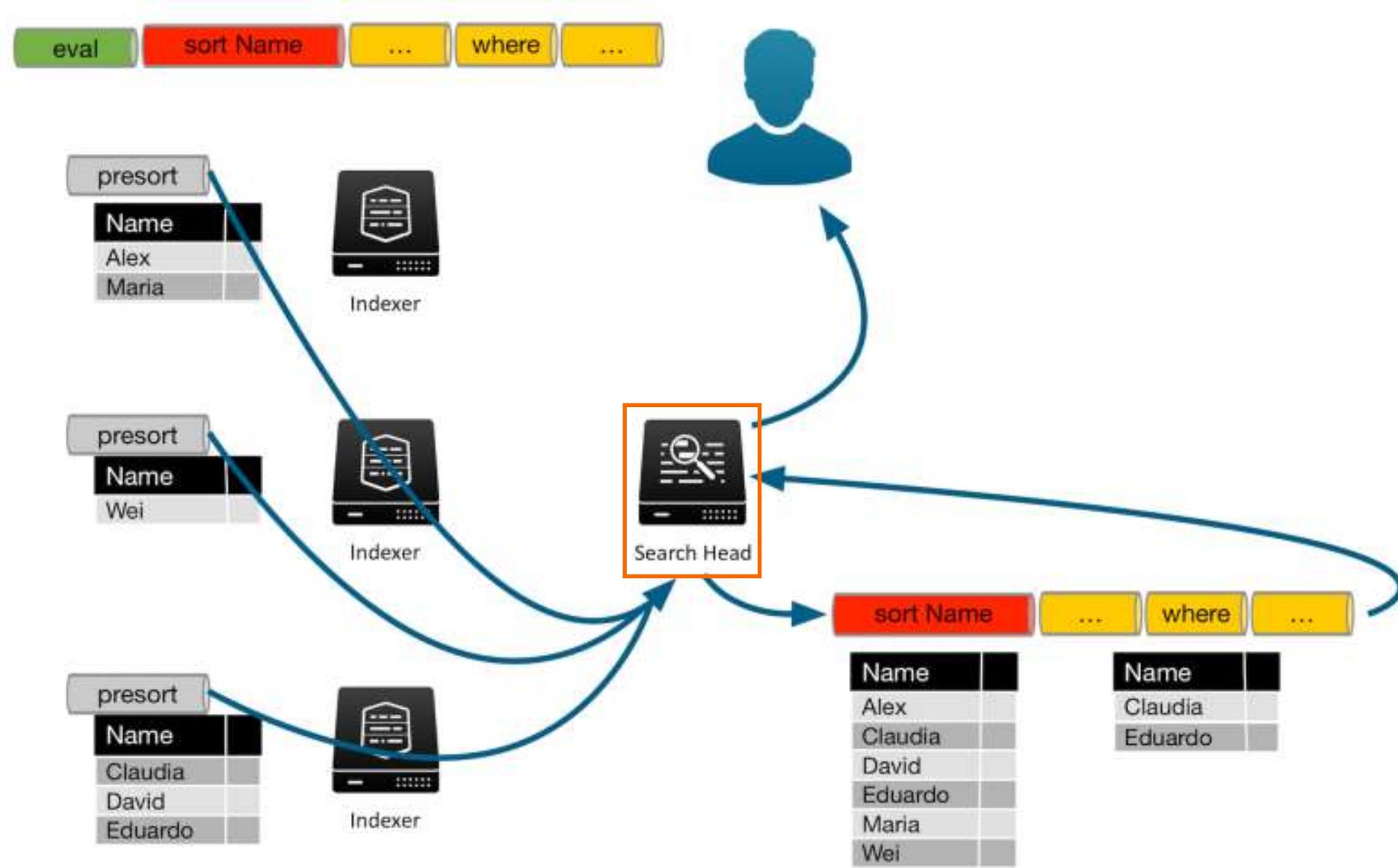
Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/Search/Writebettersearches>

# Search Processing Example



# Search Processing Example

- Results sent to **Search Head** > run **sort** on the **ENTIRE dataset**.
- **Following commands** run on **Search Head**.



# Tips for Writing Better Searches

## > Reduce the amount of data Splunk must Search

- Specify and limit the index(es) and other meta-fields like host(s), source(s) and sourcetype(s)
- Limit the time range for searching
- Fine-tune your searches to your unique events as much as possible
- Reduce the number of fields being passed down the SPL pipeline for processing (use `fields` command)



## > Distributed Search

- Place streaming commands earlier in the pipeline

```
index=my_index host=my_host sourcetype=my_srctype field1=val1 raw_text_search earliest=-1h latest=now
| eval field2=do_some_eval_here
| rename field2 as new_field2
| fields field1, new_field2
| stats count(field1) by new_field2
```

Be as specific as possible within search

Overrides the time-picker

Streaming commands used earlier in the search pipeline

Reduce number of fields in pipeline

Transforming commands used later in the search pipeline





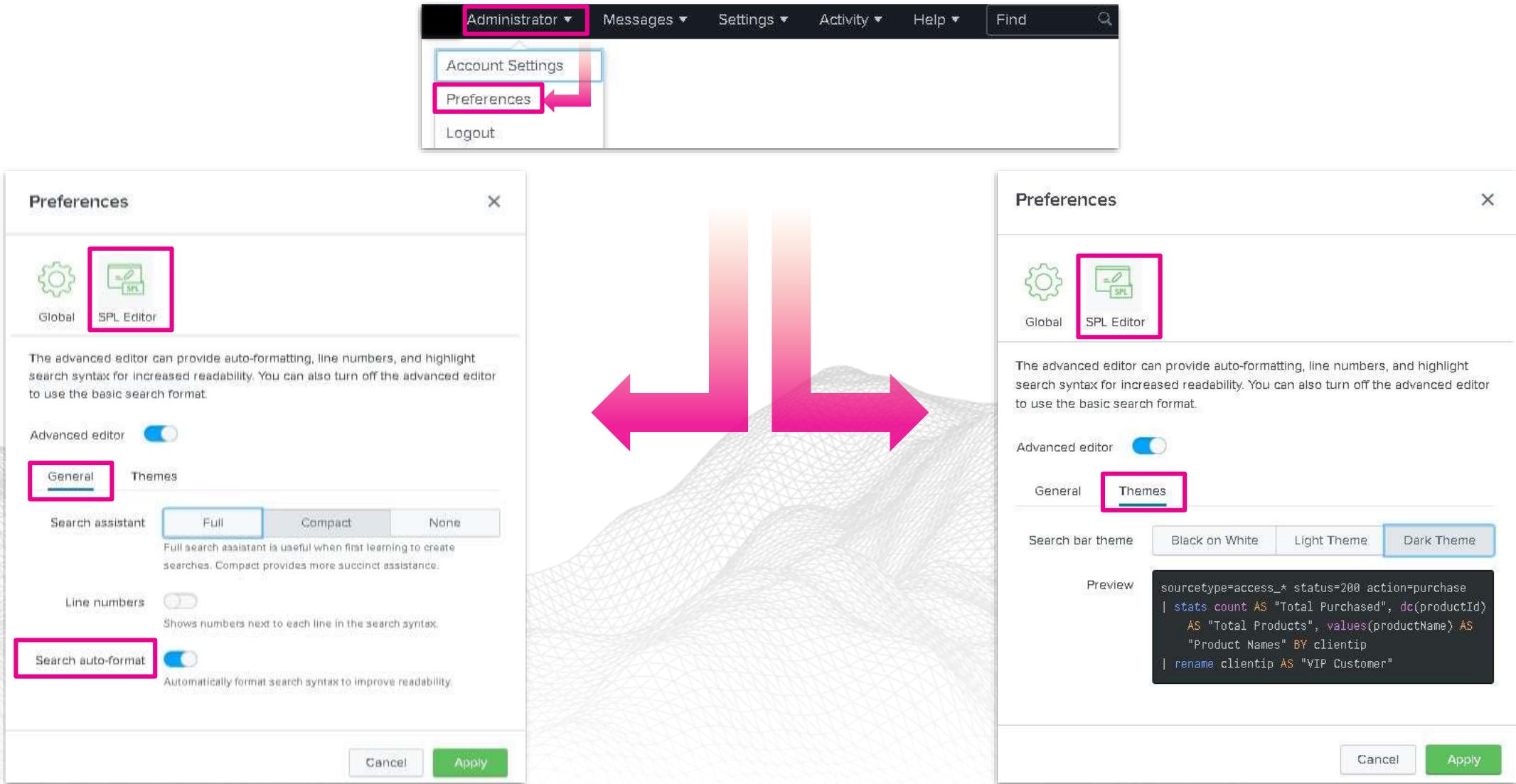
# Today's Scenario

You are a Splunk Power User at Buttercup Enterprise

Your colleagues who are new to Splunk and need your help to write better searches and deal with complex datasets that the systems generates

Your task is to help your colleagues in getting more value out of Splunk.

# UI Improvements



The screenshot illustrates several UI improvements across different Splunk components:

- Account Settings:** The "Administrator" dropdown menu is highlighted with a pink border. A pink arrow points from the "Preferences" link in the dropdown to the "Preferences" link in the main navigation bar below it.
- Preferences Dialog:** The "SPL Editor" icon is highlighted with a pink border. The "General" tab is selected, and the "Search auto-format" toggle switch is highlighted with a pink border. A large central diagram features two large pink arrows pointing towards each other, positioned between the left and right preference dialogs.
- Second Preferences Dialog:** The "SPL Editor" icon is highlighted with a pink border. The "Themes" tab is selected. The "Dark Theme" button is highlighted with a pink border. A preview window shows SPL search code.

