Multivalue Fields

Document Usage Guidelines

- Should be used only for enrolled students
- Not meant to be a self-paced document, an instructor is needed
- Lab Exercise slides reference the hands-on lab exercise guide
- Do not distribute

Course Goals

- Define multivalue fields
- Evaluate multivalue fields
- Manipulate multivalue fields
- Analyze multivalue fields
- Format multivalue fields
- Configure extractions of multivalue fields

Course Outline

- What are Multivalue Fields?
- Create Multivalue Fields
- Evaluate Multivalue Fields
- Analyze Multivalue Data

What are Multivalue Fields?

Topic Objectives

- Define multivalue fields
- Define self-describing data
- Understand how JSON data is handled in Splunk
- Use the **`spath`** command to interpret self-describing data
- Manipulate multivalue fields with **`mvzip`** and **`mvexpand`**
- Convert single-value fields to multivalue fields with:
  - **`makemv`** command
  - **`stats list`** and **`values`** functions
  - **`transaction`** command

Define Multivalue Fields

- Multivalue fields are fields that hold more than one value:
  - Email logs containing To, Cc, and Bcc values
  - AWS configuration records
  - JSON and XML arrays
- Multivalue functions and commands provide ways to count, evaluate, modify, create, combine, sort, and split multivalue fields
  - Typically contain a **`mv`** prefix
  - Exceptions discussed in this module:
    - **`split`**, **`list`**, and **`values`** functions
    - **`transaction`** and **`multikv`** commands

Define Self-Describing Data

- JSON and XML data are types of "self-describing data" where the structure is embedded in the data itself
- Comprised of metadata which may include:
  - Properties/elements/attributes
  - Data types/items
  - Compression/encoding scheme
  - Other info

JSON and XML Examples

JSON

```
{"menu": {
      "id": "file",
      "value": "File",
      "popup": {
            "menuitem": [
                  {"value": "New", "onclick":
"CreateNewDoc()"},
                  {"value": "Open", "onclick": "OpenDoc()"},
                  {"value": "Close", "onclick": "CloseDoc()"}
            ]
      }
}}
```

XML

```
<menu id="file" value="File">
      <popup>
            <menuitem value="New" onclick="CreateNewDoc()" />
            <menuitem value="Open" onclick="OpenDoc()" />
            <menuitem value="Close" onclick="CloseDoc()" />
      </popup>
</menu>
```

- Splunk extracts fields from JSON and XML based on the formatting
- Both consist of:
  - Collections of name/value pairs
  - Ordered lists of values (arrays)

Splunk and JSON Data

- Splunk preserves JSON structure when displaying data from JSON-formatted log files
- In JSON data:
  - { } indicates an object, which is a grouping of field-value pairs
  - [ ] indicates an array of objects
- When extracted by Splunk, the array contents become multivalue fields

```
index=systems sourcetype=system_info
```

i	Time	Event
>	6/21/22 1:59:10.000 PM	{ [-] CPU: { [+] } CPU_CORES: [[+]] RAM: { [+] } ROOT_USERS: [[+]] SYSTEM: [[+]] asctime: 2022-06-21T19:59:10+00:00 created: 1655841550.118906 levelname: INFO message: system_info name: SystemLogger

JSON event data

Automatically Extracting JSON Data

```
index=systems sourcetype=system_info
```

i	Time	Event
>	6/21/22 1:59:10.000 PM	{ [-] CPU: { [+] } CPU_CORES: [[-] { [-] core: core_number_1 core_percent_used: 0 guest: 0 guest_nice: 0 idle: 100 iowait: 0 irq: 0 nice: 0 softirq: 0 steal: 0 system: 0 user: 0 }] RAM: { [+] } ROOT_USERS: [[-] Admin Duke Zed Camilian] SYSTEM: [[+]

The object array **CPU_CORES[]** has multiple objects with the same fields. When *extracted*, these fields are represented as multivalue fields.

```
CPU_CORES: [ [-]
  { [-]
    core: core_number_1
    core_percent_used: 0
    guest: 0
    guest_nice: 0
    idle: 100
    iowait: 0
    irq: 0
    nice: 0
    softirq: 0
    steal: 0
    system: 0
    user: 0
```

- a CPU_CORES{}.core 16
- a CPU_CORES{}.core_percent_used 11
- a CPU_CORES{}.guest 1
- a CPU_CORES{}.guest_nice 1
- a CPU_CORES{}.idle 11
- a CPU_CORES{}.iowait 2
- a CPU_CORES{}.irq 1
- a CPU_CORES{}.nice 3
- a CPU_CORES{}.softirq 2
- a CPU_CORES{}.steal 2
- a CPU_CORES{}.system 4
- a CPU_CORES{}.user 8

Interesting Fields List

When *ingested*, the array **ROOT_USERS[]** becomes a multivalue field with 4 values

```
ROOT_USERS: [ [-]
  Duke
  Camilian
  Admin
  Cardozo
```

JSON Field Name with {} Caveat

- Functions and commands process {} within a field name as special characters
- For consistent results, use the **rename** command to remove the {} entirely from the field name

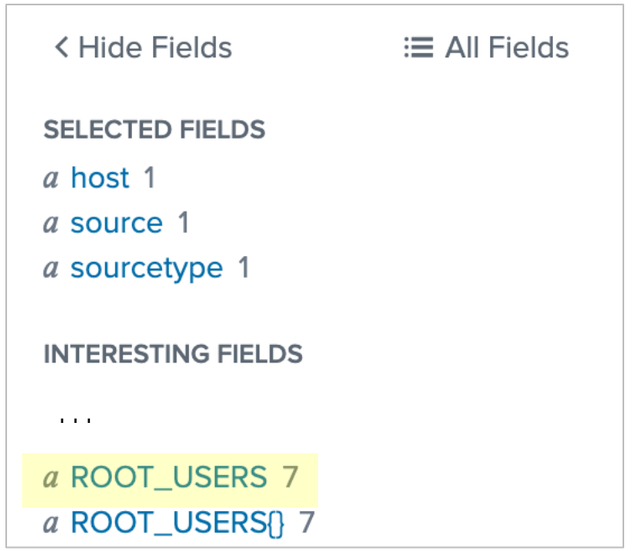
```
index=systems sourcetype=system_info
| rename ROOT_USERS{} as root_users
| eval root_users=mvsort(root_users)
```

JSON Field Name with {} Caveat (cont.)

