Exporting Splunk Data at Scale with Exporttool

Table of Contents

exporttool.py

Exporting Splunk Data at Scale with Exporttool. This is a python script that can be run on each Splunk Indexer for the purpose of exporting historical bucket data (raw events + metadata) at scale by balancing the work across multiple CPUs then forwarding to Cribl.

Exporttool also supports the Splunk SmartStore configuration. SmartStore uses AWS S3 API to plug into the remote storage tier. Remote storage options are AWS S3 and S3 API-compliant object stores, including Dell/EMC ECS, NetApp StorageGrid, Pure Storage Flash Blade and SwiftStack. All you need to do is spin up Linux instances with lots of CPUs and memory, mount the AWS S3 (compliant) object store, install free Splunk, install Exporttool/netcat, and export the data. Indexer guids come into play with Splunk's SmartStore config that affect the directory structure within the index but scribl was rewritten to track down .tsidx files within the index you wish to export then uses the parent directory as a target bucket for export.

Supported: On-Prem Splunk using local or SmartStore storage and Splunk Cloud using a SmartStore configuration.

Not-Supported: Splunk Cloud using a non-SmartStore configuration.

Demo Videos

- HD - Exporting Splunk Data at Scale with Scribl (Scribl renamed to Exporttool)
- 4K - Exporting Splunk Data at Scale with Scribl (Scribl renamed to Exporttool)

Background

Exporting large amounts of previously indexed data from Splunk is challenging via the Splunk-supported approaches detailed here:
https://docs.splunk.com/Documentation/Splunk/8.2.6/Search/Exportsearchresults.

The core Splunk binary in every install provides a switch (cmd exporttool) that allows you to export the data from the compressed buckets on the indexers back into their original raw events. You can dump them to very large local csv files or stream them to stdout so a script can redirect over the network to a receiver such as Cribl Stream. The ‘exporttool’ argument has been used by others for quite a while but it isn't well documented.

Assuming that Splunk is installed in /opt/splunk/, the below commands can be applied to a particular bucket in an index called “bots” to export it.

Exporting to stdout:

/opt/splunk/bin/splunk cmd exporttool
/opt/splunk/var/lib/splunk/bots/db/db_1564739504_1564732800_2394 /dev/stdout -csv

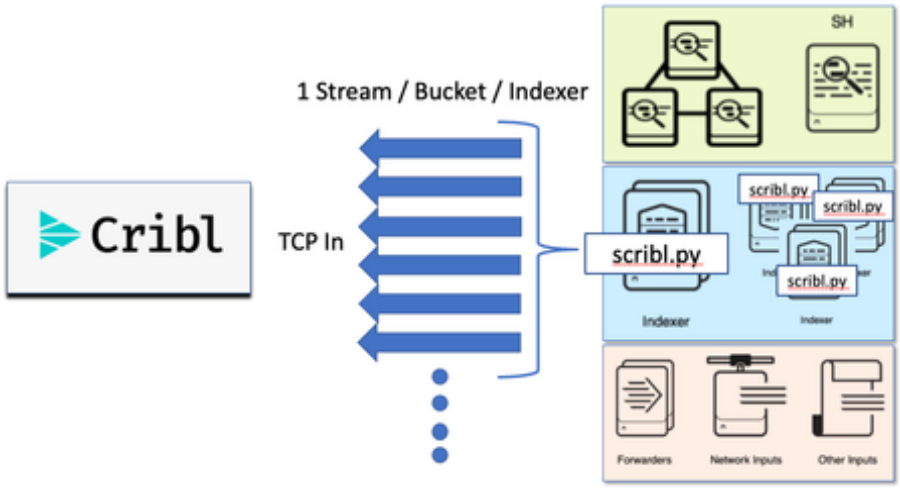
Exporting to a local csv file:

/opt/splunk/bin/splunk cmd exporttool
/opt/splunk/var/lib/splunk/bots/db/db_1564739504_1564732800_2394
/exports/bots/db_1564739504_1564732800_2394.csv -csv

There will be many buckets so some poor soul will need to build a script to export all or some of the buckets and some sort of parallelization should be used to speed the process up. The exported data will be very large (uncompressed, 3-20x) compared to the size of the individual buckets that make up the index!

