Improving data pipeline processing in Splunk Enterprise

Trying to modify splunkd using the props.conf and transforms.conf files is not simple. However, you know that applying EVAL logic to and performing regex extractions on pipeline data will allow you to change the value of a field to provide more meaningful information, extract interesting nested fields into top-level fields, and redact certain information from the data. You want to learn how to apply these Splunk transforms during event parsing on an indexer or heavy forwarder.

Altering data ingestion rules is risky. Only advanced Splunk users should do so, and they should always develop on their laptop. When using the configuration samples shown here, you may need to change parameters and values, according to your configuration. In addition, follow these guidelines to help ensure you implement safe transforms:

- Use Visual Studio Code with the Splunk extension to manage configs.
- Use _index_earliest and _index_latest to view recently ingested data
- Use the [copy_to_meta] transform to debug.

Determine license usage

The license_usage.log file is verbose and searches of it are expensive. To better be able to compute and allocate ingestion costs, use INGEST_EVAL to compute the string length for each event and write it to an indexed field.

- props.conf: [default] TRANSFORM-z-last_transform= add_raw_length_to_meta_field
- transforms.conf: [add_raw_length_to_meta_field] INGEST_EVAL = event_length=len(_raw)

Then run the following search to perform license calculation:

```
|tstats
sum(event_length) AS total_ingestion
WHERE index=* _index_earliest=-30d@d _index_latest=-1d@d
BY sourcetype _time span=1d@d
|xyseries_time sourcetype total_ingestion
```

You can split the results by other fields, such as host or source, as needed.

Estimate data requirements

Prior to a new data source being onboarded, additional context might be required to ensure sufficient compute and
storage is available. You can use INGEST_EVAL and CLONE_SOURCETYPE functionality to emit metrics events that can describe the data coming in.

The metric event takes up to 150 bytes of license.

- props.conf: [v:orig:data] TRANSFORMS-enable_estimate_mode_drop_orig = v_estimation_set_metrics,
  v_estimation_create_metrics, v_estimation_drop_orig
- transforms.conf
  • [v_estimation_create_metrics] REGEX = (.*) CLONE_SOURCETYPE = v:estimate:pipeline
  • [v_estimation_set_metrics] INGEST_EVAL = orig_host=<host>, orig_source=<source>,
    orig_sourcetype=<sourcetype>, orig_index=<index>
  • [v_estimation_drop_orig] INGEST_EVAL = queue="nullQueue"
  • [v_estimation_metric_info] INGEST_EVAL = index="<enter index name here>", metric_name="<enter metric name here>", _value=len(_raw)

Then run the following search to see the estimation:

```bash
|mstats prestats=t max(_value) avg(_value)
  WHERE index=data_metrics AND metric_name="estimation_mode"
  BY orig_sourcetype span=5m
|timechart max(_value) avg(_value) BY orig_sourcetype
```

Selective routing to other destinations

Sometimes data need to be shared with other destinations, Splunk Enterprise, or third party systems. Use cases can require all or a subset of data to be shared. INGEST_EVAL can be used to control data routing groups.

- props.conf: [v:interesting:data] TRANSFORMS-example_data_route = v_sample_route_buttercup_bu
- transforms.conf: [v_sample_route_buttercup_bu] INGEST_EVAL = _TCP_ROUTING=if(match(host, "buttercup[12]"), "splunkcloud_indexers", "splunk_onprem_indexers")
- outputs.conf: [tcpout] defaultGroup = splunk_onprem_indexers
  [tcpout:splunkcloud_indexers] server = inputs.buttercup.splunkcloud.com:9997
  [[tcpout:splunk_onprem_indexers] server = 10.10.10.10:9997

Manage conflicting time formats

Any well-curated Splunk Enterprise instance uses sourcetype to accurately identify the event format timestamp. However, collisions occasionally occur in a single sourcetype where there are conflicting date stamps. INGEST_EVAL offers a new approach of using the strptime() function to solve this problem.