- Using **`rename`** is recommended because certain commands produce inconsistent results when used with **`{}`**-containing fields
- For example, the **`eval`** statement *seems* to overwrite **`ROOT_USERS{}`**

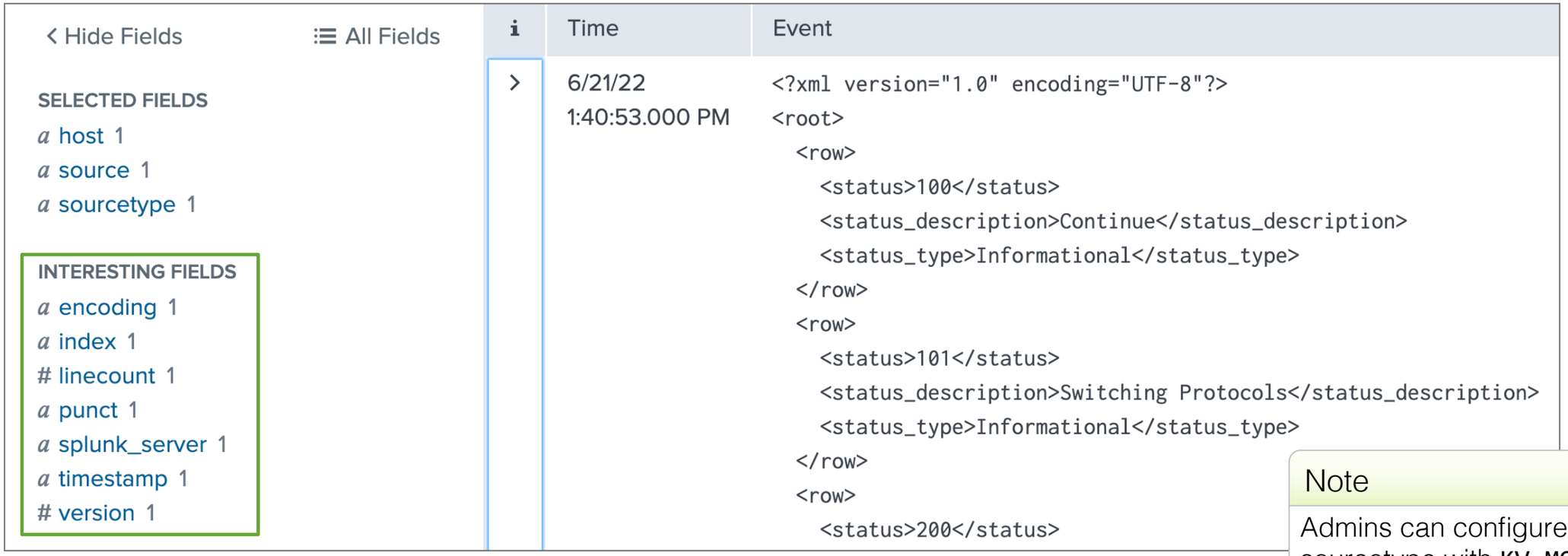
```
index=systems sourcetype=system_info
| eval ROOT_USERS{}=upper('ROOT_USERS{}')
```

  - Instead, it creates a new field, **`ROOT_USERS`** (without the **`{}`**)



Interpreting XML

By default, Splunk doesn't extract fields automatically from XML data



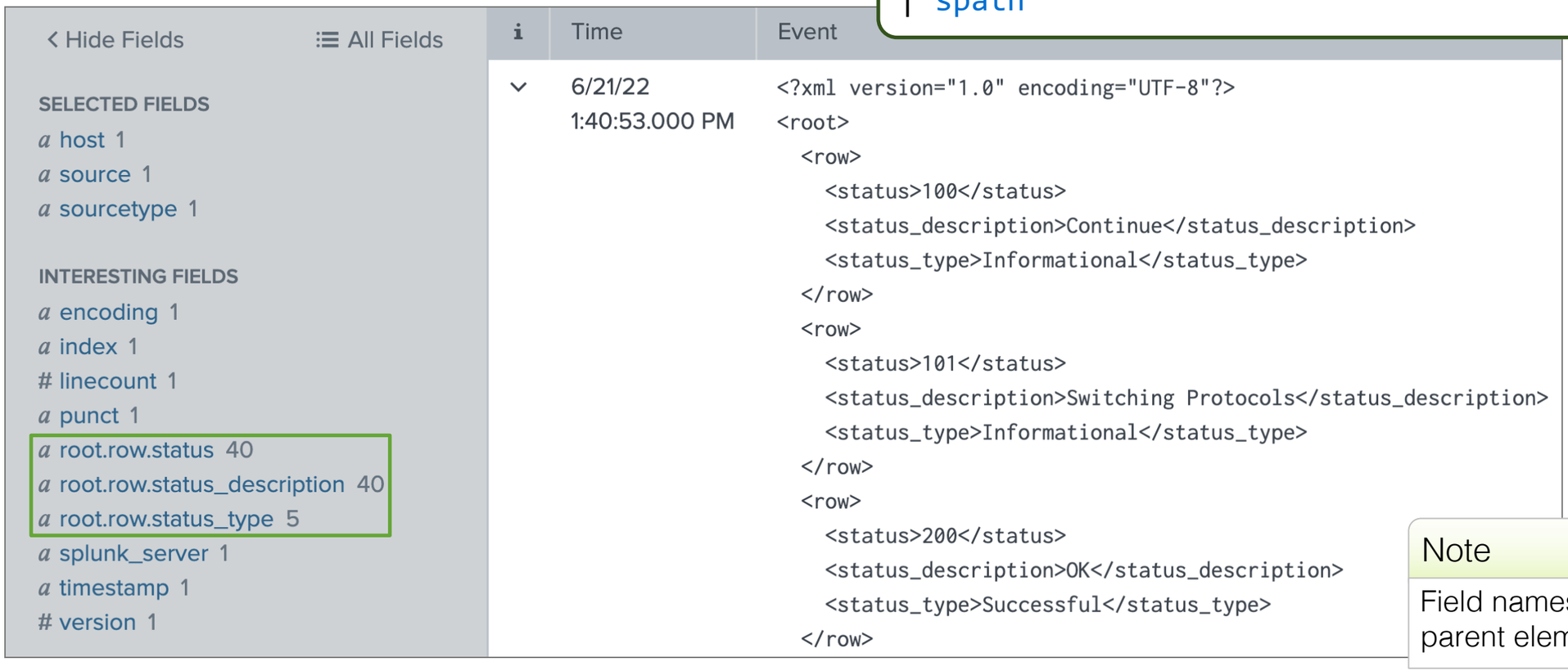
Note

Admins can configure the sourcetype with `KV_MODE=XML` to automatically extract fields at search time.

