Splunk Search 101

Lab Guide

Overview

The purpose of this workshop is to enable your teams to search, investigate, analyze, and report on data in the Splunk platform in order to increase your teams' productivity and efficiency in identifying and resolving issues.

Prerequisites

In order to complete these exercises, you will need your own Splunk instance. Splunk's hands-on workshops are delivered via the Splunk Show portal and you will need a Splunk.com account in order to access this.

If you don't already have a Splunk.com account, please create one here before proceeding with the rest of the workshop.

⚠️ Troubleshooting Connectivity

If you experience connectivity issues with accessing either your workshop environment or the event page, please try the following troubleshooting steps. If you still experience issues please reach out to the team running your workshop.

- **Use Google Chrome** (if you're not already)
- If the event page (i.e. *https://show.splunk.com/event/<eventID>*) didn't load when you clicked on the link, try **refreshing the page**
- **Disconnect from VPN** (if you're using one)
- **Clear your browser cache and restart your browser** (if using Google Chrome, go to: Settings > Privacy and security > Clear browsing data)
- **Try using private browsing mode** (e.g. Incognito in Google Chrome) to rule out any cache issues
- **Try using another computer** such as your personal computer - all you need is a web browser! Cloud platforms like AWS can often be blocked on corporate laptops.

Table of Contents

Table of Contents 2

Lab 1 – Searching with Keywords, Field-Value Pairs, and Booleans 3

Example #1: App and Web log events in the last 60 min 3

Solution 3

Example #2: App log events for a specified host and platform in the last 60 min 3

Solution 3

Example #3: App log events for a specified host and excluding a specified platform in the last 60 min 4

Solution (best efficiency) 4

Solution (alternative) 4

Lab 2 – The stats Command 5

Goal: Write an SPL query to track the number of errors vs. successes as well as the average load times in your App log events in the last 60 min and group the results by the platform field 5

Solution 5

Lab 3 – The timechart Command 6

Goal: Write an SPL query to chart the median load times across platforms over time 6

Solution 7

Lab 4 – The eval & where Commands 8

Goal: Write an SPL query to find apps with high errors or load times in the last 60 min 8

Solution 9

Alt. Solution 10

Lab 1 – Searching with Keywords, Field-Value Pairs, and Booleans

Example #1: App and Web log events in the last 60 min

1. Navigate to the **Search & Reporting** app and click on the **Search** tab in the menu bar.
2. Start by exploring the available data in your environment! Under the "How to search" section, click on **Data Summary**
    a. You will see that there are only 4 source types available in our lab environment.
3. **Write an SPL search** to find all App log and Web log events in the last 60 min:
    a. Looking at the available source types in the previous step, we can easily identify the relevant ones to use in this query:
        i. For Web log events, we will look at sourcetype=access_combined
        ii. For App log events, we will look at sourcetype=tests-json
    b. This query will use field-value pairs only

Solution

```
sourcetype=tests-json OR sourcetype=access_combined
```

Example #2: App log events for a specified host and platform in the last 60 min

1. Navigate to the **Search & Reporting** app and click on the **Search** tab in the menu bar if you're not already there.
2. **Write an SPL search** to find all App log events in the last 60 min where the host is test-01 and the platform is iOS:
    a. Start by specifying the sourcetype, since we already know App logs will have sourcetype=tests-json
    b. Specify the provided host value in the host field, and do the same for the platform value
    c. This query will use field-value pairs only

Solution

```
sourcetype=tests-json host=test-01 platform=iOS
```

Example #3: App log events for a specified host and excluding a specified platform in the last 60 min

1. Navigate to the **Search & Reporting** app and click on the **Search** tab in the menu bar if you're not already there.
2. **Write an SPL search** to find all App log events in the last 60 min where the host is test-01 and the platform is **NOT** iOS:
    d. Start by specifying the sourcetype, since we already know App logs will have `sourcetype=tests-json`
    e. Specify the provided host value in the host field
    f. This query will use field-value pairs as well as boolean operators
        i. Use a boolean operator to exclude events where platform=iOS from the results returned

Solution (best efficiency)

```
sourcetype=tests-json host=test-01 platform!=iOS
```

Solution (alternative)

```
sourcetype=tests-json host=test-01 NOT platform=iOS
```

Lab 2 – The `stats` Command

Goal: Write an SPL query to track the number of errors vs. successes as well as the average load times in your App log events in the last 60 min and group the results by the platform field