# SPL Commenting

- Use three backticks (```) before and after your comment
- Comment out portions of your search to help identify and isolate problems
- To make very long SPL easier to read, add comments directly after the pipe (|)

```
index=security sourcetype=linux_secure
| ````single-series column chart````  
chart count over vendor_action
```

```
index=security sourcetype=linux_secure  
```| single-series column chart  
chart count over vendor_action```
```

```
index=security "failed password" earliest=-14d@d latest=@d
| ````line chart with week-to-week comparison````  
timechart span=1d count as Failures
| timewrap 1w
| rename _time as Day
| eval Day = strftime(Day, "%A")
```

## Useful to:

- Explain each "step" of a complicated search that is shared with other users.
- Discuss ways of improving a search with other users.
- Leave notes for yourself in unshared searches that are works in progress.
- Troubleshoot searches by running them with chunks of SPL "**commented out**".

## Example:

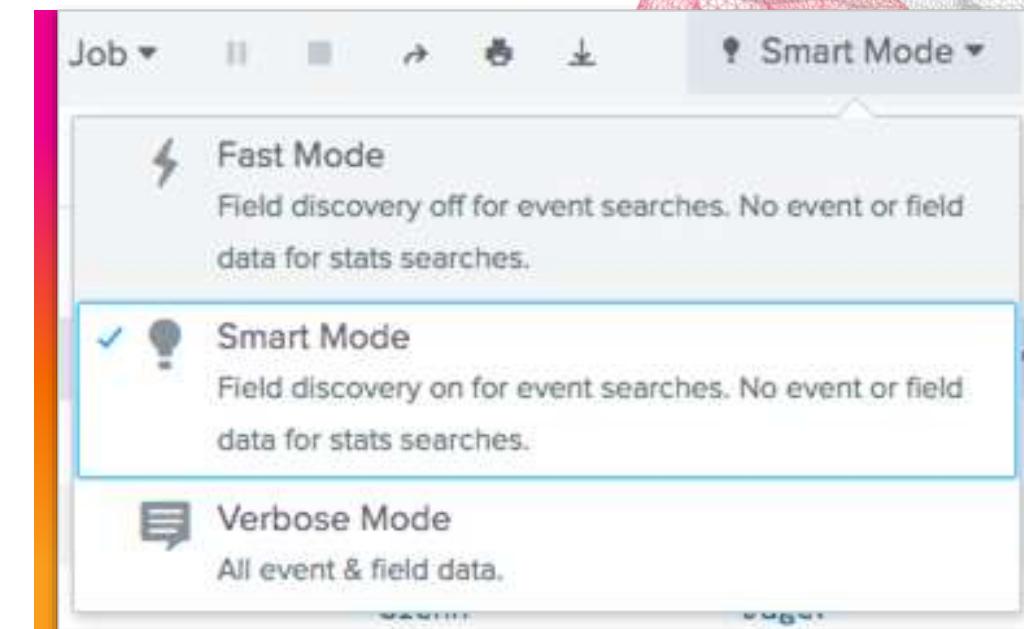
### Search

```
sourcetype=access_* status=200 ```Get all successful website access events.```
| stats count AS views count(eval(action="addtocart")) AS addtocart count(eval(action="purchase")) AS purchases by productName
    ```Create counts of site views, add-to-cart actions, and purchase actions. Break them out by product name.```
| eval viewsToPurchases=(purchases/views)*100 ```Find the ratio of site views to purchases.```
| eval cartToPurchases=(purchases/addtocart)*100 ```Find the ratio of add-to-cart actions to purchases.```
| table productName views addtocart purchases viewsToPurchases cartToPurchases ```Put all this data into a table.```
| rename productName AS "Product Name", views AS "Views", addtocart as "Adds To Cart", purchases AS "Purchases" ```Rename some table columns.```
▼
```

# Splunk Search Modes

Fast vs. Smart vs. Verbose

- › **Fast:** emphasizes speed over completeness
- › **Smart:** balances speed and completeness (default)
- › **Verbose:** emphasizes completeness



Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/Search/Changethesearchmode>

# What Happens on Data Ingestion?

**Event segmentation – data is broken into smaller segments for indexing**

- › Data is segmented by separating terms into smaller pieces
- › First with **major breakers** and then with **minor breakers**

Raw Data 91.205.189.15 -- [13/Aug/2019:18:22:16]

**Major segment breaker**  
(space is the breaker)

91.205.189.15

-

-

13/Aug/2019:18:22:16

91  
205  
189  
15  
91.205  
91.205.189  
91.205.189.15  
205.189  
and so forth

Minor segment breaker **dot**  
(.) breaks it further



# How is Data being Stored ?

Raw Events
10.0.0.6
9/28/2016
jeff@splunk.com



LEXICON

Term	Postings List
0	0
6	0
9	1
10	0
28	1
2016	1
10.0.0.6	0
9/28/2016	1
com	2
jeff	2
splunk	2
jeff@splunk.com	2

# SPL#1 > Successful Purchase for an IP

splunk>

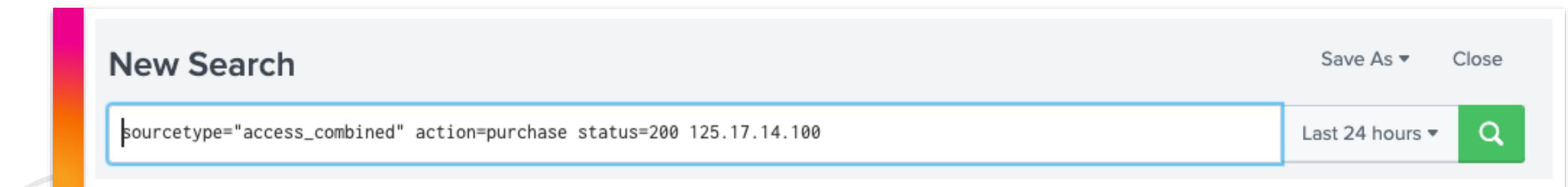


# SPL#1 > Successful Purchase Actions for an IP

Search for a status value of 200 for IP 125.17.14.100 – for Last 24 hours

**Solution:**

```
sourcetype=access_combined action=purchase status=200 125.17.14.100
```



# SPL#2 > Exploring the TERM directive

splunk>



# Learning to use TERM in SPL



TERM( ) directive is useful for finding **terms** more efficiently, when:

- They **contain** minor breakers
- Are **bound by** major breakers, such as spaces or commas
- They **don't contain** major breakers

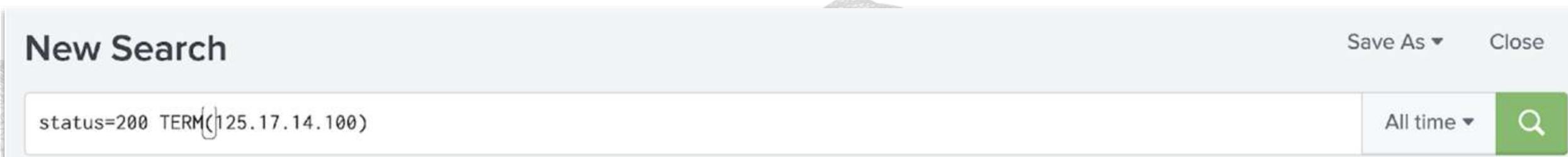
# Using the TERM directive

Add TERM for an IP Address

**Update your previous search to include TERM**

**Solution:**

status=200 TERM(125.17.14.100)



# Using the TERM directive

Add TERM for an IP address

**Inspect Job Efficiency > Investigate Execution Time  
> search.log (look for “base lispy”)**

New Search

status=200 TERM(125.17.14.100)

All time

✓ 260 events (before 6/20/21 1:37:08.000 PM) No Event Sampling

Events (260) Patterns Statistics Visualization

Format Timeline

Job

Inspect Job

Search job inspector

This search has completed and has returned 14 results.

(SID: 1622461331.83) [search.log](#)

UnifiedSearch - Expanded index search = (status=200 TERM(125.17.14.100))

UnifiedSearch - base lispy: [ AND 125.17.14.100 200 ]

Minor Segment (.)  
breakers are not used.  
The **Exact string** is  
searched in the data.

# When Can You Use TERM?

Your term **must** be bounded by major segmenters.

- E.g. spaces, tabs, carriage returns.
- See **segmenters.conf** for the complete list.

Your term **cannot** contain major segmenters.

## Few Major Breakers:

- A space
- A newline
- A tab
- Square brackets [ ]
- Parenthesis ( )
- Curly brackets { }
- An exclamation point !
- A question mark ?
- A semicolon ;
- A comma ,
- Single and double quotation marks ' "
- The ampersand sign &

ip 10.0.0.6 - 807256800 GET /images/launchlogo.gif		ip=TERM(10.0.0.6)
ip=10.0.0.6 - 807256804 GET /shuttle/missions.html		TERM(ip=10.0.0.6)
ip10.0.0.6 - 807256944 GET /history/history.html		TERM(ip10.0.0.6)
10.0.0.6:80 - 807256966 GET /skylab/skylab-4.html		TERM(10.0.0.6*)
9/28/16 1:30 PM - name=Willy Wonka sex=m age=46		TERM("Willy Wonka")

Learn more:

> segmenters.conf: <https://docs.splunk.com/Documentation/Splunk/latest/Admin/Segmentersconf>

> CASE and TERM: <https://docs.splunk.com/Documentation/Splunk/latest/Search/UseCASEandTERMtomatchphrases>

# Task using SPL#2 > Explore TERM Directive

## Try out the Searches



### Task 1

Exploring the usage of TERM  
directive

Try below searches and investigate/discuss the following:

- 1) Are all of these 3 searches going to work?
- 2) In **search.log** find “**base lispy**” and review the output.

a) TERM(131.178.233.243)

b) TERM(status=200)

c) product\_id=TERM(MCB-6)

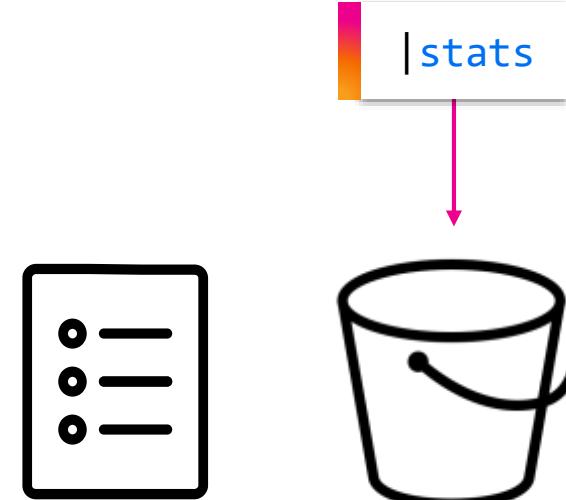
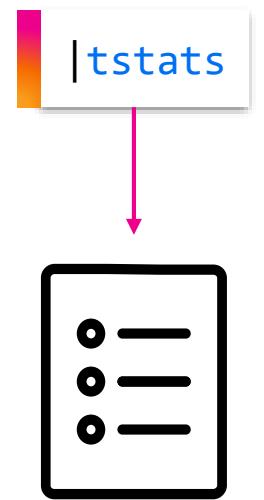
# SPL#3 > *tstats* with PREFIX

splunk>



# Why use `tstats`?

- `tstats` is faster than `stats` because it only looks at the indexed metadata (the **tsidx files**), while `stats` looks at the **raw data** from the upstream query.



Learn more:

<https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Tstats>

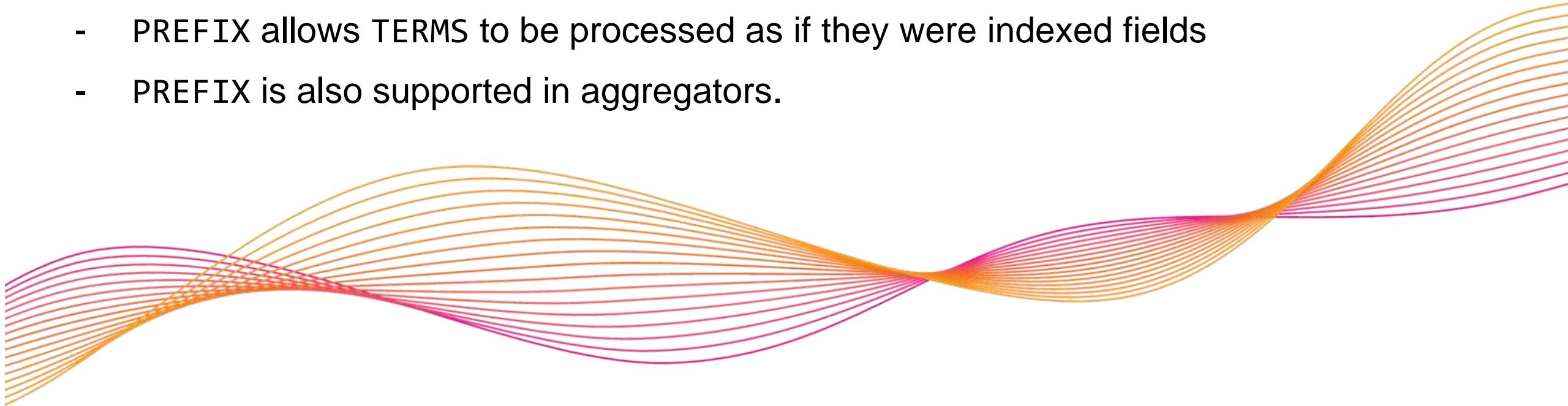
splunk>

# tstats limitations and challenges

- Pre existence of indexed fields - indexed fields can be either from indexed data, or from Data Model acceleration.
  - At ingestion we can extract metadata from raw event and create indexed fields.
  - Some structured data sources can optionally create indexed fields automatically.
  - The HTTP Event collector has a “**fields**” section.
  - Post ingestion we can create a **datamodel**.
- It is difficult to discover the existence of indexed fields when available
  - The [walklex](#) command helps
  - The existence of TERMS can be inferred from log data

# Introducing **tstats** with PREFIX

- With PREFIX, indexed fields are no longer a requirement
- The PREFIX directive massively increases the instances where **tstats** can be used
- PREFIX allows TERMS to be processed as if they were indexed fields
- PREFIX is also supported in aggregators.



Learn more: <https://conf.splunk.com/files/2020/slides/PLA1089C.pdf>

# What is the PREFIX directive?

Search on a raw segment in your indexed data as if it were an extracted field.

Locate a recurring **segment** in your raw event data.

Similar to the CASE and TERM directives in that, that it matches strings in your raw data.

**Must be** in lower case - the matching is **NOT** case sensitive.

## PREFIX

# Differences between a classic search and tstats

Consider the following log:

```
01-21-2020 12:25:44.311 +0000 INFO Metrics
- group=thruput, ingest_pipe=1,
name=thruput,
instantaneous_kbps=3.366894499322308,
instantaneous_eps=12.163696322058637,
average_kbps=47.777961955016565,
total_k_processed=31355244,
kb=104.6298828125, ev=378,
load_average=2.42
```

# Differences between a classic search and tstats

```
01-21-2020 12:25:44.311 +0000 INFO Metrics -  
group=thruput, ingest_pipe=1, name=thruput,  
instantaneous_kbps=3.366894499322308,  
instantaneous_eps=12.163696322058637,  
average_kbps=47.777961955016565,  
total_k_processed=31355244, kb=104.6298828125,  
ev=378, load_average=2.42
```

## Classic search

```
index=_internal group=thruput name=thruput  
| bin span=1767s _time  
| stats  
sum(kb) as indexer_kb  
avg(instantaneous_kbps) as instantaneous_kbps  
avg(load_average) as load_avg  
by host _time
```

# Differences between a classic search and tstats

```
01-21-2020 12:25:44.311 +0000 INFO Metrics -  
group=thruput, ingest_pipe=1, name=thruput,  
instantaneous_kbps=3.366894499322308,  
instantaneous_eps=12.163696322058637,  
average_kbps=47.777961955016565,  
total_k_processed=31355244,  
kb=104.6298828125, ev=378, load_average=2.42
```

## Classic search

```
index=_internal group=thruput name=thruput  
| bin span=1767s _time  
| stats  
sum(kb) AS indexer_kb  
avg(instantaneous_kbps) AS instantaneous_kbps  
avg(load_average) AS load_avg  
BY host _time
```

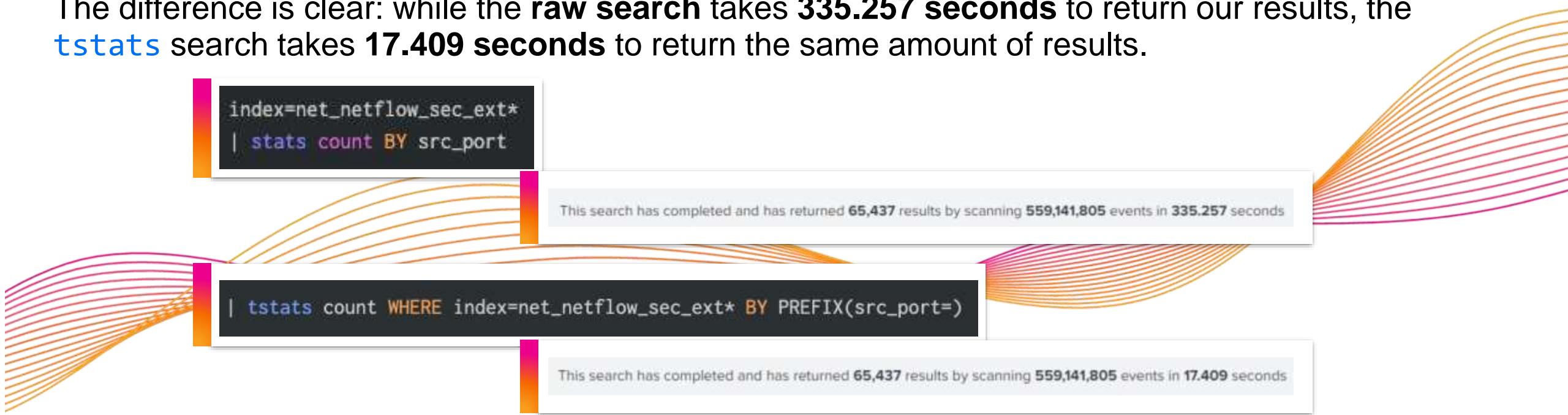
## tstats search

```
|tstats sum(PREFIX(kb=)) AS indexer_kb avg(PREFIX(instantaneous_kbps=)) AS instantaneous_kbps avg(PREFIX(load_average=)) AS load_avg  
WHERE  
index=_internal  
TERM(group=thruput)  
BY host _time  
span=1767s
```

# Simple search across Netflow logs

Now let's have a look at a real world sample dataset to see the difference between `tstats` and a raw search, where we can see the difference between one and the other. The total number of events in this case is 559,141,805.

The difference is clear: while the **raw search** takes **335.257 seconds** to return our results, the **`tstats`** search takes **17.409 seconds** to return the same amount of results.



# tstats Challenge

How many actions are performed by **jsessionid**, for the most active 10 sessions?

```
| tstats count(PREFIX(action=)) AS count WHERE index=main BY PREFIX(jsessionid=) | rename jsessionid= AS jsession | sort - count | head 10
```

jsession	count
sd2s13ff6adff2	8
sd3s17ff1adff7	8
sd5s18ff4adff7	8
sd7s12ff5adff3	8
sd10s14ff8adff1	7
sd10s15ff3adff8	7
sd10s15ff6adff2	7
sd10s15ff8adff4	7
sd10s18ff10adff10	7
sd1s11ff6adff2	7

# SPL#4 > makeresults and eval

splunk>



# eval command

For example, defining a variable in programming

The **eval** command **calculates an expression and puts the resulting value into a search results field**

**Example:** ... | eval velocity=distance/time

If the fields velocity  
is not present in events

Creates a **New Field**  
“velocity” & add it to  
search results

If the field velocity  
is present in events

**Overwrites** the value  
of field “velocity”  
in search results

Calculates **velocity** if  
“distance” & “time”  
fields **are present** in  
the events

# eval Command

Versatile command for calculations and operations

What can be done with the `eval` command? A few **examples**:

## 1. If/else conditions

```
... | eval error=if(status==200, "OK", "Problem")
```

## 2. Case

```
... | eval error_msg=case(status==404, "Not found", status==503,  
"Service Unavailable", status==200, "OK")
```

## 3. Mathematical Operations

```
... | eval velocity=round(distance/time,2)
```

## 4. Field operations – combine, split, concatenate, check matches (any many more...)

```
sourcetype=A OR sourcetype=B | eval phone=coalesce(number,subscriberNumber)
```

# SPL#4 > makeresults Command

`eval` executes sequentially > allows **Mathematical, Boolean & String** calculations

New Search Save As ▾ Close