Use `spath` with XML

The `spath` command interprets the XML structure so you can access the data as Splunk fields

```
index=systems sourcetype=status_definitions
| spath
```



Note

Field names include the parent elements (datapath).

spath Command

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

- Extracts fields from self-describing data (XML and JSON)
- Can be used alone or with optional arguments:
  - **input**: specifies which **<field>** to extract data from (defaults to **_raw**)
  - **output**: data to be extracted is written to this **<field>** (defaults to value of the **path** argument)
  - **path**: the location path to the value you want to extract
    - Valid syntax is **path=<datapath>** or just **<datapath>**
    - By default, extracts all fields from first 5000 characters of **input**

`spath` Command: Location Steps

- The `path` argument can contain one or more location steps, separated by periods
- Each step contains field name and optional index (position) in curly brackets
  - If index is an integer, it specifies the position of data in an array
  - If index is a string preceded by an `@` symbol, it specifies an XML attribute
- Examples:
  - `recordings.album.artist`
  - `entities.hashtags{3}.text`
  - `purchases.book.title{@yearPublished}`

Note

Indexes in JSON and XML are slightly different. In JSON, numbering begins with 0; in XML, it begins with 1.

spath Example 1

Scenario

IT wants to create a table containing status, description, and status type from data in an XML file.

```
index=systems sourcetype=status_definitions
| spath
| table root.*
| rename root.row.* as *
```

status	status_description	status_type
100	Continue	Informational
101	Switching Protocols	Informational
200	OK	Successful
201	Created	Successful
202	Accepted	Successful
203	Non-Authoritative Information	Successful
204	No Content	Successful
205	Reset Content	Successful
206	Partial Content	Successful
300	Multiple Choices	Redirection
301	Moved Permanently	Redirection
302	Found	Redirection
303	See Other	Redirection
304	Not Modified	Redirection

spath Example 2

Use curly brackets without an integer to indicate the index of an entire array

Scenario

The Dev team wants to display a table of popup menu values from data in a JSON file.

```
index=systems sourcetype=menu_json
| spath output=menuItems path=menu.popup.menuitem{}.value
| table menuItems
```

menuItems
New
Open
Close

```
{"menu": {
    "id": "file",
    "value": "File",
    "popup": {
        "menuitem": [
            {"value": "New", "onclick": "CreateNewDoc()"},
            {"value": "Open", "onclick": "OpenDoc()"},
            {"value": "Close", "onclick": "CloseDoc()"}
        ]
    }
}}
```

Note

If you do not use the **output** option, then you should rename the field to remove the {}. Doing so avoids issues that can occur when using field names containing {} with certain commands.

spath Example 3

Use the @ symbol to specify an XML attribute

Scenario

The Docs team wants to extract a table of book publication dates from an XML file.

```
index=systems sourcetype=library_xml
| spath output=publicationDates path=purchases.book.title{@yearPublished}
| table publicationDates
```

```
<purchases>
    <book>
        <title yearPublished=1867>War and Peace</title>
        <author>Leo Tolstoy</author>
    </book>
    <book>
        <title yearPublished=1866>Crime and Punishment</title>
        <author>Fyodor Dostoyevsky</author>
    </book>
    <book>
        <title yearPublished=1877>Anna Karenina</title>
        <author>James Joyce</author>
    </book>
    <book>
        <title yearPublished=1880>The Brothers Karamazov</title>
        <author>Fyodor Dostoyevsky</author>
    </book>
    <book>
        <title yearPublished=1851>Moby Dick</title>
        <author>Herman Melville</author>
    </book>
</purchases>
```

publicationDates
1867
1866
1877
1880
1851

spath Example 4

Even within unstructured data, a field can have a valid self-describing data structure that can be extracted using **spath**

The **system_info** field has a valid JSON structure

`spath` Example 4 (cont.)

To extract data from the **`system_info`** field, invoke the **`spath`** command with the **`input`** option

```
index=systems sourcetype=server_log
| spath input=system_info
```

Use the `eval` Command `spath` Function

- As an alternative to the `spath` command, you can also use the `spath` function of the `eval` command
- `spath(X,Y)` where:

  `X`: input source field

  `Y`: XML or JSON formatted location path to the value you want to extract from `X`

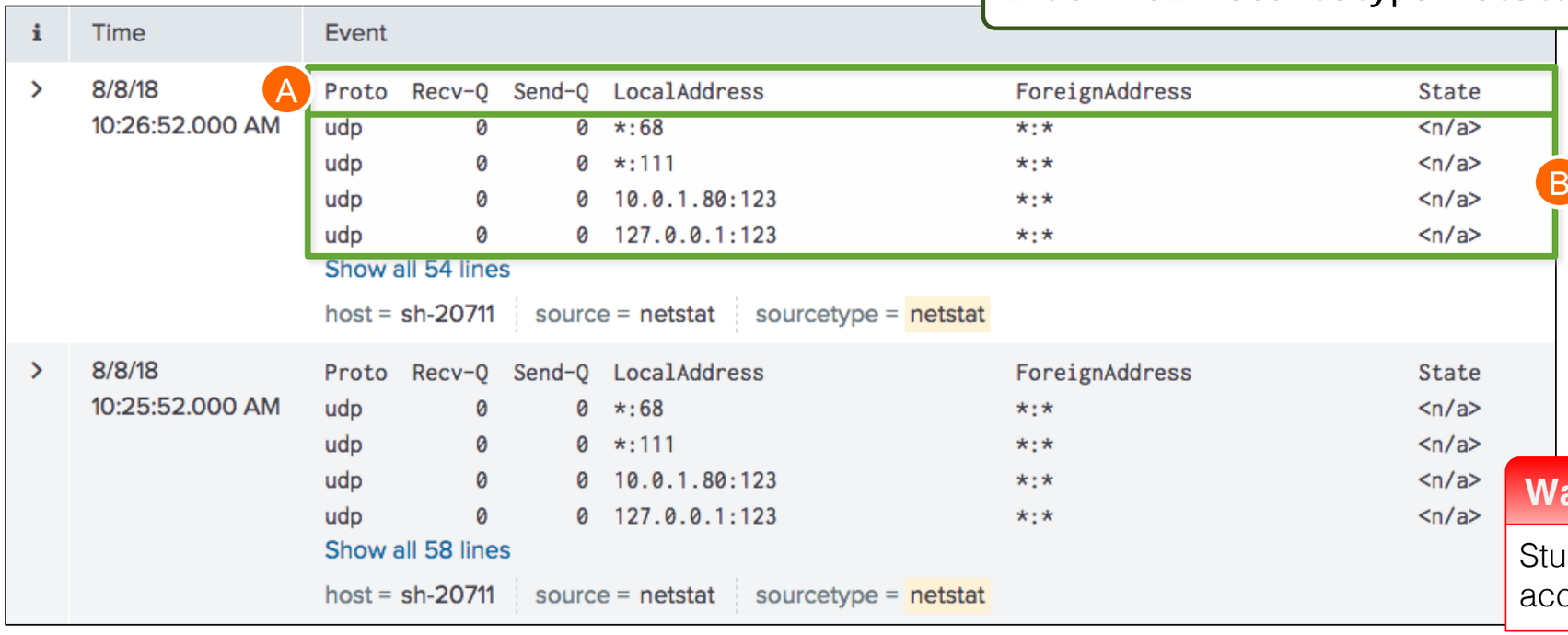
```
index=systems sourcetype=status_definitions
| eval description = upper(spath(_raw, "root.row.status_description"))
| table description
```

description
CONTINUE
SWITCHING PROTOCOLS
OK
CREATED
ACCEPTED
NON-AUTHORITATIVE INFORMATION
NO CONTENT
RESET CONTENT
PARTIAL CONTENT
MULTIPLE CHOICES
MOVED PERMANENTLY

Table-Formatted Event Example

- Some data types are formatted as large single events in a table
- Each event contains titles with tabular values
  - Field names are derived from the title row A
  - All other rows represent values B

```
index=main sourcetype=netstat
```



Warning

Students do not have access to the **main** index.