1. Navigate to the **Search & Reporting** app and click on the **Search** tab in the menu bar if you're not already there.
2. Since we want to focus on App log events, we will use: `sourcetype=tests-json`
3. The statistical calculations we want to perform are the following:
    a. To count the total number of events: `stats count`
    b. To calculate the average load time: `stats avg(loadtime)`
    c. (Optional) To calculate the 90th percentile value for the load time field: `stats perc90(loadtime)`
4. We can combine the above stats commands into a single one and group the results by the platform field as follows in order to find the total number of events and the average load time by platform:
```
stats count avg(loadtime) by platform
```
5. Since we also want to include the type of event (error vs success) in our results, we'll add it to the group-by clause in our stats command:
```
stats count avg(loadtime) by platform type
```
6. Lastly, we want to sort in descending order, so we use the sort command:
```
sort - count
```

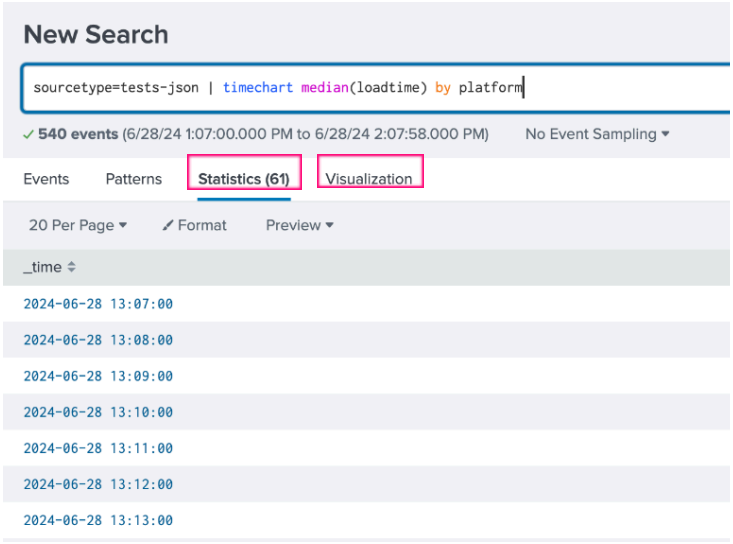
Solution

```
sourcetype=tests-json
| stats count avg(loadtime) by platform type
| sort - count
```

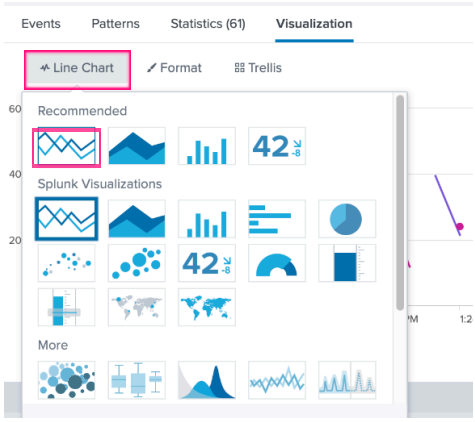
Lab 3 – The `timechart` Command

Goal: Write an SPL query to chart the median load times across platforms over time

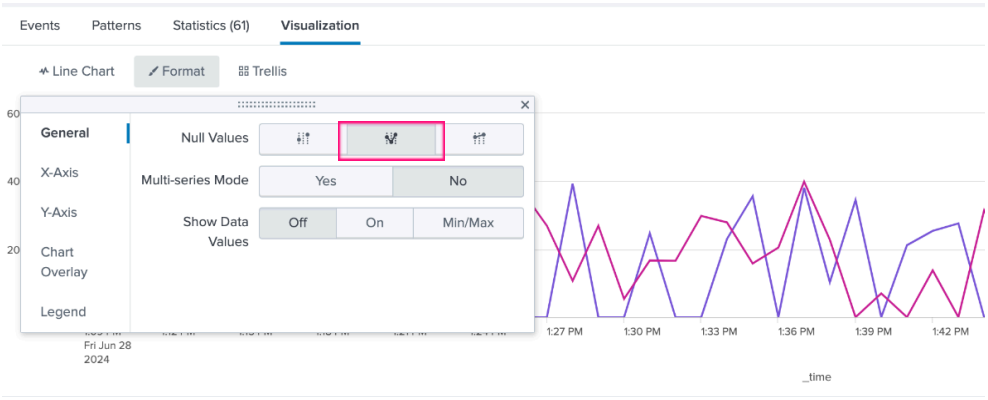
1. Navigate to the **Search & Reporting** app and click on the **Search** tab in the menu bar if you're not already there.
2. Since we want to focus on App log events, we will use: `sourcetype=tests-json`
3. The only thing we want to calculate here is the median load times by platform over time, which we can do by using timechart as follows:
   ```
   | timechart median(loadtime) by platform
   ```
4. Once we get the results we are looking for, we should visualize them in a **Line Chart**:
   a. In the results window, switch from the **Statistics** tab to the **Visualization** tab