Requirements

Splunk stores its collected data on the indexers within the "Indexing Tier" as detailed below. The data is compressed and stored in a collection of time series buckets that reside on each indexer or SmartStore Object Store. Each bucket contains a rawdata journal, along with associated tsidx, and metadata files. The search heads access these buckets and it's very rare for someone to access them directly from the indexer CLI unless there is a need to export data to retrieve the original raw events. We will use the indexer CLI to export the original raw events (per bucket and in parallel) as well as a few other pieces of important metadata as detailed below.



For a deeper dive into how Splunk indexes data, see this: https://docs.splunk.com/Documentation/Splunk/latest/Indexer/HowSplunkstoresindexes

You will need:

- CLI access to each Linux indexer with the index/buckets that need to be exported. This process only applies to on-prem or non-SplunkCloud deployments.
- To install nc (netcat) on each indexer to act as the transport mechanism until we have enough demand to build the transport into the script.
- To make sure outbound communication from each indexer to the Cribl Worker TCP port is open.

Frozen Data

The Splunk exporttool switch that Exporttool depends on requires a complete hot/warm/cold directory containing all of the metadata files and the journal.gz file. When buckets are moved to a frozen archive, all of the metadata files are removed with only the journal.gz file remaining. Exporttool can not extract raw events from frozen archives.

Buckets must first be "thawed" as described here. It's a straightforward process of copying the frozen buckets somewhere and running a "splunk rebuild" for each bucket to recreate the metadata. Depending on where you place the journal.gz file, you may see errors where the rebuild command notices that you are thawing into an index (directory) that the splunk binary does not know about and can be ignored. Here is the error: "*ERROR ProcessTracker - (subchild_43__RollFixMetadata) IndexConfig - Asked to check if idx= is an index with a remote storage, but that index does not exist on the system or is disabled*". Exporttool can be run against this thawed data.

Technical

Scale

We achieve scale for large volumes of data by processing buckets in parallel across as many CPUs as you would like to dedicate AND by streaming the data directly from disk with a single read to Cribl without ever having to write extracted/uncompressed event data to disk. Extracting/uncompressing the event data to disk would result in enormous disk IO bottlenecks and disk space consumption.

Disk speed (IOPS) and the number of CPUs are generally your limiting factors on the indexers. While disk speed is a factor, it's usually not a factor in the overall scale picture because Splunk indexers will usually have high IOPs capabilities. You can monitor the linux processes to get a feel for whether Exporttool processes are in SLEEP or RUN mode. If they spend most of their time in SLEEP mode, they are being throttled by disk, network, Cribl workers, etc and adding more CPUs will probably not buy you more speed.

The Exporttool script running on your indexers and Cribl Stream workers are built to scale and will usually not be your bottleneck. Your bottlenecks will almost certainly be bandwidth

constraints between your indexers and your final destination. Depending on where you deploy your Cribl Stream workers, that bandwidth bottleneck might exist between the indexers and Cribl workers or between your Cribl workers and the final destination. If you happen to have unlimited bandwidth, you will likely find your next bottleneck to be the ingest rate at your destination platform.

Exported Data Format

The exported data will be csv formatted with a header followed by the individual events. It's important to call out that these events are often multiline events with the most common example being windows logs. The below events are examples that are generated by Splunk and then passed via stdin to the exporttool.py script.

The _raw field contains the original event and the other fields were captured/created during ingest. _time is the time extracted from the event which will be the primary time reference used by the destination analytics platform. The sourcetype field will likely be what is used by the destination to determine how to parse and where to route the event.

As seen below, the exported data contains important fields (_time, source, sourcetype, and raw) that need to be broken out via a "scribl" pipeline.