Extract Fields Using the `multikv` Command

- For table-formatted events, **`multikv`** creates an event for each row
- Field names derived from header row of each event

```
index=main sourcetype=netstat
```

i	Time	Event
>	8/8/18 10:26:52.000 AM	Proto Recv-Q Send-Q LocalAddress ForeignAddress State udp 0 0 *:68 *:* <n/a> udp 0 0 *:111 *:* <n/a> udp 0 0 10.0.1.80:123 *:* <n/a> udp 0 0 127.0.0.1:123 *:* <n/a> Show all 54 lines host = sh-20711 source = netstat sourcetype = netstat

```
index=main sourcetype=netstat
| multikv
```

INTERESTING FIELDS

- a eventtype 1
- a ForeignAddress 46
- a index 1
- # linecount 1
- a LocalAddress 100+
- a Proto 2
- a punct 1
- # Recv_Q 1
- # Send_Q 8
- a splunk_server 1
- a State 4
- a tag 5
- a tag::eventtype 5
- a timestamp 1

>	8/8/18 10:41:52.000 AM	udp	0	0 *:68	*:*	
		host = sh-20711	source = netstat	sourcetype = netstat		
>	8/8/18 10:41:52.000 AM	udp	0	0 *:111	*:*	<n/a>
		host = sh-20711	source = netstat	sourcetype = netstat		
>	8/8/18 10:41:52.000 AM	udp	0	0 10.0.1.80:123	*:*	<n/a>
		host = sh-20711	source = netstat	sourcetype = netstat		
>	8/8/18 10:41:52.000 AM	udp	0	0 127.0.0.1:123	*:*	<n/a>
		host = sh-20711	source = netstat	sourcetype = netstat		

Use `multikv`: `fields` and `filter` Options

- Use `fields` option to extract only specified fields
- Use `filter` option to include only table rows containing at least one field value from a specified list of fields

```
index=main sourcetype=netstat
| multikv fields LocalAddress ForeignAddress State filter ESTAB LISTEN
| table LocalAddress, ForeignAddress, State
```

LocalAddress	ForeignAddress	State
*:46449	*:*	LISTEN
*:22	*:*	LISTEN
127.0.0.1:8191	127.0.0.1:38404	ESTAB
127.0.0.1:38404	127.0.0.1:8191	ESTAB
127.0.0.1:38430	127.0.0.1:8191	ESTAB

Note

The arguments supplied to the `filter` option are treated like they are connected using `OR` operators.

Multivalue Function: `list`

```
...| stats list(X) [as <newfield>| by <field>]
```

- A multivalue function that can be used with **`stats`** or **`chart`**
- Lists all field values for a given field, **`X`**
  - Listed fields can be renamed as **`<newfield>`**, otherwise the field name will be **`list(X)`**
  - Values can be grouped by **`<field>`**
  - If more than one value is listed, the result is a multivalue field
- The order of the values reflects the order of events

Multivalue Function: `list` Example

Scenario

Which websites has each employee accessed during the last 60 minutes?

```
index=network sourcetype=cisco_wsa_squid
| stats list(s_hostname) as "Websites visited:"
  by cs_username
```

cs_username	Websites visited:
acurry@buttercupgames.com	- - www.goppo.com www.goppo.com
bgenin@buttercupgames.com	www.hybridarcade.com
cberztiss@buttercupgames.com	- -
cganttchart@buttercupgames.com	-
djohnson@buttercupgames.com	-
dpiazza@buttercupgames.com	-
fullian@buttercupgames.com	- www.espn.go.com www.espn.go.com
gbottazzi@buttercupgames.com	- -
gfacello@buttercupgames.com	www.fftoday.com www.fftoday.com www.fftoday.com www.fftoday.com www.fftoday.com

Multivalue field

Multivalue Function: `values`

```
...| stats values(X) [by <field> | as <newfield>]
```

- A multivalue function that can be used with **`stats`** or **`chart`**
- Syntax is identical to **`list`** function
- Unlike the **`list`** function, the **`values`** function lists all *unique* values of field, **`X`**

Multivalue Functions: `list` vs `values`

```
index=network sourcetype=cisco_wsa_squid
| stats list(s_hostname) as "Websites visited:"
  by cs_username
```

cs_username	Websites visited:
adombrowski@buttercupgames.com	www.ebgames.com www.ebgames.com
apucci@buttercupgames.com	www.ebgames.com www.ebgames.com www.ebgames.com
blu@buttercupgames.com	www.ebgames.com
cberztiss@buttercupgames.com	www.ebgames.com
cmunson@buttercupgames.com	www.collegegrad.com
cquinn@buttercupgames.com	www.ebgames.com www.ebgames.com
djohnson@buttercupgames.com	www.ebgames.com
edutra@buttercupgames.com	www.ebgames.com
ewarwick@buttercupgames.com	www.collegegrad.com www.ebgames.com

```
index=network sourcetype=cisco_wsa_squid
| stats values(s_hostname) as "Websites visited:"
  by cs_username
```

cs_username	Websites visited:
adombrowski@buttercupgames.com	www.ebgames.com
apucci@buttercupgames.com	www.ebgames.com
blu@buttercupgames.com	www.ebgames.com
cberztiss@buttercupgames.com	www.ebgames.com
cmunson@buttercupgames.com	www.collegegrad.com
cquinn@buttercupgames.com	www.ebgames.com
djohnson@buttercupgames.com	www.ebgames.com
edutra@buttercupgames.com	www.ebgames.com
ewarwick@buttercupgames.com	www.collegegrad.com www.ebgames.com

`transaction` Command

When processed by the **`transaction`** command, single-value fields become multivalue fields (if there are several values)

```
index=web sourcetype=access*
| transaction clientip
| table clientip status action
```

clientip	status	action
221.207.229.6	200	addtocart view
118.142.68.222	200 400	addtocart purchase view
12.130.60.4	200	changequantity view
99.61.68.230	200	addtocart changequantity purchase view
58.68.236.98	200	addtocart purchase

What are Multivalue Fields Lab Exercise

Time: 20 minutes

Tasks:

- Extract fields from an XML file using the `spath` command and the `spath` function of the `eval` command
- Use the `spath` command to extract fields from the `system_info` field and use this data to display server performance
- Complete a search with a multivalue `stats` function
- Use a multivalue `stats` function to list all unique users active on the AD/DNS server during the last 4 hours

Create Multivalue Fields

Topic Objectives

- Create multivalue fields with:
  - `split` (multivalue `eval` function)
  - `makemv` (multivalue command)

Multivalue **`eval`** Functions

The **`eval`** functions discussed in this topic (and succeeding topics) can all be used with **`eval`**, **`fieldformat`**, and **`where`** commands…

```
...| eval <field>=mvfunction(…)
```

```
...| where <booleanExpression>
```

