Converting logs into metrics

Problem

As an experienced Splunk admin, you have followed data onboarding best practices and have a well-formatted JSON event that you hand off to a data consumer.

```
{
    "device": {
        "deviceId": "127334527887",
        "deviceSourceId": "be:f3:af:c2:01:f1",
        "deviceType": "IPGateway"
    },
    "timestamp": 1489095004000,
    "rawAttributes": {
        "WIFI_TX_2_split": "325,650,390,150,150,780,293,135,325",
        "WIFI_RX_2_split": "123,459,345,643,234,534,123,134,656",
        "WIFI_SNR_2_split": "32,18,13,43,32,50,23,12,54",
    }
}
```

The consumer comes back to you, however, with the following complaint:

"These events are from our router. The device field at the top describes the router itself, and then the rawAttributes describes all of the downstream devices that connect to the router and their respective performance values like transmit, receive, and signal to noise values. We want to be able to report on these individual downstream devices over time and associate those individual devices with the router that serviced them as well as investigate the metrics over time. We use this data to triage customer complaints and over time, improve the resiliency of our network."

This means that multi-device single events must be transformed into distinct values. Worse, every single search that works with this data will need to contain all that same transformation logic, every time. You could do some ingestion-time props and transforms work to pre-format the data, coupled with some summary stats searches to transform these events for consumption. But this solution required more valuable CPU cycles used, slower search performance overall, additional resource contention, and really just a bad day for the data consumer. You need a better solution.

Solution

You need to pre-process the data. Building metrics with dimensions for each of the consumer devices will allow them to rely on super-fast mstats for their near real-time reporting. With familiar search processing language, you can apply the needed transformations in the stream, before the data is indexed. Doing so removes complexity from the
data, reduces search and index time resource consumption, improves data quality, and in the case of this
customer, reduces their mean time to identify problems in their environment because they're not waiting for actionable
data to be generated.

This example uses data sent from a heavy forwarder to Splunk DSP firehose. You may need to change the data source and applicable Splunk Technical Add-ons to fit your environment.

1. Run the following search:
   ```splunk
   | from splunk_firehose() | where source="carrierjson"
   | eval bodyjson = from_json_object(tostring(body))
   | eval attr=ucast(bodyjson.rawAttributes, "map<string,string>", null),
     tx=split(attr.WIFI_TX_2_split, ","),
     rx=split(attr.WIFI_RX_2_split, ","),
     snr=split(attr.WIFI_SNR_2_split, ","),
     macs=split(attr.ClientMac_split, ","),
   | eval deviceRange = mvrange(0,length(macs)-1)
   | eval temp=for_each(iterator(deviceRange, "i"), {
     "timestamp": timestamp,
     "host": spath(bodyjson, "device.deviceId"),
     "attrs": {"default_dimensions": {"mac": mvindex(macs, i)}},
     "body": [{"name": "wifi_tx", "value": mvindex(tx, i)},
           {"name": "wifi_rx", "value": mvindex(rx, i)},
           {"name": "wifi_snr", "value": mvindex(snr, i)}]
   }]
   | mvexpand limit=0 temp
   | eval host=tostring(temp.host), attributes=ucast(temp.attrs,
     "map<string,any>", null), body=temp.body, kind="metric"
   | select host, timestamp, attributes, body, kind, source
   | into splunk_enterprise_indexes("Splunk_Connection", "carrier_metrics",
     "main");
   ```

2. Verify with a search that the metrics are flowing in.
3. Using the analytics workbench, build charts from these metrics to verify that the MAC address and host dimensions are both available for splitting.

---

**Search explanation**

The table provides an explanation of what each part of this search achieves. You can adjust this query based on the specifics of your environment.

<table>
<thead>
<tr>
<th>SPL</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Search data from Splunk DSP Firehose and filter the results to only the carrierjson source.</td>
</tr>
</tbody>
</table>

The information provided in Splunk Lantern is intended for informational and educational purposes only. All information is provided in good faith, however, Splunk disclaims any and all representations and warranties, express and implied, regarding the information provided, including without limitation any warranties and representations regarding the completeness, adequacy or accuracy of the information. You agree to take full responsibility for the results arising from the use of the information provided.
| eval bodyjson = from_json_object(tostring(body))

Convert the body text into a string, then into json for easy field access. The body is initially a “union” type and requires conversion to a string prior to parsing the json. The resulting json is stored in a field called bodyjson. The new bodyjson field is a structured element that can be referenced with dot notation.

| eval attr=ucast(bodyjson.rawAttributes, "map<string,string>", null),
  tx=split(attr.WIFI_TX_2_split, ","),
  rx=split(attr.WIFI_RX_2_split, ","),
  snr=split(attr.WIFI_SNR_2_split, ","),
  macs=split(attr.ClientMac_split, ",")

Our metrics and the client mac address are embedded inside of a field. Use the split command to turn these delimited values into their own multi-value fields. Use ucast for some additional needed type conversion.