Example:

"_time",source,host,sourcetype,"_raw","_meta"

1564734905,"source::10.1.1.1","host::hogshead","sourcetype::fgt_utm","date=2019-08-02 time=08:35:05 devname=hogshead devid=FGT60D4614044725 logid=1059028704 type=utm subtype=app-ctrl eventtype=app-ctrl-all level=information vd=root appid=38131 user="""" srcip=10.1.1.103 srcport=51971 srcintf=""internal"" dstip=172.217.11.227 dstport=443 dstintf=""wan1"" profiletype=""applist"" proto=6 service=""HTTPS"" policyid=1 sessionid=594789 applist=""default"" appcat=""General.Interest"" app=""Google.Accounts"" action=pass hostname=""ssl.gstatic.com"" url=""/"" msg=""General.Interest: Google.Accounts,"" apprisk=elevated","_indextime::1564734907 _subsecond::.000 syslog-server::jupiter severity::notice facility::user punct::""=--_=::_=_=_=_=_=-_=--_=_=_=_=\""\""_=..._=_=\""\""_=..._="""

1564734846,"source::WinEventLog:Microsoft-Windows-PowerShell/Operational","host::titan","sourcetype::XmlWinEventLog:Microsoft-Windows-PowerShell/Operational","<Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event'><System><Provider Name='Microsoft-Windows-PowerShell' Guid='{A0C1853B-5C40-4B15-8766-3CF1C58F985A}'/><EventID>4103</EventID><Version>1</Version><Level>4</Level><Task>106</Task><Opcode>20</Opcode><Keywords>0x0</Keywords><TimeCreated SystemTime='2019-08-02T08:34:06.167139700Z'/><EventRecordID>5968761</EventRecordID><Correlation ActivityID='{135BC459-4718-0000-AAD1-74131847D501}'/><Execution ProcessID='3016'

ThreadID='3720'/><Channel>Microsoft-Windows-PowerShell/Operational</Channel><Computer>titan.thirstyberner.com</Computer><Security UserID='S-1-5-18'/></System><EventData><Data Name='ContextInfo'>	Severity = Informational

	Host Name = Default Host

	Host Version = 4.0

	Host ID = 89297903-4c6b-4e9d-b0a4-49c76b2c36ae

	Engine Version = 4.0

	Runspace ID = 657f43dd-6fb5-42c9-8b93-154f3a1e53dd

	Pipeline ID = 1

	Command Name = Out-Null

	Command Type = Cmdlet

	Script Name =

	Command Path =

	Sequence Number = 1565734

	User = THIRSTYBERNER\SYSTEM

	Shell ID = Microsoft.PowerShell

</Data><Data Name='UserData'></Data><Data Name='Payload'>ParameterBinding(Out-Null): name=""InputObject""; value=""True""

</Data></EventData></Event>","_indextime::1564734908 punct::<_='://../////'><><_='--'_='{----}'/><></><></><><"

Cribl Stream Routing

The routing of data as you need it into one or more destination formatted as you need it to be is one of the most important use cases of Cribl Stream. You will likely have indexes you wish to export which contain multiple sourcetypes. The Splunk sourcetype assignment is contained in every event that Cribl Stream processes. You can filter, optimize, route, etc each of those sourcetypes however you choose. We used Splunk's Boss of the SOC dataset for testing because it is real-world security data ingested during a live campaign and it contains a very diverse

collection of data (souretypes) to best identidy unexpected bugs (multiline events, gigantic events, etc). The github repo details over 100 sourcetypes available in the BOTSv3 dataset.

exporttool.py

The scribl.py script is available in this repo.