The **`booleanExpression`** would contain a **`mvfunction`**

```
...| fieldformat <field>=mvfunction(…)
```

…and as part of **`eval`** expressions

```
...| stats function(eval(mvfunction(…)))
```

Create Multivalue Fields with `split`

```
...| eval <field>=split(<mvfield>,"<delimiter>")
```

- Multivalue function
- Separates a single-value field based on a given delimiter and returns a multivalue field as result
  - **`<mvfield>`** is a single-value field
  - **`"<delimiter>"`** is the delimiter that **`<mvfield>`** is split with

```
...| eval newfield=split(fieldA,";")
   | table fieldA newfield
```

fieldA	newfield
value1;value2;value3	value1 value2 value3

`split` Example

Scenario

For the **Last 7 days**, Sales wants to extract each part of the `productId` as a separate value within a multivalue field in order to capture a new field called `productCodes` (The `productId` has the format AA-BB-CCC.)

```
index=web sourcetype=access_combined
action=purchase status=200 productId!=NULL
| eval productCodes = split(productId, "-")
| table productId, productCodes, product_name
```

productId	productCodes	product_name
FS-SG-G03	FS SG G03	Final Sequel
MB-AG-T01	MB AG T01	Manganiello Bros. Tee
DB-SG-G01	DB SG G01	Mediocre Kingdoms

 |

Convert Single Value Fields with `makemv`

```
...| makemv [delim=<string> |tokenizer=<regex>] <field>
```

- Multivalue command
- Converts an *existing* single value field to a multivalue field, based on a provided string or regex
  - **`delim`** specifies a string delimiter; defaults to a single space " "
  - **`tokenizer`** specifies a regular expression
- **`makemv`** is a distributable streaming command

makemv Example

makemv captures words and makes them into values of mv_uri_path

```
index=web uri_path=*
| table uri_path
| dedup uri_path
| eval mv_uri_path = uri_path
| makemv tokenizer="(\w+)" mv_uri_path
```

uri_path	mv_uri_path
/cart/success.do	cart success do
/cart.do	cart do
/productscreen.html	productscreen html
/cart/error.do	cart error do

Evaluate Multivalue Fields

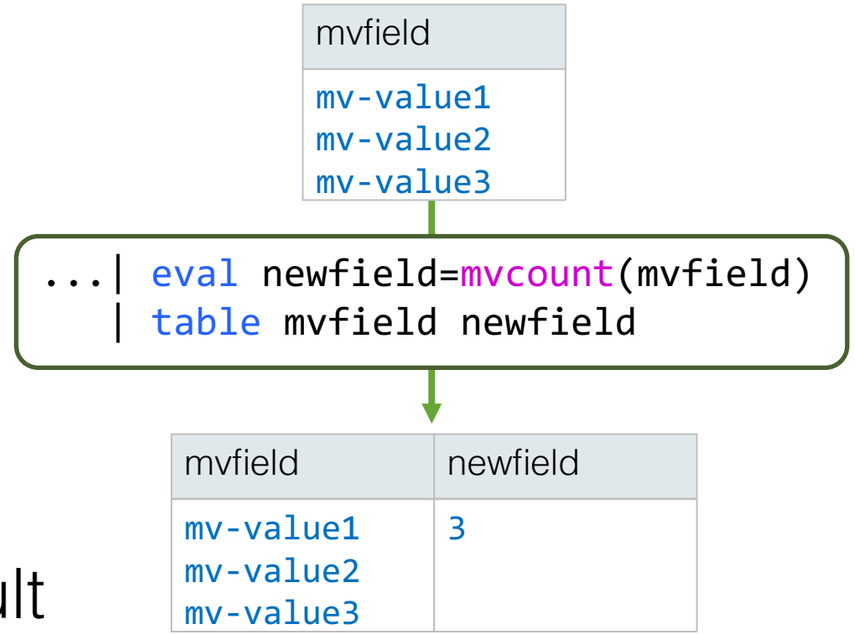
Topic Objectives

- Evaluate multivalue fields using multivalue **`eval`** functions:
  - `mvcount`
  - `mvindex`
  - `mvfilter`

Count Values with `mvcount`

```
...| eval <field>=mvcount(<mvfield>)
```

- Multivalue function
- Counts the number of values for a specified multivalue field, **`<mvfield>`**
- Returns **`NULL`** when:
  - The field has no values
  - If a field does not exist for a particular result



`mvcount` Example

Scenario

Sales wants to know how many transactions in the online store over the last 24 hours contained more than 4 actions within half an hour.

```
index=web sourcetype=access_combined
| transaction JSESSIONID maxspan=30m
| eval actions=mvcount(action)
| where actions > 4
| stats count
```

count
15

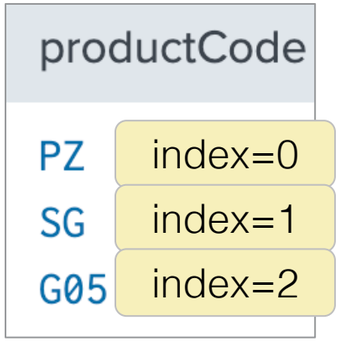
action	actions
addtocart purchase	2
addtocart purchase	2
addtocart purchase view	3

This is what you would see if you put **`action`** and **`actions`** in a table

Pull Specific Values Using `mvindex`

```
...| eval <field>=mvindex(<mvfield>,startIndex,[endIndex])
```

- Multivalue function
- Assigns an array index to a multivalue field and returns a value based on a given integer
  - **`<mvfield>`** is a multivalue field
  - **`startIndex`** and **`endIndex`** are integers referencing the position of a value within a multivalue field index
    - If **`endIndex`** is specified, Splunk returns values between the **`startIndex`** integer and **`endIndex`** integer
    - If **`endIndex`** is not specified, only the value at **`startIndex`** is returned



mvindex Example

Scenario

Sales wants a count of all product names (`product_name`) sold yesterday on the Buttercup Games website whose second `productCodes` value is SH. (Recall that `productCodes` is created from `productId`.)

```
index=web sourcetype=access_combined action=purchase
status=200 productId!=NULL
| eval productCodes = split(productId, "-")
| eval productCode2 = mvindex(productCodes, 1)
| where ( productCode2 = "SH" )
| stats count(productCode2) as count by product_name
```

product_name	count
Fire Resistance Suit of Provolone	128
Holy Blade of Gouda	92
World of Cheese	152
World of Cheese Tee	104

This is what you would see if you put `productCodes` and `productCode2` in a table

productCodes	productCode2
PZ SG G05	SG
DB SG G01	SG
MB AG T01	AG
FI AG G08	AG
SC MG G10	MG

Filter Multivalue Field Values with `mvfilter`

```
...| eval <field>=mvfilter(<booleanExpression>)
```

- Multivalue function
- Filters a multivalue field based on an arbitrary Boolean expression
- **`<booleanExpression>`** can reference only one field at a time

`mvfilter` Example: Two Solutions

Scenario

IT wants a list of all `ROOT_USERS` whose name begins with C.

```
index=systems sourcetype=system_info
| rename ROOT_USERS{} as root_users
| eval names=mvfilter(match(root_users, "^C"))
| dedup names
| table names
```