- props.conf: [demutliplexed_datetime_formats]
  DATETIME_CONFIG = CURRENT
  TRANSFORMS-extract_date = demultiplex_datetime
- transforms.conf: [demultiplex_datetime] INGEST_EVAL= _time=case(isnotnull(strptime(_raw, "%c")), strptime(_raw, "%c"), isnotnull(strptime(_raw, "%H:%M:%S %y-%m-%d")),strptime(_raw, "%H:%M:%S %y-%m-%d"),

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.
isnotnull(strptime(_raw, "%Y-%m-%d %H:%M:%S")), strptime(_raw, "%Y-%m-%d %H:%M:%S")

This example initially sets the time of the event to be the current time. After this, you use a transform to try to replace that time by testing the known time formats using a case statement and picking the first that matches. This is not very computationally efficient as it invokes strftime multiple times, but you are able to get the answer in a single invocation of INGEST_EVAL.

**Extract the time and date from the file name**

Sometimes the date and time files are split up and need to be rejoined for date parsing. Previously, you would need to use datetime_config.xml and hope for the best or roll your own. With INGEST_EVAL, you can tackle this problem more elegantly.

- props.conf: [compound_date_time] DATETIME_CONFIG = CURRENT
  TRANSFORMS-get-date = construct_compound_date
  SHOULD_LINEMERGE = false
  LINE_BREAKER = (\[\n\r\]+)
- transforms.conf: [construct_compound_date] INGEST_EVAL=_time=strptime(replace(source,".*/(20\d\d-\d\d-\d\d\d)\.log","\1").substr(_raw,0,10),"%Y-%m-%d%H:%M:%S")

The regex replace pops out the date from the source, appends the first 10 chars from _raw, and then runs through strftime and assigns the result to _time. If the eval fails to execute, _time is not updated and the previously set CURRENT time remains.

**Sample events**

Consider a web server that generates 1000s of events per second. You only care about errors and the ratio of errors to OK. You want to sample the OK events and provide high resolution for errors.

- props.conf: [v:orig:data] TRANSFORMS-sample_200_data = v_sample_200_data
- transforms.conf: [v_sample_200_data] INGEST_EVAL = queue=if(match(_raw, "status=200") AND (random()%100)!=0, "nullQueue", "indexQueue")

**Drop fields from INDEXED_CSV**

Both INDEXED_CSV and INDEXED_JSON are useful, but but create indexed fields for every column or element. This can inflate your TSIDX size and increase disk usage. Sometimes you need a subset of these fields for fast search but want to have the remaining available via schema on the fly.

- props.conf: [reduced_columns] DATETIME_CONFIG = CURRENT
  INDEXED_EXTRACTIONS = CSV
  TRANSFORMS-drop_fields = drop_useless_fields
  EXTRACT-removed-columns = [^,]+,[^,]+,[^,]+,(\?<random_nonsense>[^,]+),(\?<long_payload>[^,]+)
Export and import data from Splunk Enterprise

Sometimes you would like to bulk export data from an existing Splunk Enterprise index and reingest it on your laptop for development. This pattern allows you to run a search that extracts data from an install via CSV export and import it again via a specific sourcetype. This is achieved by creating a “protocol” for encoding via search, and then decoding via transforms. Note that this does not reparse data or carry any indexed fields across.

- props.conf: [import_data] DATETIME_CONFIG = CURRENT
  TRANSFORMS-extract-metadata = drop_header, extract_metadata_copy_to_meta, reassign_meta_to_metadata, remove_metadata_from_raw
  SEDCMD-strip_double_quotes= s/""/"/g