It’s important to note that SPS is strongly and implicitly typed. In order to split the data, the field feeding it must be seen as a string. Use ucast to take the "rawAttributes" field from the bodyjson and turn it into a map/json of type string. Split can then be used to
create multi-value fields from each of the attributes. Repeat this for each metric field.

| eval deviceRange = mvrange(0,length(macs)-1)

Use a counter to help loop through the multivalue results. In this data, the first value in tx, rx, and snr all relate to the first value in the mac address field, as does the second value in each of those fields belong to the second mac address, and so on through n. The counter needs to go from the beginning to the end. Use `mvrange` to generate a list of numbers from 0 through the total count of mac addresses in the data. This helps to create new events from the embedded records.
| eval temp=for_each(iterator(deviceRange, "i"),
"timestamp": timestamp,
"host": spath(bodyjson, "device.deviceId"),
"attrs": {"default_dimensions": {"mac": mvindex(macs, i)}},
"body": [{"name": "wifi_tx", "value": mvindex(tx, i)},
          {"name": "wifi_rx", "value": mvindex(rx, i)},
          {"name": "wifi_snr", "value": mvindex(snr, i)}])

You may have previously used a “for each” in SPL to loop through fields. Use that same concept here as well as an iterator to give sequential access to each member of a multi-value field. Use the `iterator` command to walk through each of the device index numbers. Each loop of the “for each” generates a new array item into temp, and that array item is the metrics json string.

This part of the SPL builds JSON in-line as that resulting object and references `mvindex` of each metrics field, with the placeholder "i" as this loop’s respective index number. The resulting JSON matches the schema/format SPS expects to send metrics formatted data to Splunk.

The field "temp" is now a multi-value field that contains 1 block of metrics per device. The first item belongs to the mac address "9D:A2:89:CB:AC:F7", and the body contains 3 metrics, one for each data point tx, rx, and snr (only tx is expanded in the image).
Using **mvexpand**, create 1 new event for each metric stored in the temp field. The pipeline goes from 1 eps to a little less than 8 eps, demonstrating that each original event is expanding into about 8 unique events - 1 per embedded device.

The resulting events contain all the fields from the previous event. Each event is full of all those temporary fields used to build the metrics. Those aren’t valuable and should be culled prior to sending to your Splunk deployment.
Use the host, attributes, and body fields from my `temp` field to overwrite the temporary and unnecessary fields at the top level of the event. Match the host field to the router ID and convert the body field to the metric data, not the _raw data. Convert the attributes to the metrics attributes/dimensions, and set the field "kind" to "metric", which tells your Splunk deployment that this event should be processed as a metrics instead of an event.

```splunk
| eval host=tostring(temp.host),
  attributes=ucast(temp.attribs, "map<string,any>", null),
  body=temp.body, kind="metric"
```

Now the host field that matches the router ID and the body field contains the metric data instead of the _raw data. Attributes now contain the metrics attributes/dimensions, and the field "kind" is set to "metric", which tells your Splunk deployment that this event should be processed as a metrics instead of an event.

```splunk
| select host, timestamp, attributes, body, kind, source
```
SPL  

| into splunk_enterprise_indexes("Splunk_Connection", "carrier_metrics", "main");  

Explanation  

processed as a metric instead of an event. Finally, use select to reduce the output to just these required fields. This leaves a metrics-compatible schema that is ready to send to your Splunk deployment. 

| Name your Splunk connection and the index you want to send to. This example names the index "carrier_metrics", but the data can also provide the index name. You can also point to a field in the data rather than using a specific index name.  

Results  

If you look in search, you'll see metrics flowing in.

If you look at the analytics workbench, you can build charts from these metrics and see that the mac address and host dimensions are both available for splitting.

In ten commands, you have gone from a condensed multivalue plain text event into an expanded metric-based statistic, ready to use within seconds of arriving at Splunk Stream Processor Services. At first glance, this may seem like a lot of work. But when you consider that had you not done this, the user would have had to figure out the best way to split up all those events and that logic would have to be used every single time that data is required. This is a significant barrier removed and it'll make you a hero to your users.

The information provided in Splunk Lantern is intended for informational and educational purposes only. All information is provided in good faith, however, Splunk disclaims any and all representations and warranties, express and implied, regarding the information provided, including without limitation any warranties and representations regarding the completeness, adequacy or accuracy of the information. You agree to take full responsibility for the results arising from the use of the information provided.
Next steps

These additional Splunk resources might help you understand and implement these recommendations:

- Community Case Study: [Complex data transformation using real time-time stream processing](#)
- Blog: [What is Stream Processing?](#)
- Docs: [Stream Processor Service](#)
- Docs: [Data Stream Processor](#)