`match` finds values in **`root_users`** that match the regex: **`^C`**

Use rename to make **`ROOT_USERS{}`** easier to use

```
index=systems sourcetype=system_info
| rename ROOT_USERS{} as root_users
| eval names=mvfilter(root_users LIKE "C%")
| dedup names
| table names
```

The **`LIKE`** Boolean finds values in **`root_users`** that start with **`C`**

names
Camilian
Camilian
Cardozo

Create and Evaluate Multivalue Fields Lab Exercise

Time: 15 minutes

Tasks:

- Use the **`makemv`** command to convert the **`productId`** field into a multivalue field and find all products sold yesterday whose **`productId`** contains "SH"
- Use various multivalue **`eval`** functions to complete a search that will find employees who used a workstation other than their own
- Challenge: Test the previous search by using an **`eval`** function to simulate the scenario of an employee logging into another employee's workstation

Analyze Multivalue Data

Topic Objectives

- Analyze multivalue data using multivalue **`eval`** functions:
  - **`mvsort`**
  - **`mvzip`**
  - **`mvjoin`**
  - **`mvmap`**
  - **`mvappend`**
- Analyze multivalue data using the **`mvexpand`** multivalue command

Sort Multivalue Field Values with `mvsort`

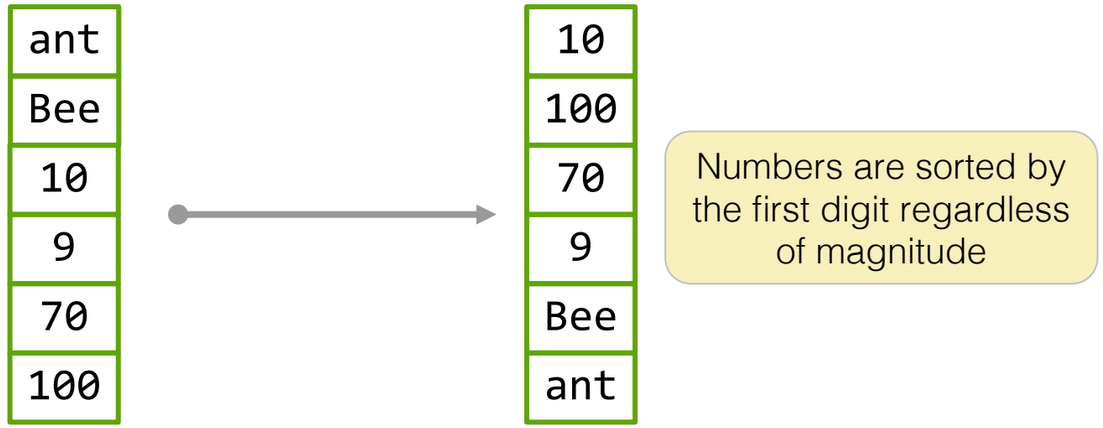
```
...| eval <field>=mvsort(<mvfield>)
```

- Multivalue function
- Returns a multivalue field with values sorted lexicographically
- `<mvfield>` is a multivalue field

`mvsort` Function: Lexicographical Order

Items are sorted based on their encoding; in Splunk, encoding is almost always UTF-8

- Numbers are sorted based on the first digit
- Numbers are before letters
- Uppercase letters are before lowercase letters



Note

The `mvsort` function always sorts lexicographically. This is different than the `sort` command which first determines what type of field it is sorting and then sorts based on field type.

`mvsort` Example

Scenario

IT wants a list of root users for each of its virtual AWS instances (i.e., systems) for the last 7 days.

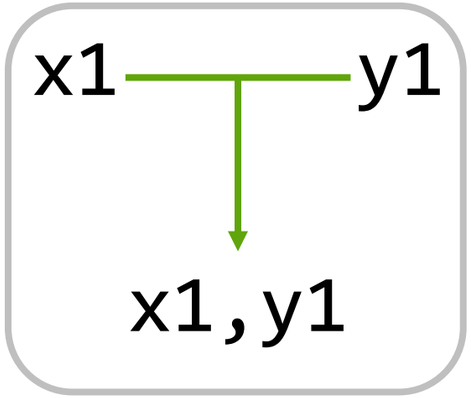
```
index=systems sourcetype=system_info
| rename ROOT_USERS{} as root_users,
  SYSTEM{} as system
| eval root_users=mvsort(root_users)
| table system, root_users
| stats values(root_users) as root_users by system
```

system	root_users
nix_system_idx_1	Admin Camilian Cardozo Duke
nix_system_idx_1_UK	Admin Camilian Cardozo Duke
nix_system_idx_2	Beulah Camilian Duke Zed

Combine Multivalue Fields with `mvzip`

```
...| eval <field>=mvzip(X,Y["<delimiter>"])
```

- Multivalue function
- Concatenates the values of two multivalue fields with a delimiter
  - `X` and `Y` are multivalue fields
  - The first value of `X` is concatenated with the first value of `Y`, etc.
  - "`<delimiter>`" is used as the separator; if none specified, the default delimiter is a comma (`,`)



- Fields `X` and `Y` should have the same number of values
- Resulting field values are treated as strings

`mvzip` Example 1

Scenario

IT wants a report for the last 24 hours of all its AWS systems, along with each of its cores and the percent used for each core. Present the `core` and `core_percent_used` in a single field, concatenated by a colon and a space. Show only cores with usage more than 0.0.

```
index=systems sourcetype=system_info
| rename SYSTEM{} as system, CPU_CORES{}.core as core,
  CPU_CORES{}.core_percent_used as percent_used
| eval zip_percent_used = mvzip(core, percent_used, ": ")
| stats count as sum_core by system, asctime,
  zip_percent_used
| search zip_percent_used!="*: 0.0"
| stats list(zip_percent_used) as "CPU Core Usage" by
  system, asctime
```

core	percent_used
core_number_5	1.0

system	asctime	CPU Core Usage
nix_system_idx_1	2020-09-16 23:05:16,950	core_number_5: 1.0
nix_system_idx_1	2020-09-16 23:08:05,156	core_number_2: 1.0
nix_system_idx_1	2020-09-16 23:09:01,225	core_number_8: 1.0
nix_system_idx_1	2020-09-16 23:15:33,685	core_number_6: 1.0
nix_system_idx_1	2020-09-16 23:20:14,031	core_number_12: 1.0 core_number_15: 1.0 core_number_9: 1.0

`mvzip` Example 2

If the `X` and `Y` fields do not have the same number of values, the results will not be complete

```
index=systems sourcetype=system_info
| rename SYSTEM{} as system
| rename ROOT_USERS{} as users
| eval systemUsers=mvzip(system,users,"-")
| table system, users, systemUsers
```