```
| makeresults  
| eval distance_km=200,time_hours=10,  
    speed_kmph=distance_km/time_hours,  
    speed_meters_per_sec=speed_kmph/3.6,  
    speed_meters_per_sec=toString(speed_meters_per_sec)." m/s"  
| table distance_km, time_hours, speed_kmph, speed_meters_per_sec
```

Last 24 hours 🔍

Single `eval` command performing multiple operations

✓ 1 result (5/18/21 6:00:00.000 AM to 5/19/21 6:38:40.000 AM) No Event Sampling ▾ Job ▾ ⋮ Smart Mode ▾

Events Patterns Statistics (1) Visualization

20 Per Page ▾ Format Preview ▾

distance_km	time_hours	speed_kmph	speed_meters_per_sec
200	10	20	5.6 m/s

# Task using SPL#4 > Using eval to customize outputs

Use eval to format the final output

## Try out the Searches



### Task 2

Trying your hands at eval command

In our dataset we have 3 categories of products:

- 1) Books; 2) Gifts; 3) Clothing.

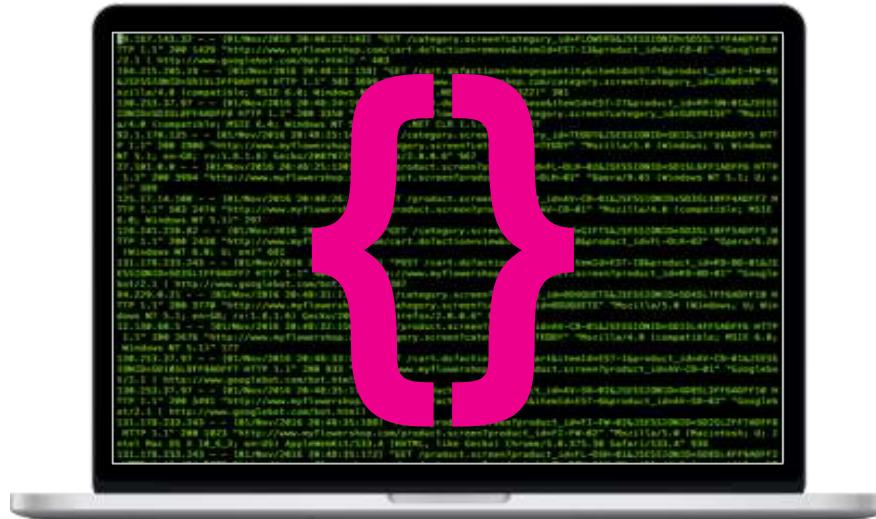
Depending upon the **top category** in your dataset, find the top sold category and output the final message as **one of below**.

Most of Our Customers Love Reading Books

Most of Our Customers Love Sharing Gifts

Most of Our Customers Love Trying New Clothes

# SPL#5 > eval's Hidden Feature



splunk>

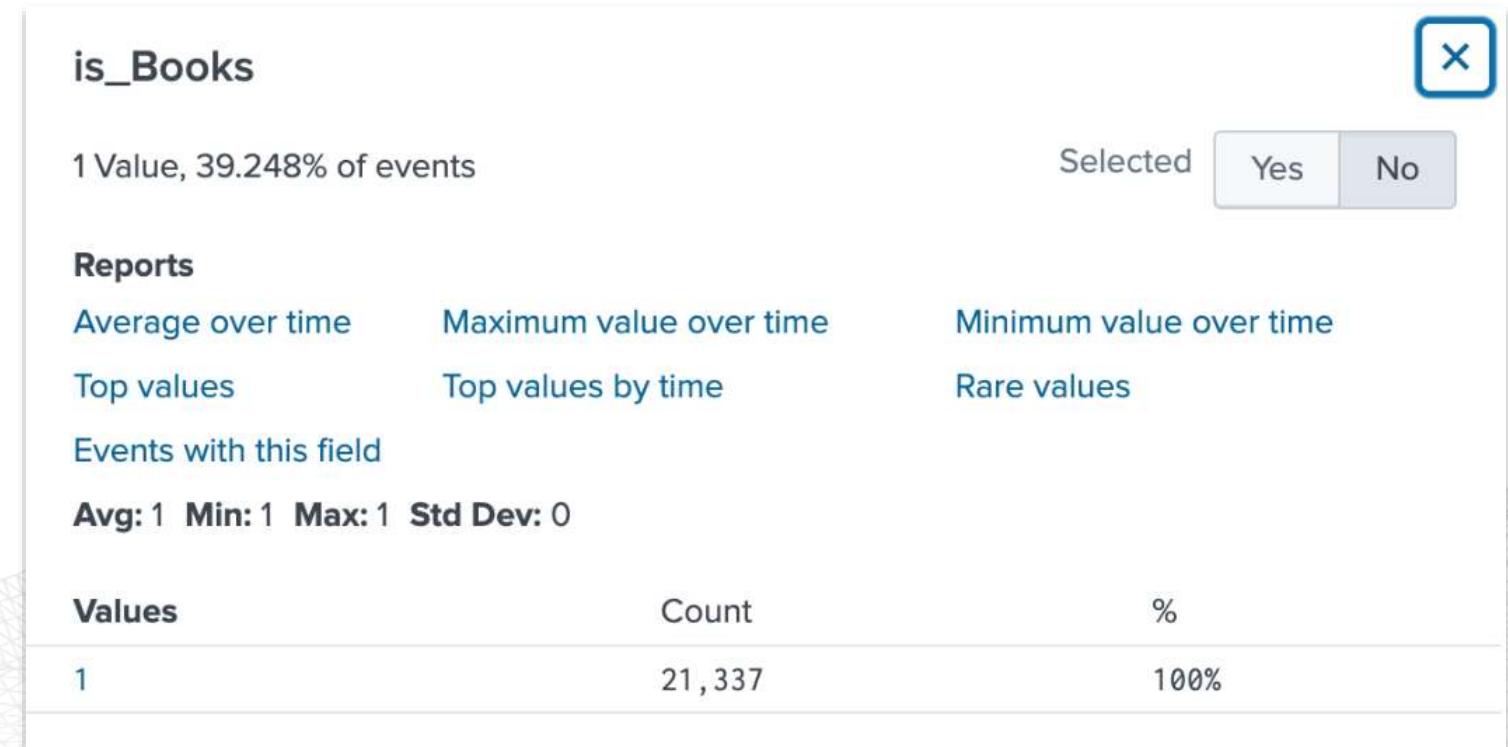


# eval and the Field Name Trick

We can use `eval` to **dynamically** create additional **fields** based on the **value** of another field. The trick is to use curly braces “`{}`”.

```
sourcetype = access_combined  
| eval is_{category} = 1
```

```
# is_Books 1  
# is_Clothing 1  
# is_Gifts 1
```



# eval and the Field Name Trick

That is nice to know, but what problem are we solving here? What's the point?



# eval and the Field Name Trick

Let's answer this question with a practical exercise  
a.k.a. Challenge Time!

How could we count the **Total Sales** for each **category**, and get the results in a format suitable to use for a **multi-metric** or a **base search like below?**

Total\_Sales\_of\_Books  

41305.21

Total\_Sales\_of\_Clothing  

103928.31

Total\_Sales\_of\_Gifts  

29798.31

# eval and the Field Name Trick

We could get to the result with `stats` alone, but the **output format** is not ideal for a **base search**

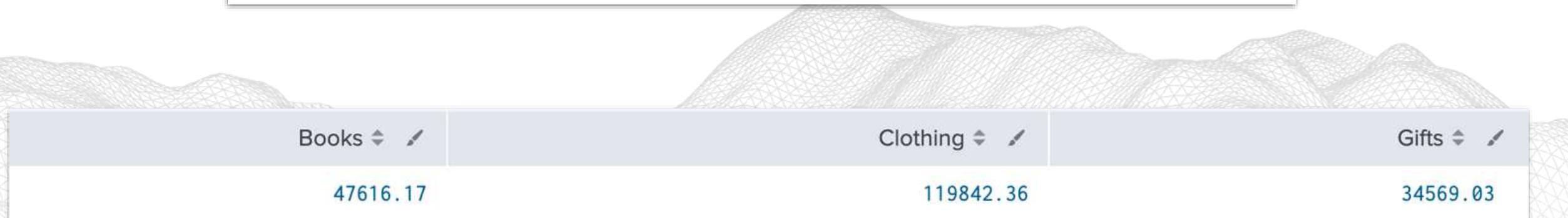
```
sourcetype = access_combined action=purchase  
| stats sum(product_price) as Total_Sales by category
```

category	Total_Sales
Books	46575.37
Clothing	116475.66
Gifts	33750.60

# eval and the Field Name Trick

We could do that with `timechart`, but this is **not the fastest** way and could lead to inconsistent results depending on the use case.

```
sourcetype = access_combined action=purchase  
| timechart span=1w sum(product_price) as "Total Sales" by category  
| fields - _time
```



# eval and the Field Name Trick

Here's a more elegant way using `eval` and the **field name trick**

- We keep only the purchase actions
- We create a new field called “**Total\_Sales\_of\_{category}**” and give each event the **product\_price** as value
- We combine with `stats` and **wildcards** to calculate the total amount sold per **category**

```
sourcetype = access_combined action=purchase  
| eval Total_Sales_of_{category} = product_price  
| stats sum(Total_Sales_of_*) as Total_Sales_of_*
```

Total\_Sales\_of\_Books 

41305.21

Total\_Sales\_of\_Clothing 

103928.31

Total\_Sales\_of\_Gifts 

29798.31

# Break?



splunk>

# SPL#6 > Complex Queries



```
sourcetype="complex_query"
| eval ticket_end_time =
if(actionCode=="D", strftime(actionTime,"%Y/%m/%d
%H:%M:%S"),NULL)
| eval ticket_start_time =
if(actionCode=="I", strftime(firstOccurrence,"%Y/%m/%d
%H:%M:%S"),NULL)
| transaction serverName serverSerial ticketNumber
| eval ticketDuration = ticket_end_time -
ticket_start_time
| eval pretty_ticketDuration =
floor(ticketDuration/60/60)." Hours
".floor(floor(ticketDuration -
(floor(ticketDuration/60/60)*60*60))/60)." Min
".floor(ticketDuration%60)." Sec."
| stats avg(ticketDuration) as
Average_Ticket_Duration list(pretty_ticketDuration)
as pretty_ticketDuration by serverName serverSerial
```



# Macros

- **Reusable search strings or portions of search strings**
- **Time range independent**
- **Pass arguments to the search**

splunk>

# Creating a Macro

**1. Click on Settings**

The screenshot shows the Splunk Settings interface. At the top, there are navigation items: Administrator, 4 Messages, and a Settings dropdown menu. Below these are several sections: KNOWLEDGE (Searches, reports, and alerts; Data models; Event types; Tags; Fields; Lookups; User interface; Alert actions), DATA (Data inputs; Forwarding and receiving; Indexes; Report acceleration summaries; Virtual indexes; Source types), SYSTEM (Server settings; Server controls; Health report manager; Instrumentation; Licensing; Workload management), and USERS AND AUTHENTICATION (Roles; Users; Tokens; Password Management; Authentication Methods). A sidebar on the left includes icons for Add Data, Explore Data, and Monitoring Console, each with a corresponding list of sub-options.

**2. Click on Advance search**

This screenshot shows the Splunk Advanced search interface. It features a sidebar with 'Advanced search' highlighted. Other sections include INDEX (Indexer clustering; Forwarder management; Data Fabric; Distributed search) and a 'Recent' section with links to 'Saved searches' and 'Dashboards'.

**3. Click on +Add new**

The screenshot shows the 'Create macro' dialog box. It has fields for 'Destination app' (set to S4N\_SPLBP), 'Name' (set to totalUSD), and 'Definition'. The 'Definition' field contains the following SPL command:

```
stats sum(product_price) as total_sales by product_name  
| eval total_sales=("$" + tostring(round(total_sales,2),",commas"))
```

Below the definition, there's a checkbox for 'Use eval-based definition?' which is checked. The dialog also includes sections for 'Arguments', 'Validation Expression', and 'Validation Error Message'. At the bottom right are 'Cancel' and 'Save' buttons, with 'Save' highlighted.

**4. Choose Destination app, Name, Definition**

# Using Macros

To use a macro use the **backtick character**:



## New Search