- transforms.conf: [drop_header] INGEST_EVAL = queue=if(_raw="\"\"","nullQueue", queue)
  [extract_metadata_copy_to_meta] SOURCE_KEY=_raw
  WRITE_META = true
  REGEX = ^\d+(?:\d+)?%\%%(\[^\%]+)%\%%(\[^\%]+)%\%%(\[^\%]+)%\%%(\[^\%]+)%\%%
  FORMAT = my_index::"$1" my_host::"$2" my_source::"$3" my_sourcetype::"$4"
  [reassign_meta_to_metadata] INGEST_EVAL = host:=my_host, source:=my_source, index:=my_index, sourcetype:=my_sourcetype,
  my_host:=null(), my_source:=null(), my_index:=null(), my_sourcetype:=null()
  [remove_metadata_from_raw] INGEST_EVAL = _raw=replace(_raw,
  "^[\^%]+%\%%(?:[^%]+)%\%%(?:[^%]+)%\%%(?:[^%]+)%\%%(?:[^%]+)%\%%(.*)"","1")

Run the following search:

```
index=* 
|eval _raw=_time.%%%.index.%%%.host.%%%.source.%%%.sourcetype.%%%._raw 
|table _raw
```

The “protocol” uses a %%% as a separator and orders the data as index, host, sourcetype, source, and then _raw. It assumes that the % character is only found in _raw to optimize our REGEX statement.

Extract a REgEx indexed field

By default, Splunk Enterprise ingests data with its universal indexing algorithm, which is a general-purpose tokenization process based around major and minor breakers. However, some log data is consistently named with value attribute pairs and in this instance, you can use REGEX transforms with REPEAT_MATCH = true to implement something similar to “INDEXED_CSV” and “INDEXED_JSON” but for logs. You disable major breakers and write REGEX expressions that find value attribute pairs in the following forms a="b", a=b, a='b' and write out a::b into _meta to create an indexed field with the name “a” and value “b.” Each of the value attribute pairs can be convert via a REGEX transform to indexed fields. Lots of log files follow this pattern, including the splunkdmetrics.log.

- props.conf: [indexed_log] TIME_FORMAT = %Y-%m-%d %H:%M:%S
  SHOULD_LINEMERGE = false
  LINE_BREAKER = ([\n\r]+)
  TRANSFORMS-extract_indexed_fields= regex_extract_doubled_quoted_av_pairs,
  regex_extract_single_quoted_av_pairs, regex_extract_unquoted_av_pairs
With the fields automatically converted into indexed fields via REGEX, you can do computation on our log file entirely with tstats, providing high speed computation. Note that over precision in the numeric values will bloat the size TSIDX file due to high cardinality. When dealing with high precision metrics, indexes are superior as they store numbers as numbers.

### Conduct complex and selective encryption routing

You might need to obfuscate data prior to storage in Splunk Enterprise, but in some scenarios, still give the possibility to reverse the obfuscation. General reporting can occur on obfuscated "low security" datasets, but a select few can be granted access to "high security" datasets to perform the "reversal". This also allows for different retentions, where for the first 30 days reversal is possible, however, after that, the reversal key is removed. This can be useful in compliance and regulatory use-cases, such as for the financial and health industries, and GDPR. Using INGEST_EVAL and CLONE_SOURCETYPE makes this possible.

```plaintext
• props.conf: [v:email:data:orig] TRANSFORMS-clone_data = v_hash_make_clone, v_hash_make_mask
[v:email:data:reference_map] TRANSFORMS-make_map_reference = v_hash_make_map_reference

• transforms.conf: [v:hash_make_clone] REGEX = (^*)
CLONE_SOURCETYPE = v:email:data:reference_map
[v:hash_make_mask] INGEST_EVAL = email_hash=sha256(replace(_raw, "^(.*)email=(\S+)(.*)", "\3")), _raw=replace(_raw, "^(.*)email=(\S+)(.*)", "\1@email_hash":"\3"")
# this transform routes data, and emits reference map _raw for "high security" index
[v:hash_make_map_reference] INGEST_EVAL = index=secure, queue=if(match(_raw, "email="), "indexQueue", "nullQueue"), email_hash=sha256(replace(_raw, "^(.*)email=(\S+)(.*)", "\3")), _raw="hash=".email_hash."email=".replace(_raw, "^(.*)email=(\S+)(.*)", "\2"), "\1"
```

### Next steps

The content in this article comes from a [Conf20 Talk](#), one of the thousands of Splunk resources available to help users succeed.