The field with fewer values determines how many values the new field will have

system	users	systemUsers
nix_system_idx_4	Admin Duke Zed Camilian	nix_system_idx_4-Admin
nix_system_idx_4_UK	Admin Duke Zed Camilian	nix_system_idx_4_UK-Admin
nix_system_idx_3	Beulah Camilian Admin Nells	nix_system_idx_3-Beulah

Concatenate with `mvjoin`

```
...| eval <field>=mvjoin(<mvfield>,"<delimiter>")
```

- Multivalue function
- Concatenates, i.e., links together, values of a multivalue field with a specified delimiter, which outputs to a single value field
  - **`<mvfield>`** is a multivalue field
  - **`"<delimiter>"`** is used as the separator

mvjoin Example

Scenario

IT wants to see a list of root users (concatenated with a comma) on each of its AWS instances during the last 60 minutes.

```
index=systems sourcetype=system_info
| rename ROOT_USERS{} as users, SYSTEM{} as system
| eval users=mvsort(users), users=mvjoin(users,",")
| stats values(system) as systems by users
```

users	systems
Admin,Beulah,Camilian,Nells	nix_system_idx_3
Admin,Camilian,Cardozo,Duke	nix_system_idx_1
Admin,Camilian,Duke,Zed	nix_system_idx_4
Beulah,Camilian,Duke,Zed	nix_system_idx_2

Operate on Multivalue Fields with **mvmap**

```
...| eval <evalField>=mvmap(<mvfield>,<expression>)
```

- Multivalue function
- Performs an operation over the values of a multivalue field then returns a multivalue field as a result
  - **<mvfield>** is the multivalue field to be operated on
  - **<expression>** provides the operation to execute

mvmap Example

```
index=systems sourcetype=system_info
| rename SYSTEM{} as system, CPU_CORES{}.core as core,
  CPU_CORES{}.core_percent_used as percent_used
| eval percent_used = mvmap(percent_used, percent_used/2)
| eval zip_percent_used = mvzip(core, percent_used, ": ")
| stats count as sum_core by system, asctime,
  zip_percent_used
| search zip_percent_used!="*: 0.0"
| stats list(zip_percent_used) as "CPU Core Usage" by
  system, asctime
```

mvmap applies the operation (dividing by 2) to all the values of **percent_used**

system	asctime	CPU Core Usage
nix_system_idx_1	2019-09-06 17:45:26,956	core_number_11: 0.50
nix_system_idx_1	2019-09-06 17:51:03,371	core_number_4: 0.50
nix_system_idx_1	2019-09-06 17:53:51,578	core_number_11: 0.50
nix_system_idx_1	2019-09-06 18:02:16,201	core_number_9: 0.50

Combine Fields and Values with `mvappend`

```
...| eval <field>=mvappend(arg1, arg2, arg3, ..., argN)
```

- Multivalue function
- Takes two or more single-value fields, multivalue fields, and strings and returns a multivalue field as the result
- Each input becomes a separate value within the new multivalue field

fieldA	mvfield
value	mv-value1 mv-value2

```
...| eval newfield=mvappend(fieldA,"abc",mvfield)
   | table fieldA mvfield newfield
```

fieldA	mvfield	newfield
value	mv-value1 mv-value2	value abc mv-value1 mv-value2

mvappend Example

```
| makeresults
| eval manufacturer = "Donco",
       city = "Shenzen",
       country = "China",
       products = mvappend("Holy Blade of Gouda", "Mediocre Kingdoms",
                           "Dream Crusher")
```

_time	city	country	manufacturer	products
2022-06-21 15:17:26	Shenzen	China	Donco	Holy Blade of Gouda Mediocre Kingdoms Dream Crusher

The new values of the multivalue field, **products**

Note

makeresults is a useful command for creating data to test and troubleshoot with.

Analyze Multivalue Fields with **mvexpand**

```
...| mvexpand <mvfield> [limit=<integer>]
```

- Multivalue command
- Takes a multivalue field and creates a separate event for each value; related field values are copied from the original event
  - **<mvfield>** is a multivalue field
  - Assigning an **<integer>** to **limit** determines the number of new events created from a multivalue field
- **mvexpand** is a distributable streaming command

field	mvfield
value1	mv-value1 mv-value2

```
...| mvexpand mvfield
```

field	mvfield
value1	mv-value1
value1	mv-value2

mvexpand Example

```
index=systems sourcetype=system_info
| rename ROOT_USERS{} as users, SYSTEM{} as system
| table system users
| mvexpand users
```

At this point, **users** is a multivalue field

system	users
nix_system_idx_1	Duke Camilian Admin Cardozo
nix_system_idx_4	Admin Duke Zed Camilian

system	users
nix_system_idx_1	Duke
nix_system_idx_1	Camilian
nix_system_idx_1	Admin
nix_system_idx_1	Cardozo
nix_system_idx_4	Admin
nix_system_idx_4	Duke

Values of **users** have been split up into their own events

Original event data that has been copied to each new event

Better Together: `mvexpand` + `fields`

- Remember: new events are created in memory, not in the index
- It is recommended to use **`fields`** before **`mvexpand`** to retain only the necessary fields for the search
  - Conserves memory usage and improves performance

```
index=systems sourcetype=system_info
| rename SYSTEM{} as system, CPU_CORES{}.core as core,
  CPU_CORES{}.core_percent_used as percent_used
| table system core percent_used
| eval zip_percent_used= mvzip(core, percent_used, ": ")
| fields system zip_percent_used
| mvexpand zip_percent_used
```

This search has completed and has returned **1,037,232** results by scanning **86,436** events in **4.07** seconds

```
index=systems sourcetype=system_info
| rename SYSTEM{} as system, CPU_CORES{}.core as core,
  CPU_CORES{}.core_percent_used as percent_used
| table system core percent_used
| eval zip_percent_used=mvzip(core, percent_used, ": ")
| mvexpand zip_percent_used
```

This search has completed and has returned **1,037,160** results by scanning **86,430** events in **6.911** seconds

Analyze Multivalue Data Lab Exercise

Time: 10 minutes

Tasks:

- Display information about AWS system data by completing a search with multivalue **`eval`** and **`stats`** functions
- Challenge: Modify the previous search so that **`_number`** is removed from each value of **`CPU_CORES{}.core`**