```
index=main sourcetype="access_combined"
`totalUSD`
```

✓ 38,290 events [3/21/23 3:00:00.000 PM to 3/22/23 3:04:48.000 PM] No Event Sampling ▾

Events (38,290) Patterns Statistics (10) Visualization

20 Per Page ▾ ✓ Format Preview ▾

product_name	total_sales
Batguy Slippers	\$95,269.98
Batguy Watch	\$37,122.84
Costume- ManHawk	\$372,547.58
Double Fudge Sundae	\$84,061.25
Mad Comics- Batguy	\$47,955.20
Mad Comics- Bronze Man	\$47,184.38
Mad Comics- Flyman	\$48,564.88
Pony Potpourri	\$37,962.00
Waterproof Scratch and Sniff	\$18,622.68
Zombie Survival Guide	\$56,778.93

# Adding Macro Arguments

Destination app **S4N\_SPLBP**

**Name \*** Enter the name of the macro. If the search macro takes an argument, indicate this by appending the number of arguments to the name. For example: mymacro(2)  
**totalUSD(2)**

**Definition \*** Enter the string the search macro expands to when it is referenced in another search. If arguments are included, enclose them in dollar signs. For example: \$arg1\$  
`stats $function$(product_price) as $newname$ by product_name  
| eval $newname$=("$" + tostring(round($newname$,2),"commas"))`  
 Use eval-based definition?

**Arguments** Enter a comma-delimited string of argument names. Argument names may only contain alphanumeric, '-' and '\_' characters.  
**function, newname**

**Validation Expression** Enter an eval or boolean expression that runs over macro arguments.

**Validation Error Message** Enter a message to display when the validation expression returns 'false'.

**Cancel** **Save**

- Great way to make `macros` more flexible.
- Number of arguments needs to be included in parenthesis in the name field.
- Argument names are surrounded by dollar(\$) signs.

# Using Macro with arguments

Allows us to change variables easily at search time.

New Search

```
index=main sourcetype="access_combined"
| `totalUSD(avg,average_price)`
|sort average_price
```

✓ 3,808 events (3/21/23 4:00:00.000 PM to 3/22/23 4:08:18.000 PM) No Event Sampling ▾

Events Patterns Statistics (10) Visualization

20 Per Page ▾ Format Preview ▾

product_name	average_price
Mad Comics- Batguy	\$12.70
Mad Comics- Bronze Man	\$12.70
Mad Comics- Flyman	\$12.70
Zombie Survival Guide	\$15.21
Double Fudge Sundae	\$22.75
Batguy Slippers	\$25.70
Waterproof Scratch and Sniff	\$4.99
Batguy Watch	\$9.99
Pony Potpourri	\$9.99
Costume- ManHawk	\$97.50

# Arguments Validation

**Destination app** S4N\_SPLBP

**Name \*** Enter the name of the macro. If the search macro takes an argument, indicate this by appending the number of arguments to the name. For example: mymacro(2)  
totalUSD(3)

**Definition \*** Enter the string the search macro expands to when it is referenced in another search. If arguments are included, enclose them in dollar signs. For example: \$arg1\$  
stats \$function\$(product\_price) as \$newname\$ by product\_name  
|\$format\$ \$newname\$=("\$" + tostring(round(\$newname\$,2), "commas"))

Use eval-based definition?

**Arguments** Enter a comma-delimited string of argument names. Argument names may only contain alphanumeric, '\_' and '-' characters.  
function, newname, format

**Validation Expression** Enter an eval or boolean expression that runs over macro arguments.  
\$format\$="fieldformat" OR \$format\$="eval"

**Validation Error Message** Enter a message to display when the validation expression returns 'false'.  
totalUSD(3) must include a stats function, a field name for the new value, and a format argument of eval or

**Cancel** **Save**

- Validation expression arguments to be surrounded by dollar(\$) signs.
- Can use an eval expression or boolean operators.
- Provide informative message for the end user in case the `macro` returns with error.

# Arguments order

Arguments needs to be passed in the order they appear in the definition

New Search

```
index=main sourcetype="access_combined"
| `totalUSD(sum,total_sales,fieldformat)`
|sort total_sales
```

1 2 3

✓ 10,073 events (3/21/23 4:00:00.000 PM to 3/22/23 4:42:59.000 PM) No Event Sampling ▾

Events Patterns Statistics (10) Visualization

20 Per Page ▾ ✓ Format Preview ▾

product_name	total_sales
Waterproof Scratch and Sniff	\$4,818.36
Batguy Watch	\$9,200.79
Pony Potpourri	\$9,750.24
Mad Comics- Batguy	\$12,407.90
Mad Comics- Flyman	\$12,776.20
Mad Comics- Bronze Man	\$12,890.50
Zombie Survival Guide	\$15,362.10
Double Fudge Sundae	\$23,182.25
Batguy Slippers	\$25,597.20
Costume- ManHawk	\$95,842.50

# Macro validation

Executing the macro incorrectly will produce an error for the end user that we specified earlier in our “Validation error message” field.

## New Search

```
index=main sourcetype="access_combined"
| `totalUSD(sum,total_sales,count)`
|sort total_sales
```

 Error in 'SearchParser': Encountered the following error while validating macro 'totalUSD(sum,total\_sales,count)': totalUSD(3) must include a stats function, a field name for the new value, and a format argument of eval or fieldformat.

# Nested Macros

Destination app: S4N\_SPLBP

Name\*: EUsales

Definition\*: |search region\_un=Europe  
|`totalUSD`

Use eval-based definition?

Arguments: Enter a comma-delimited string of argument names. Argument names may only contain alphanumeric, '\_' and '-' characters.

Validation Expression: Enter an eval or boolean expression that runs over macro arguments.

Validation Error Message: Enter a message to display when the validation expression returns 'false'.

Cancel Save



**Inner `Macros` should always be created first before passing them to the outer macros !**

# Expand Macros SPL

The screenshot shows a Splunk search interface for the dataset "EUsales". The search bar at the top contains the query "EUsales". Below the search bar, the results section displays 521 events from March 21, 2023, to March 22, 2023. The results are sorted by product\_name. A color gradient bar is positioned between the search bar and the results table.

A large callout box highlights the keyboard shortcut for expanding macros:

**⌘ command + shift + E**

**OR**

**ctrl + shift + E**

A pink arrow points from the bottom of the keyboard shortcut diagram down towards the expanded search string.

A modal window titled "Expanded Search String" displays the following SPL command:

```
index=main sourcetype="access_combined" action=purchase  
| iplocation clientip  
| rename Country as country  
| lookup geo_attr_countries.csv country  
| search region_un=Europe  
| stats sum(product_price) as total_sales by product_name  
| eval total_sales=("$" + tostring(round(total_sales,2),"commas"))
```

At the bottom of the modal, there are "Cancel" and "Open as new search" buttons.

# SPL#7 > **eventstats** and **streamstats** Search

splunk>





## Use Case Example

Thinking beyond stats command

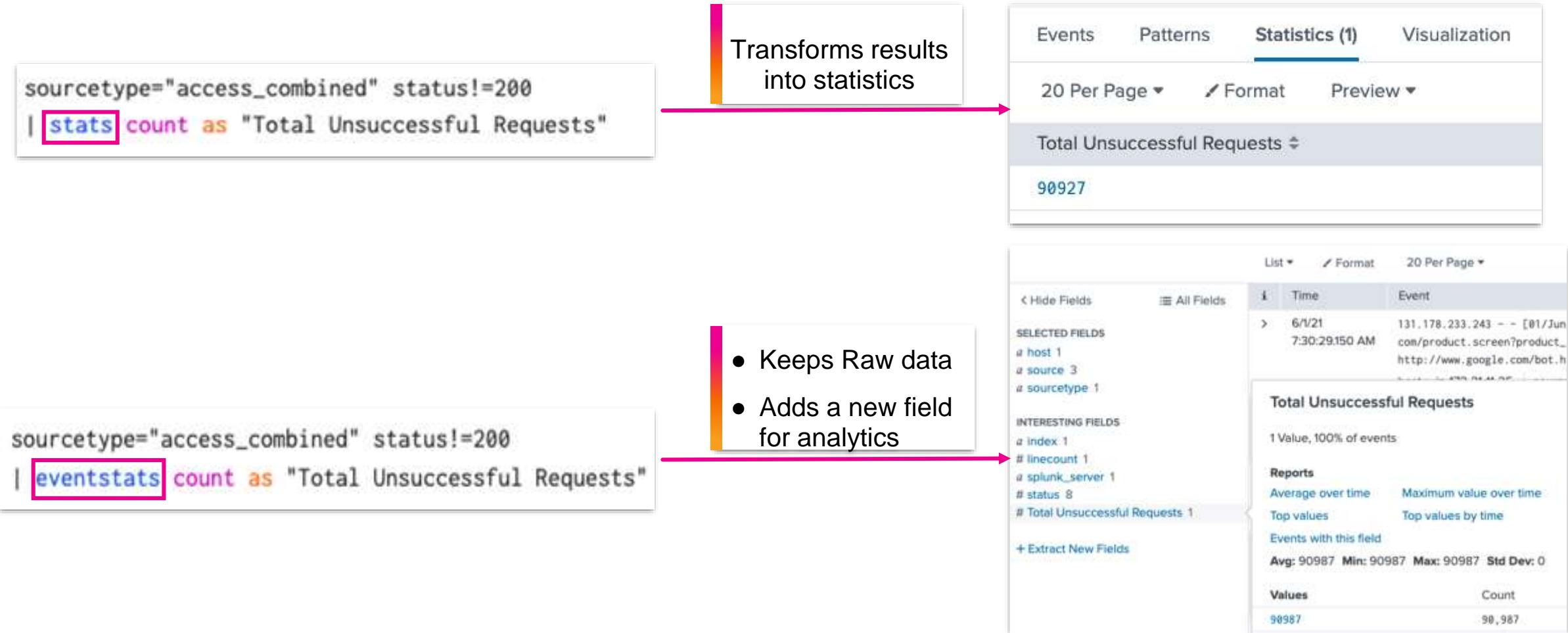
**stats may not always be helpful**

How do you start solving such a use case?

**How many transactions did it take to increase the total revenue from \$500 to \$1000 and what were total number of successful transactions?**

# stats is a transforming command

While eventstats and streamstats add new fields to search while retaining raw data



# Beyond stats: eventstats and streamstats

Adding Aggregate results to result set for additional calculations

## | eventstats

Generates **summary statistics** from fields in **all your events** and saves those statistics in a **new field**

## | streamstats

Adds ~~cumulative summary statistics~~ to all search results in a streaming manner

<b>stats command</b> (works on all results)	<b>eventstats command</b> (works on all results)	<b>streamstats command</b> (works on each event at the time its seen)
Events are transformed into a <b>table of aggregated search results</b>	Aggregations are placed into a <b>new field that is added to each of the events</b> in your output	<b>Calculates statistics for each event</b> at the time the event is seen
You <b>can only</b> use the fields in your aggregated results in <b>subsequent commands in the search</b>	You <b>can use</b> the fields in your events in <b>subsequent commands in your search</b> , because the <b>events have not been transformed</b>	

# Statistics functions available with eventstats and streamstats

Similar to stats – more complex functions can be used

Type of function	Supported functions and syntax				
Aggregate functions	<code>avg()</code> <code>count()</code> <code>distinct_count()</code> <code>estdc()</code> <code>estdc_error()</code>	<code>exactperc&lt;int&gt;()</code> <code>max()</code> <code>median()</code> <code>min()</code> <code>mode()</code>	<code>perc&lt;int&gt;()</code> <code>range()</code> <code>stdev()</code> <code>stdevp()</code>	<code>sum()</code> <code>sumsq()</code> <code>upperperc&lt;int&gt;()</code> <code>var()</code> <code>varp()</code>	
Event order functions	<code>earliest()</code>	<code>first()</code>	<code>last()</code>	<code>latest()</code>	
Multivalue stats and chart functions	<code>list(X)</code>	<code>values(X)</code>			

Learn more:

[https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eventstats#Stats\\_function\\_options](https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eventstats#Stats_function_options)

splunk>

# SPL#7 > eventstats Search

Search for failed purchase transactions over the last 24 hours

**Solution:** sourcetype="access\_combined" status!=200  
| eventstats count as "Total Unsuccessful Requests"

New field created with total count of unsuccessful requests

The screenshot shows a Splunk search interface. On the left, a sidebar lists various fields: a referer 100+, a referer\_domain 1, a req\_time 100+, a splunk\_server 1, # status 8, # timeendpos 8, # timestamppos 8, # Total Unsuccessful Requests 1, a uri 100+, a uri\_path 3, a uri\_query 100+, a user 1, a useragent 22, and # version 1. An orange arrow points from the text "New field created with total count of unsuccessful requests" to the "# Total Unsuccessful Requests 1" field in the sidebar. To the right, a detailed view of the "# Total Unsuccessful Requests" field is shown. It is labeled "Total Unsuccessful Requests" and is described as "1 Value, 100% of events". There are three report options: "Average over time", "Maximum value over time", and "Minimum value over time". Below these are "Top values", "Top values by time", and "Rare values". A section titled "Events with this field" displays statistics: Avg: 23607, Min: 23607, Max: 23607, and Std Dev: 0. A table at the bottom shows the single value 23607 with a count of 23,607 and 100%.

Values	Count	%
23607	23,607	100%

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eventstats>

# SPL#7 > streamstats Search

Search for failed purchase transactions over the last 24 hours

**Solution:** sourcetype="access\_combined" status!=200  
| streamstats count as "Total Unsuccessful Requests"

New field created in streaming manner >

Adding the count as it finds a new event with status!=200 and adds count to event

The screenshot shows a search interface with a histogram of request counts and a summary table for total unsuccessful requests.

**Histogram (Left):**

- a method 2
- # other 100+
- a product\_id 10
- a product\_name 10
- # product\_price 7
- a punct 9
- a referer 100+
- a referer\_domain 1
- a req\_time 100+
- a splunk\_server 1
- # status 8
- # timeendpos 8
- # timestamppos 8
- # Total Unsuccessful Requests 100+ (highlighted with a pink border)
- a uri 100+
- a uri\_path 3

**Total Unsuccessful Requests (Right):**

Total Unsuccessful Requests		
>100 Values, 100% of events		
Selected Yes No		
Average over time	Maximum value over time	Minimum value over time
Top values	Top values by time	Rare values
Events with this field		
Avg: 11959 Min: 1 Max: 23917 Std Dev: 6904.387530162734		
Top 10 Values	Count	%
1	1	0.004%
10	1	0.004%

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Streamstats>

splunk>

# SPL#8 > Percentage of Total Revenue

splunk>



# SPL#8 > Percentage of Total Revenue

Percentage of total revenue generated by “Zombie Survival Guide” in the last 4 hours

product_name	per_prod_rev	total_rev	rev_pct
Zombie Survival Guide	64520.82	954817.07	6.757401

**Solution:**

```
sourcetype="access_combined" product_id!="" earliest=-4h
| eventstats sum(product_price) as total_rev
| eventstats sum(product_price) as rev_per_prod by product_id
| eval percent_revenue=rev_per_prod/total_rev*100
| where product_name="Zombie Survival Guide"
| stats values(rev_per_prod) as per_prod_rev, values(total_rev) as
total_rev, values(percent_revenue) as rev_pct by product_name
```

# SPL#8 > Challenge Task

What search should you run to show the table below?

Product Name	Percentage Revenue
Zombie Survival Guide	6.77 %

# SPL#8 > Challenge Task Solution

## Solution one:

```
sourcetype=access_combined product_id!=""
| eventstats sum(product_price) as total_rev
| eventstats sum(product_price) as rev_per_prod by product_id
| eval percent_revenue=rev_per_prod/total_rev*100
| where product_name="Zombie Survival Guide"
| dedup product_name, percent_revenue
| table product_name, percent_revenue
```

## Solution two:

```
sourcetype=access_combined product_id!=""
| eventstats sum(product_price) as total_rev
| eventstats sum(product_price) as rev_per_prod by product_id
| eval percent_revenue=rev_per_prod/total_rev*100
| stats values(rev_per_prod) as per_prod_rev, values(total_rev) as total_rev, values(percent_revenue) as rev_pct by product_name
| eval rev_pct=round(rev_pct,2)." %"
| where product_name="Zombie Survival Guide"
| rename rev_pct as "Percentage Revenue", product_name as "Product Name"
| table "Product Name", "Percentage Revenue"
```

# SPL#9 > Popularity Ranking

splunk>



# SPL#9 > Popularity Ranking

Popularity ranking of each product within its category in the last 24 hours

**Solution:**

```
sourcetype="access_combined" action=purchase earliest=-24h  
| stats count by product_name, category  
| sort - count  
| streamstats count as rank by category  
| stats list(rank) as "Category Rank", list(product_name) as "Product Name"  
by category
```

category	Category Rank	Product Name
Books	1	Mad Comics- Bronze Man
	2	Zombie Survival Guide
	3	Mad Comics- Batguy
	4	Mad Comics- Flyman
Clothing	1	Batguy Watch
	2	Costume- ManHawk
	3	Batguy Slippers
Gifts	1	Waterproof Scratch and Sniff
	2	Pony Potpourri
	3	Double Fudge Sundae

# SPL#10 > Using **streamstats** and **eventstats**

stats may not be always helpful



## Task 3

How many transactions did it take to increase the **total revenue from \$500 to \$1000** and what was the **total number of successful transactions**?

### Use Case Breakdown:

1. Find **successful purchase** events
2. Calculate the total revenue from the point in time where **it was \$500** and then **increased to \$1000**
3. Calculate the **total number of events** it took to go from **\$500 to \$1000**
4. Calculate the **number of total transactions**



# Break ?

splunk>

# SPL#11 > *spath* as an SPL Command

splunk>



# Extract Specific Information

## Default > Splunk Extracts JSON

Directly in SPL

| **spath** [**input**=<field>] [**output**=<field>]  
**path**=<datapath>

**spath**  
command

As function within  
**eval** command

| **eval** A=**spath**(X,Y)

# SPL#11 > spath as an SPL command

Use spath command directly within SPL

## Syntax

```
| spath [input=<field>] [output=<field>] [path=<datapath> | <datapath>]
```

Field to read & extract  
values from

Output written  
to this field

Location path to the value needed  
for extraction

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

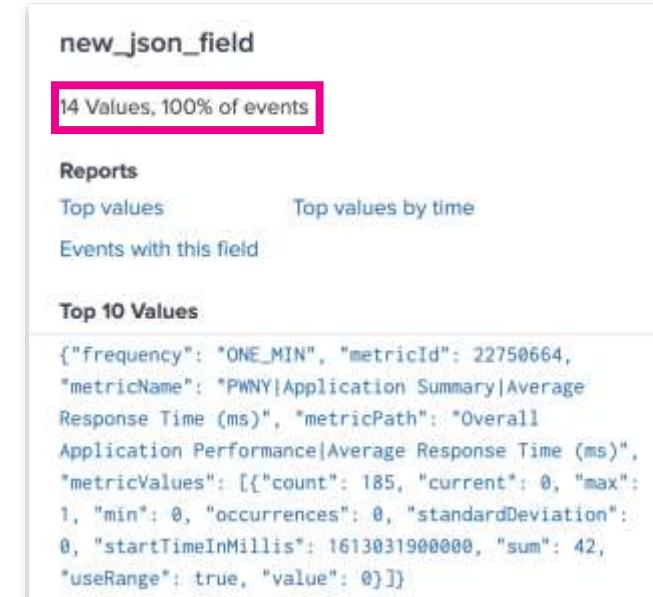
# SPL#11 > spath as an SPL command

## Use spath command directly within SPL

**Solution:** index=sample\_data application\_performance earliest=0  
| spath input=\_raw output=new\_json\_field path=application\_performance{}

Explore the newly created field  
**“new\_json\_field”**

```
# Errors per Minute 1
# Exceptions per Minute 1
# HTTP Error Codes per Minute 1
a index 1
# Infrastructure Errors per Minute 1
# linecount 1
a new_json_field 14
# Normal Average Response Time (ms)
1
a punct 1
```



**new\_json\_field**

14 Values, 100% of events

Reports

Top values      Top values by time

Events with this field

Top 10 Values

```
{"frequency": "ONE_MIN", "metricId": 22750664, "metricName": "PWNY|Application Summary|Average Response Time (ms)", "metricPath": "Overall Application Performance|Average Response Time (ms)", "metricValues": [{"count": 185, "current": 0, "max": 1, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 42, "useRange": true, "value": 0}]}]
```

- Nested **JSON Array** under **application\_performance** is extracted & added into **new\_json\_field**.
- **Note:** There are **14 values** extracted. We have **2 events** with **7 values** each embedded within **application\_performance**.

# SPL#12 > *spath* as an eval Function

splunk>



# Extract Specific Information

## Default > Splunk Extracts JSON

Directly in SPL

| spath [input=<field>] [output=<field>]  
path=<datapath>

spath  
command

As function within  
eval command

| eval A=spath(X,Y)

# SPL#12 > spath as an eval function

Use **spath** function directly within the **eval** command

## Syntax

```
| eval A=spath(X,Y)
```

From structured data type(**X**) in **XML** or **JSON** format; extract value at **Y** and assign to **A**

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# SPL#12 > spath as an eval function

## SPL#11 > Search 1 of 2 over All Time

**Solution:** `index=sample_data application_performance earliest=0  
| eval new_metricName=spath(_raw,"application_performance{}.metricName")`

```
# linecount 1
a new_metricName 7
# Normal Average Response Time (ms)
1
a punct 1
```

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# SPL#12 > spath as an eval function

## SPL#11 > Search 2 of 2 over All Time

**Solution:** index=sample\_data application\_performance earliest=0  
| eval new\_metricName=spath(\_raw,"application\_performance{4}.metricName")  
| eval new\_metricValue=spath(\_raw,"application\_performance{4}.metricValues{}.current")

# linecount 1  
a new\_metricName 1  
# new\_metricValue 1  
# Normal Average Response Time (ms)  
1  
a punct 1

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# Validate if a new field was created

splunk>



# Validate if a new field was created

Explore the output field(s):

**Solution:**

```
index=sample_data application_performance earliest=0
| spath input=_raw output=new_json_field path=application_performance{}
| eval new_metricName=spath(new_json_field,"metricName")
```

The **SPL** does **NOT**  
extract new field  
**“new\_metricName”**

Why not?