   b. Select **Line Chart** from the visualization options



c. Click on **Format** and change the **Null Values** option to **Zero** so that the lines in the chart will be fully connected throughout the chart

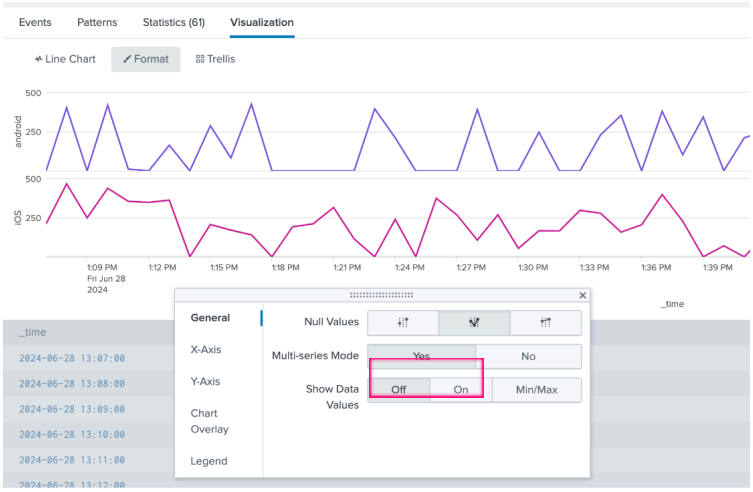


Solution

```
sourcetype=tests-json
| timechart median(loadtime) by platform
```

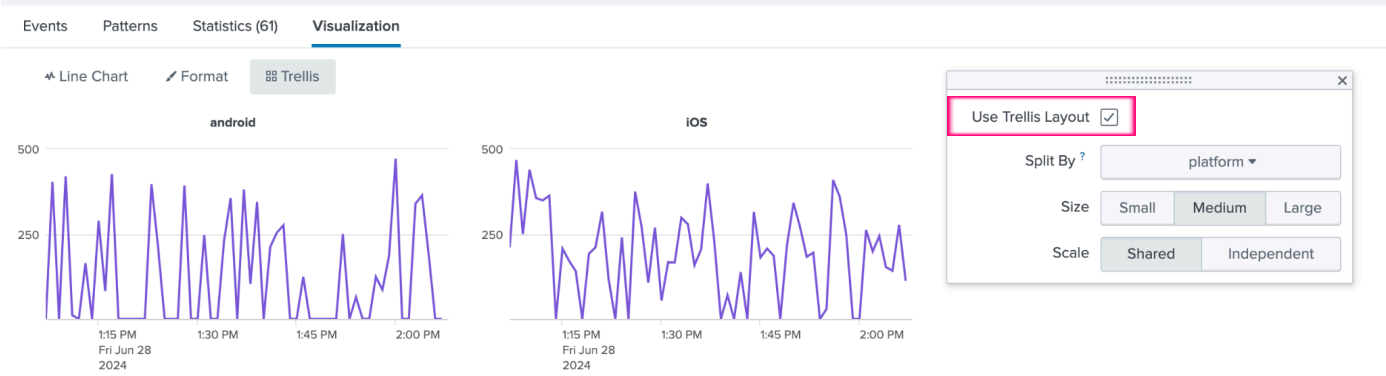
5. (Optional) If instead we wanted to see each series in its own separate chart, we could do so by following these simple steps:

a. Set the **Multi-series Mode** to **Yes**



b. Open the **Trellis** menu to the right of the **Format** menu

c. Check the **Use Trellis Layout** box, and adjust the size of the charts



Lab 4 – The `eval` & `where` Commands

Goal: Write an SPL query to find apps with high errors or load times in the last 60 min