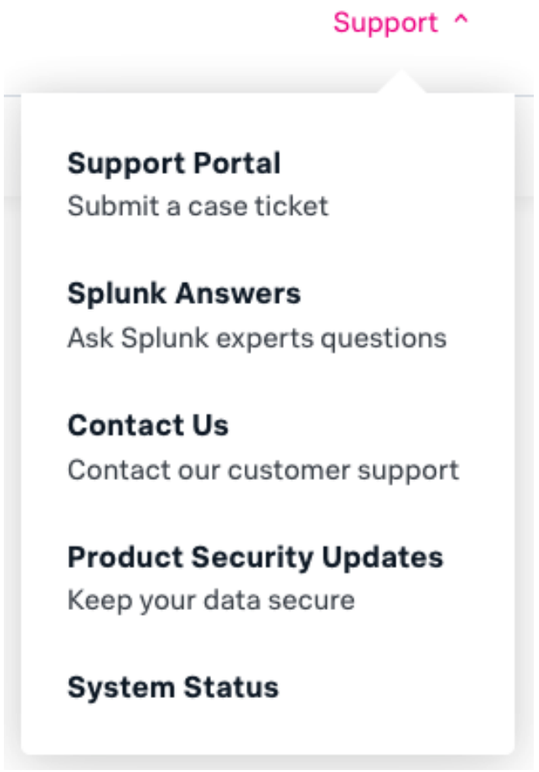
Wrap-up Slides

Community

- Splunk Community Portal
  community.splunk.com
  - Answers
  - Discussions
  - Splunk Trust
  - User Groups
  - Ideas
- Splunk Blogs
  splunk.com/blog/
- Splunk Apps
  splunkbase.com
- Splunk Dev Google Group
  groups.google.com/forum/#!forum/splunkdev
- Splunk Docs on Twitter
  twitter.com/splunkdocs
- Splunk Dev on Twitter
  twitter.com/splunkdev
- Splunk Live!
  splunklive.splunk.com
- .conf
  conf.splunk.com

Support Programs

- **Web**
  - Documentation: dev.splunk.com and docs.splunk.com
  - Wiki: wiki.splunk.com
- **Splunk Lantern**
  Guidance from Splunk experts
  - lantern.splunk.com
- **Global Support**
  Support for critical issues, a dedicated resource to manage your account – 24 x 7 x 365
  - Web: splunk.com/index.php/submit_issue
- **Enterprise, Cloud, ITSI, Security Support**
  - Web: splunk.com/en_us/about-splunk/contact-us.html#tabs/customersupport
  - Phone: (855) SPLUNK-S or (855) 775-8657



Learning Paths (cont.)

Knowledge Manager - Recommended Courses

Free eLearning courses are in blue and courses with an * are present in both learning paths.

- What is Splunk *
- Introduction to Splunk *
- Using Fields *
- Introduction to Knowledge Objects
- Creating Knowledge Objects
- Creating Field Extractions
- Enriching Data with Lookups
- Data Models
- Introduction to Dashboards
- Dynamic Dashboards
- Using Choropleth
- Search Optimization *

Learning Paths

Search Expert - Recommended Courses

Free eLearning courses are in blue and courses with an * are present in both learning paths.

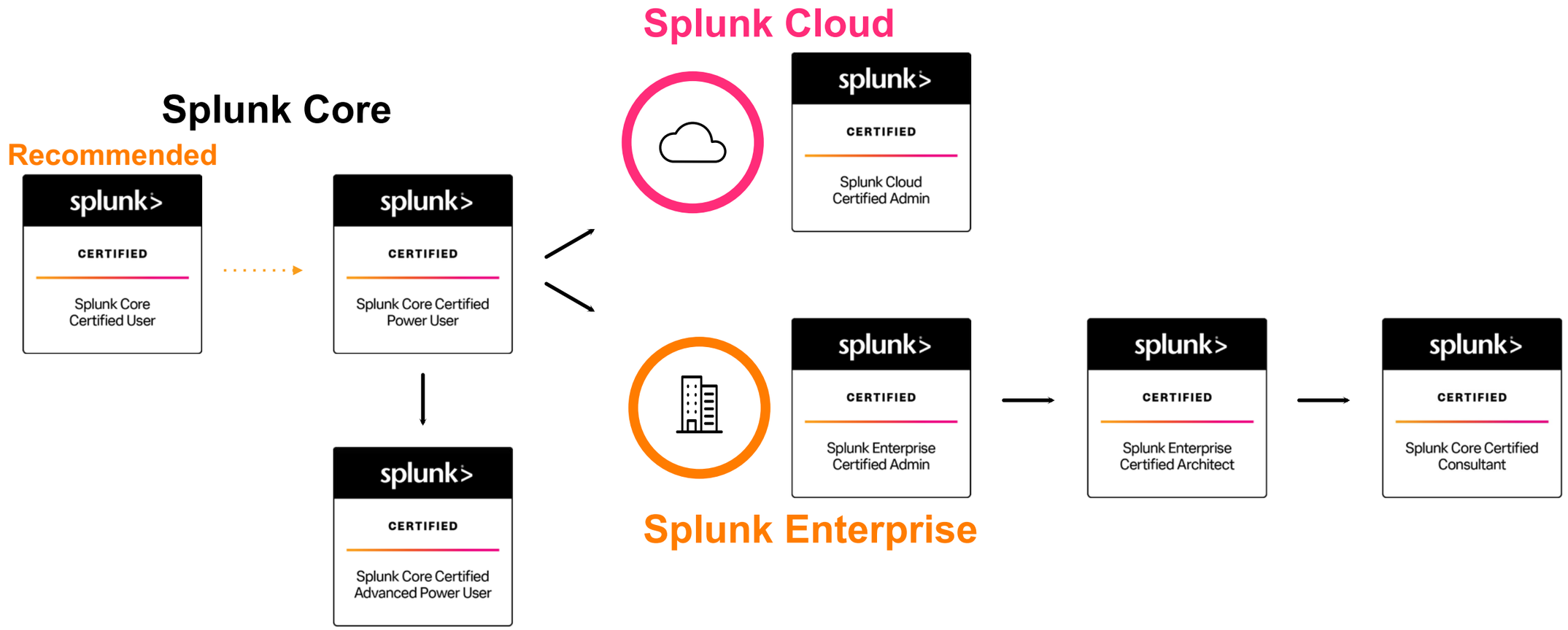
- What is Splunk *
- Introduction to Splunk *
- Using Fields *
- Scheduling Reports and Alerts
- Visualizations
- Statistical Processing
- Working with Time
- Comparing Values
- Result Modification
- Leveraging Lookups and Subsearches
- Correlation Analysis
- Search Under the Hood
- Multivalue Fields
- Search Optimization *

Splunk Certification

Offerings & Requirements

Splunk Core and Beyond

Regardless of which Splunk product you use, it all starts with Splunk Core



Splunk Core Certified Advanced Power User

This certification demonstrates an individual's ability to generate complex searches, reports, and dashboards with Splunk's core software to get the most out of their data

Prerequisite Certification(s):

- Splunk Core Certified Power User

Prerequisite Course(s):

- None

Splunk Core Certified Advanced Power User Exam

Time to study! We suggest candidates looking to prepare for this exam complete Fundamentals 3, Creating Dashboards, and Advanced Searching & Reporting **or** the following courses:

- Using Fields
- Working with Time
- Comparing Values
- Result Modification
- Leveraging Lookups and Subsearches
- Correlation Analysis
- Search Under the Hood
- Multivalue Fields
- Search Optimization
- Creating Field Extractions
- Enriching Data with Lookups
- Data Models
- Using Choropleth
- Introduction to Dashboards
- Dynamic Dashboards

See here for registration assistance.

Congratulations! You are a...

Recommended Next Steps

- Splunk Enterprise Certified Admin
- Splunk Cloud Certified Admin

 | 11 August 2022

Thank You