```
a index 1
# Infrastructure Errors per Minute 1
# linecount 1
a new_json_field 14
# Normal Average Response Time (ms)
1
a punct 1
```

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# Find out why the new field was not created

splunk>



# Find out why the new field was not created

spath command creates a multi-value field:

**Solution:** index=sample\_data application\_performance earliest=0  
| spath input=\_raw output=new\_json\_field  
path=application\_performance{}  
| table new\_json\_field

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# Find out why the new field was not created

Two events returned with multiple values

The screenshot shows a Splunk search interface with the following details:

- Events (2)**: The first tab is selected, indicated by a pink border.
- Patterns**: Unselected tab.
- Statistics (2)**: Unselected tab.
- Visualization**: Unselected tab.
- 20 Per Page**: The number of events displayed per page.
- Format**: The current display format.
- Preview**: An option to preview the results.
- new\_json\_field =**: The search query or variable name.
- Search Results (2 Events)**:
  - Event 1:
    - frequency": "ONE\_MIN", "metricId": 22750664, "metricName": "PWNY|Application Summary|Average Response Time (ms)", "metricPath": "Overall Application Performance|Average Response Time (ms)", "metricValues": [{"count": 185, "current": 2, "max": 1, "min": 0, "occurrences": 8, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 58, "useRange": true, "value": 0}]}
    - frequency": "ONE\_MIN", "metricId": 22750689, "metricName": "PWNY|Application Summary|Normal Average Response Time (ms)", "metricPath": "Overall Application Performance|Normal Average Response Time (ms)", "metricValues": [{"count": 95, "current": 2, "max": 1, "min": 0, "occurrences": 8, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 11, "useRange": true, "value": 0}]}
  - Event 2:
    - frequency": "ONE\_MIN", "metricId": 22750759, "metricName": "PWNY|Application Summary|HTTP Error Codes per Minute", "metricPath": "Overall Application Performance|HTTP Error Codes per Minute", "metricValues": [{"count": 5, "current": 0, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 0, "useRange": false, "value": 0}]}
    - frequency": "ONE\_MIN", "metricId": 22750758, "metricName": "PWNY|Application Summary|Exceptions per Minute", "metricPath": "Overall Application Performance|Exceptions per Minute", "metricValues": [{"count": 20, "current": 36, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 129, "useRange": false, "value": 0}]}
    - frequency": "ONE\_MIN", "metricId": 22750668, "metricName": "PWNY|Application Summary|Infrastructure Errors per Minute", "metricPath": "Overall Application Performance|Infrastructure Errors per Minute", "metricValues": [{"count": 15, "current": 18, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 98, "useRange": false, "value": 18}]}
    - frequency": "ONE\_MIN", "metricId": 22750687, "metricName": "PWNY|Application Summary|Errors per Minute", "metricPath": "Overall Application Performance|Errors per Minute", "metricValues": [{"count": 5, "current": 18, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 90, "useRange": false, "value": 18}]}
    - frequency": "ONE\_MIN", "metricId": 22750665, "metricName": "PWNY|Application Summary|Calls per Minute", "metricPath": "Overall Application Performance|Calls per Minute", "metricValues": [{"count": 60, "current": 37, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 185, "useRange": false, "value": 37}]}
    - frequency": "ONE\_MIN", "metricId": 22750664, "metricName": "PWNY|Application Summary|Average Response Time (ms)", "metricPath": "Overall Application Performance|Average Response Time (ms)", "metricValues": [{"count": 185, "current": 0, "max": 1, "min": 0, "occurrences": 8, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 42, "useRange": true, "value": 0}]}
    - frequency": "ONE\_MIN", "metricId": 22750689, "metricName": "PWNY|Application Summary|Normal Average Response Time (ms)", "metricPath": "Overall Application Performance|Normal Average Response Time (ms)", "metricValues": [{"count": 95, "current": 0, "max": 1, "min": 0, "occurrences": 8, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 10, "useRange": true, "value": 0}]}
    - frequency": "ONE\_MIN", "metricId": 22750759, "metricName": "PWNY|Application Summary|HTTP Error Codes per Minute", "metricPath": "Overall Application Performance|HTTP Error Codes per Minute", "metricValues": [{"count": 5, "current": 0, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 0, "useRange": false, "value": 0}]}
    - frequency": "ONE\_MIN", "metricId": 22750758, "metricName": "PWNY|Application Summary|Exceptions per Minute", "metricPath": "Overall Application Performance|Exceptions per Minute", "metricValues": [{"count": 20, "current": 36, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 128, "useRange": false, "value": 0}]}
    - frequency": "ONE\_MIN", "metricId": 22750668, "metricName": "PWNY|Application Summary|Infrastructure Errors per Minute", "metricPath": "Overall Application Performance|Infrastructure Errors per Minute", "metricValues": [{"count": 15, "current": 18, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 98, "useRange": false, "value": 18}]}
    - frequency": "ONE\_MIN", "metricId": 22750687, "metricName": "PWNY|Application Summary|Errors per Minute", "metricPath": "Overall Application Performance|Errors per Minute", "metricValues": [{"count": 5, "current": 18, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 90, "useRange": false, "value": 18}]}
    - frequency": "ONE\_MIN", "metricId": 22750665, "metricName": "PWNY|Application Summary|Calls per Minute", "metricPath": "Overall Application Performance|Calls per Minute", "metricValues": [{"count": 60, "current": 37, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 185, "useRange": false, "value": 37}]}

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# SPL#13 >

## The **mvexpand** Command

splunk>



# SPL#13 > The mvexpand command

Expand and extract the field

**Solution:**

```
index=sample_data application_performance earliest=0  
| spath input=_raw output=new_json_field  
path=application_performance{}  
| mvexpand new_json_field  
| eval new_metricName=spath(new_json_field,"metricName")
```

Did the **SPL**  
extract the  
**new\_metricName**  
field?

```
# linecount 1  
a new_json_field 14  
a new_metricName 7  
# Normal Average Response Time (ms)  
1  
a punct 1
```

# SPL#14 > Introducing the **rex** Command

Dealing with Improperly formatted JSON data

splunk>



# Field Extraction and Regex Refresher

DevOps use-case from Splunk4Rookies: Extracting the **platform** out of the **raw data**

- After extraction, Splunk provides the **Regular Expression** used to provide the instructed field
- SPL#14** shows how to incorporate the **rex** command (**regular expression**) within the **SPL**

Save

Name the extraction and set permissions.

Extractions Name EXTRACT- platform

Owner admin

App splunk4rookies

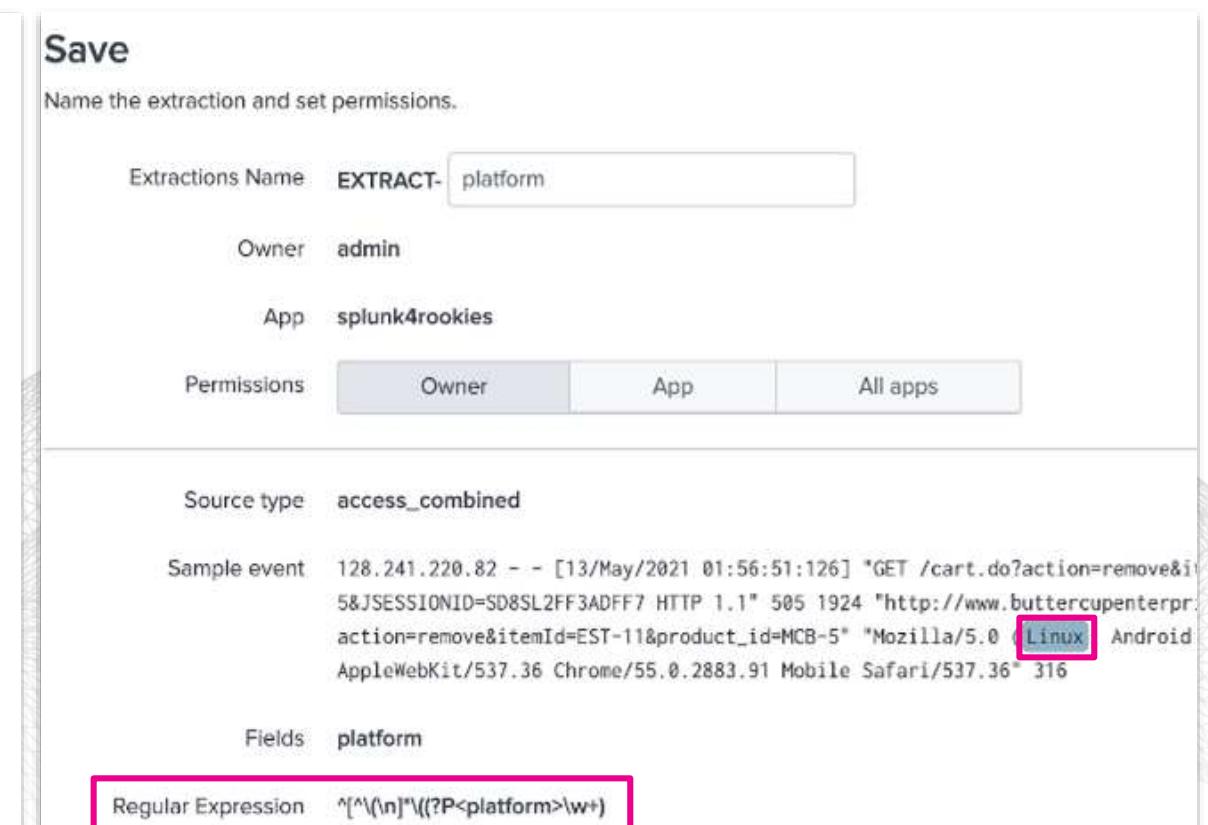
Permissions Owner App All apps

Source type access\_combined

Sample event 128.241.220.82 -- [13/May/2021 01:56:51:126] "GET /cart.do?action=remove&itemID=SD8SL2FF3ADFF7 HTTP/1.1" 505 1924 "http://www.buttercupenterpr... action=remove&itemID=EST-11&product\_id=MGB-5" "Mozilla/5.0 (Linux) AppleWebKit/537.36 Chrome/55.0.2883.91 Mobile Safari/537.36" 316

Fields platform

Regular Expression ^[^(\n)]\((?P<platform>\w+)



# Working with improperly formatted key/value in JSON

**Use Case:** From health rule violation events, extract “Memory Heap used % value”

Use the search string  
**“healthruleViolations”**  
to get relevant events.