1. Navigate to the **Search & Reporting** app and click on the **Search** tab in the menu bar if you're not already there.
2. Since we want to focus on App log events, we will use: `sourcetype=tests-json`
3. We are interested in looking at the amount of error events as well as the average load time in our app logs for each platform
    a. We can get this information using the `stats` command
        i. To count the number of "error events" (events where the value of the type field is "error") we can use an `eval` expression within the `count` function itself and save this value in a new field:
        `| stats count(eval(type="error")) as num_errors by platform`
        ii. To calculate the average load time for each platform we can use the `stats` command and the `avg()` function:
        `| stats avg(loadtime) by platform`
        iii. We can then combine these two pieces into a single `stats` command:
        `| stats count(eval(type="error")) as num_errors avg(loadtime) by platform`
4. We can now use the results from the previous step to determine whether the number of errors and the average load times per platform are "high"
    a. Just for the purposes of this exercise, we are going to combine the 2 calculated values into a new field called "app_danger" by multiplying the two:
    `| eval app_danger = num_errors * "avg(loadtime)"`
    b. The value of the app_danger field represents a "score" that will become higher or lower based on both the avg load times and the amount of error events
        i. Since we are multiplying them, if one of the 2 values drops or grows significantly, so will the score
        ii. We will compare the score to a threshold value, and if the score surpasses the threshold, we can say there is a high number of errors or load time

5. Now, we can take our calculated “score” and compare it with our threshold, which for the purposes of this exercise we are going to set as 10,000

    a. We want to find when the “score” is “high,” so we only care about the times when the score goes over our established threshold, and not if it is equal to or less than the threshold

    b. To filter down our results to only show us when that “score” is high we will use the `where` command:
    `| where app_danger > 10000`

6. Lastly, we are going to sort our results in descending order based on the values of the app_danger field (scores):
    `| sort - app_danger`

Solution

```
sourcetype=tests-json platform!=””
| stats count(eval(type=”error”)) as num_errors avg(loadtime) by platform
| eval app_danger = num_errors * “avg(loadtime)”
| where app_danger > 10000
| sort - app_danger
```

*** ***Note:*** *In a real environment, the expression to calculate the app_danger would most likely not be a simple multiplication, rather a more thorough calculation. Similarly, the threshold would most likely come from statistical analysis of our events to determine what is “normal” for us or even from leveraging the Machine Learning toolkit in Splunk.* ***

Alt. Solution

This alternative solution uses static thresholds to determine if the number of error events is too high, if the average load time is too high, or if both things are true.

Depending on which of these conditions is true, a different value is assigned to a new field called app_danger. Then, depending on the values from the app_danger field, we populate another field called results with a different message to be outputted into our results.

Lastly, we sort the results in descending order based on the value of app_danger. Line-by-line comments are included in the search below for reference.

```
sourcetype=tests-json platform!=""
``` we are interested in App log events so we focus on the tests-json sourcetype,
and we want to make sure the platform field is not empty ```
| stats count as total_events count(eval(type="error")) as num_errors
avg(loadtime) as avg_loadtime by platform
```
total number of events is saved into new field total_events,
total number of error events is saved into new field num_errors,
average loadtime is saved into new field avg_loadtime
calculations are grouped by the platform values
```
| eval app_danger=case((avg_loadtime>=235 AND num_errors>=total_events*0.4),3,
(avg_loadtime<235 AND num_errors>=total_events*0.4),2,(avg_loadtime>=235 AND
num_errors<total_events*0.4),1,true(),0)
```
create new field app_danger
if the average loadtime value is above the specified threshold of 235 AND the
number of error events is equal to or higher than 40% of the total number of
events, set the app_danger value to 3
else if the average load time value is below its specified threshold, but the
number of error events is equal to or greater than its threshold, set the
app_danger value to 2
else if the average load time value is above its threshold, but the number of
error events is below the threshold, set the value of app_danger to 1
else, set the value of app_danger to 0
```
| where app_danger>0
``` filter events further, only return results where the app_danger value is
greater than 0, meaning at least one of the values of interest is above its
threshold ```
```

```
| eval results=case(app_danger==3, "Both High Number of Errors and High Avg Load Time", app_danger==2, "High Number of Errors", app_danger==1, "High Avg Load Time")
``` create a new field called results, set the value to the appropriate message based on the app_danger values: 3=both number of errors and avg load time are too high, 2=high number of errors (but no problem with avg load time), 1=high avg load time (but no problem with number of error events)```
| sort - app_danger
``` sort results by the app_danger value in descending order: if both values are too high, those results would be at the top, followed by results where only number of errors is high, then avg load time```
```