```
Event
{
  account_name: buttercup-dev
  appd_app_uuid: PWNY:26822:buttercup-dev.saas.appdynamics.com
  application_id: 26822
  application_name: PWNY
  healthruleViolations: [
    {
      affectedEntityDefinition: [
        {
          deepLinkUrl: https://buttercup-dev.saas.appdynamics.com/#location=APP INCIDENT_MODAL&incident=3313892&application=26822
          description: AppDynamics has detected a problem with Node <b>PWNY-Apps-Weblogic-server317-servicesAdminServer</b>. <br><b>JVM Heap utilization is too high</b>
          started violating and is now <b>warning</b>. <br>All of the following conditions were found to be violating<br>For Node <b>PWNY-Apps-Weblogic-server317-
          servicesAdminServer</b>:<br>1) JVM|Memory:Heap|Used % Condition<br><b>Used %'s</b> value <b>76.00</b> was <b>greater than</b> the threshold <b>75.00</b> for the last
          <b>30</b> minutes.<br>
          detectedTimeInMillis: 0
          endTimeInMillis: 0
          id: 3313892
          incidentStatus: OPEN
          name: JVM Heap utilization is too high
          severity: WARNING
          startTimeInMillis: 1613029555001
          triggeredEntityDefinition: [
            {}
          ]
        }
      ]
    }
  ]
}
```

The required value is not extracted and is  
within the “**description**” field under  
“**healthruleViolations**” JSON object

# SPL#14 > Introducing the `rex` command

Maximum value of **JVM Heap Memory Usage** which triggered “**healthrule\_violations**”

## Rex Command Syntax:

```
| rex field=<field> "<regex-expression>"
```

Field on which the  
regex will be applied

Regular expression

**Solution:** index=sample\_data healthruleViolations earliest=0

```
| rex field=healthruleViolations{}.description
```

JSON field which  
contains the value  
we need

```
"Used\s%\s\<\b\>\svalue\s\<b\>(?\<mem_used_value>(.+?))\<"
```

Name of new field

# SPL#14 > Introducing the `rex` command continued

**Solution:**

```
index=sample_data healthruleViolations earliest=0
| rex field=healthruleViolations{}.description
"Used\s%\s</b>\svalue\s<b>(?<mem_used_value>(.+?))<"
```

The screenshot shows the Splunk search builder interface. On the left, the search command is displayed:

```
index=sample_data healthruleViolations earliest=0
| rex field=healthruleViolations{}.description
"Used\s%\s</b>\svalue\s<b>(?<mem_used_value>(.+?))<"
```

A red box highlights the `mem_used_value` field name in the command. A callout arrow points from this highlighted text to a tooltip on the right:

Name of newly extracted field

On the right, the results pane shows the extracted field `mem_used_value`. It indicates there are 2 values from 100% of events. A "Selected" checkbox is checked, with "Yes" and "No" buttons nearby. Below this, there are tabs for "Reports", "Top values", "Top values by time", and "Rare values". The "Top values" tab is selected, showing the following table:

Values	Count	%
76.00	4	200%
81.00	1	50%

# SPL#15 > Using **rex** command with SED mode

splunk>



# SPL#15 > Using rex command with SED mode

Another way to look at the maximum value of JVM Heap Memory Usage

Using `rex` in sed  
(stream editor)  
mode

```
index=sample_data healthruleViolations earliest=0
| rex mode=sed "s/Used\s%\s</b>\svalue\s<b>/ValueWeNeed=/g"
| rex field=_raw "ValueWeNeed\=(?<mem_used_value>(.+?))\<"
```

i	Time	Event
>	2/1/21 7:45:55.001 AM	<pre>{   account_name: buttercup-dev   appd_app_uuid: PWNY:26822:buttercup-dev.saas.appdynamics.com   application_id: 26822   application_name: PWNY   healthruleViolations: [     {       affectedEntityDefinition: [         {           deepLinkUrl: https://buttercup-dev.saas.appdynamics.com/#location=APP INCIDENT DETAIL_MODAL&amp;incident=3313892&amp;application=26822           description: AppDynamics has detected a problem with Node &lt;b&gt;PWNY-Apps-Weblogic-server317-servicesAdminServer&lt;/b&gt;. &lt;br&gt;&lt;b&gt;JVM Heap utilization is too high&lt;/b&gt; started violating and is now &lt;b&gt;warning&lt;/b&gt;. &lt;br&gt;All of the following conditions were found to be violating &lt;br&gt;For Node &lt;b&gt;PWNY-Apps-Weblogic-server317-servicesAdminServer&lt;/b&gt;; &lt;br&gt;1) JVM Memory:Heap Used % Condition&lt;br&gt;&lt;b&gt;ValueWeNeed=76.00&lt;/b&gt; was &lt;b&gt;greater than&lt;/b&gt; the threshold &lt;b&gt;75.00&lt;/b&gt; for the last &lt;b&gt;30&lt;/b&gt; minutes. &lt;br&gt;     }   ] }</pre>



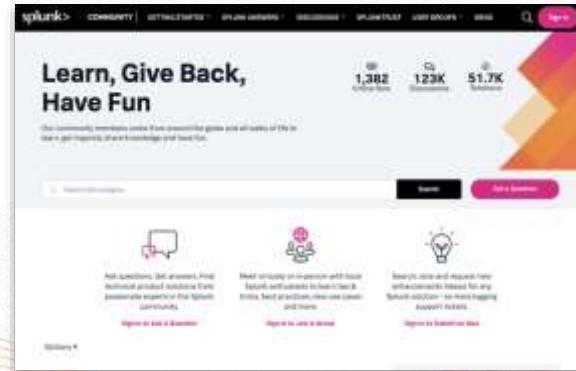
# Splunk Resources

Where to go after  
today's workshop

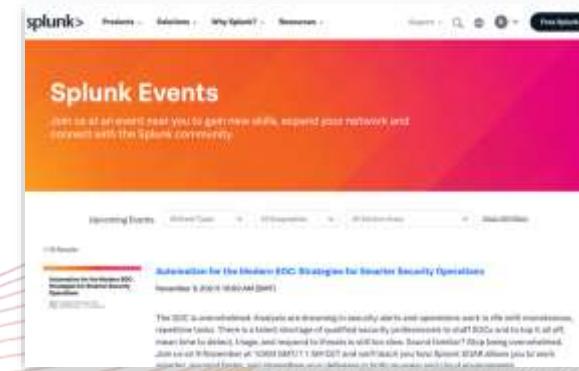
splunk>

# Splunk's Thriving Community

## Splunk Community



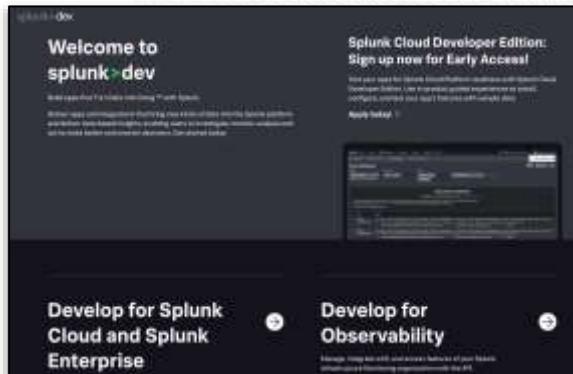
## Splunk Events



## Documentation



## Developer Resources



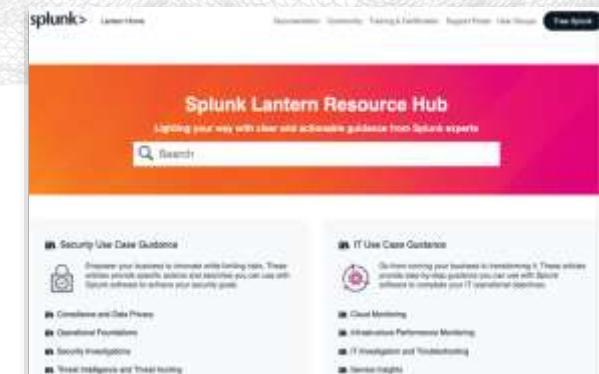
## Splunkbase



## Education



## Splunk Lantern



# Splunk Education

- Using Fields

[https://www.splunk.com/en\\_us/pdfs/training/using-fields-course-description.pdf](https://www.splunk.com/en_us/pdfs/training/using-fields-course-description.pdf)

- Search Under the Hood

[https://www.splunk.com/en\\_us/pdfs/training/search-under-the-hood-course-description.pdf](https://www.splunk.com/en_us/pdfs/training/search-under-the-hood-course-description.pdf)

- Search Optimization

[https://www.splunk.com/en\\_us/pdfs/training/search-optimization-course-description.pdf](https://www.splunk.com/en_us/pdfs/training/search-optimization-course-description.pdf)

- Multivalue Fields

[https://www.splunk.com/en\\_us/pdfs/training/multivalue-fields-course-description.pdf](https://www.splunk.com/en_us/pdfs/training/multivalue-fields-course-description.pdf)

# Deeper Dive (Self Study)

- Tips & Tricks for Optimization

- <https://docs.splunk.com/Documentation/Splunk/latest/Search/Quicktipsforoptimization>

- Mastering the Speed of Search

- Recording: <https://conf.splunk.com/files/2019/recording/FN1407.mp4>
  - Slides: <https://conf.splunk.com/files/2019/slides/FN1407.pdf>

- JSON Functions

- <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/JSONFunctions>

- Joining datasets/emulating SQL behavior

- <https://docs.splunk.com/Documentation/SplunkCloud/latest/SearchReference/SQLtoSplunk>
  - <https://conf.splunk.com/files/2019/slides/FNC2751.pdf>

# Thank You