Example Usage:

./scribl.py -h

usage: scribl.py [-h] [-t] [-n NUMSTREAMS] [-l LOGFILE] [-et EARLIEST] [-lt LATEST] [-kv KEYVAL [KEYVAL ...]] [-b] -d DIRECTORY -r REMOTEIP -p REMOTEPORT

This is to be run on a Splunk Indexer for the purpose of exporting buckets and streaming their contents to Cribl Stream

optional arguments:
-h, --help show this help message and exit
-t, --TLS Send with TLS enabled
-n NUMSTREAMS, --numstreams NUMSTREAMS
Number of parallel stream to utilize
-l LOGFILE, --logfile LOGFILE
Location to write/append the logging
-et EARLIEST, --earliest EARLIEST
Earliest epoch time for bucket selection
-lt LATEST, --latest LATEST
Latest epoch time for bucket selection
-kv KEYVAL [KEYVAL ...], --keyval KEYVAL [KEYVAL ...]
Specify key=value to carry forward as a field in
addition to _time, host, source, sourcetype, and _raw.
Can specify -kv multiple times
-b, --bucketname Add bucket=<bucketname> to the output

required named arguments:
-d DIRECTORY, --directory DIRECTORY
Source directory pointing at the index
-r REMOTEIP, --remoteIP REMOTEIP
Remote address to send the exported data to
-p REMOTEPORT, --remotePort REMOTEPORT
Remote TCP port to be used

scribl.py -d /opt/splunk/var/lib/splunk/bots/ -r 14.2.39.121 -p 20000 -t -n4 -l /tmp/scribl.log -et 1564819155 -lt 1566429310

On No! My Splunk license expired!

Worry not, my friend. When the enterprise license expires, Splunk customers are free to use the 60-day trial or even the free version of Splunk to perform the export. Sanity check my claim here: https://docs.splunk.com/Documentation/Splunk/9.0.0/Admin/MoreaboutSplunkFree .

We don't care about indexing new data and we don't care about distributed search since we will use the trial/free Splunk binary in a standalone manner on each of the indexers that have data we need to migrate. Just install trial/free Splunk on top of or alongside the existing install and point scribl.py at your splunk binary and the directory containing the buckets you need to export.

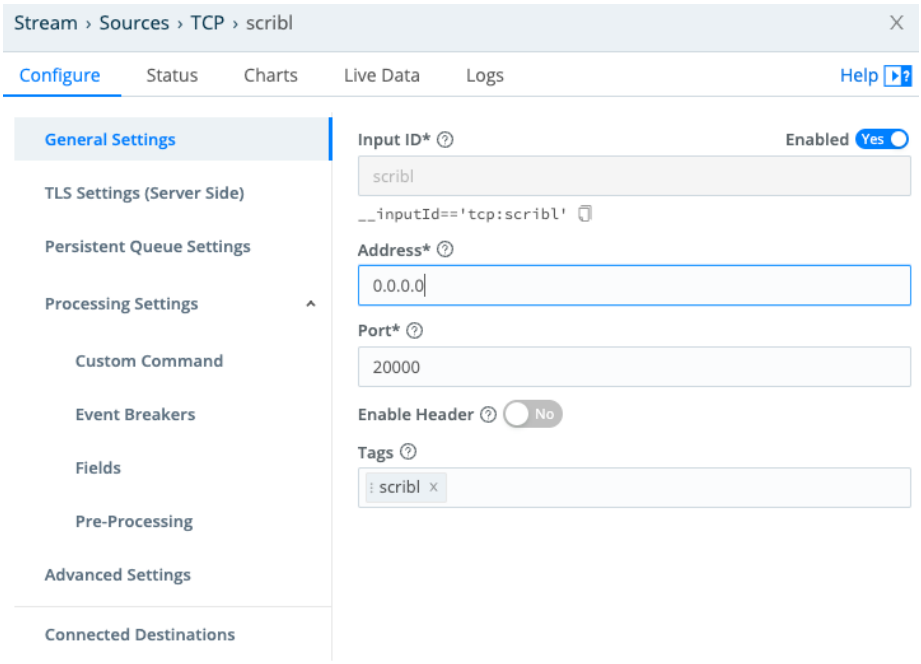
Cribl Stream Config

You can get started instantly with Cribl Cloud or even using the Cribl Free license option but keep in mind daily ingest limits (very generous) and # of cores (also very generous at 10) that can be used may factor into a full scale data export. If you choose to install Cribl Stream on-prem on in your own cloud, the documentation is your friend and will get you going quickly.

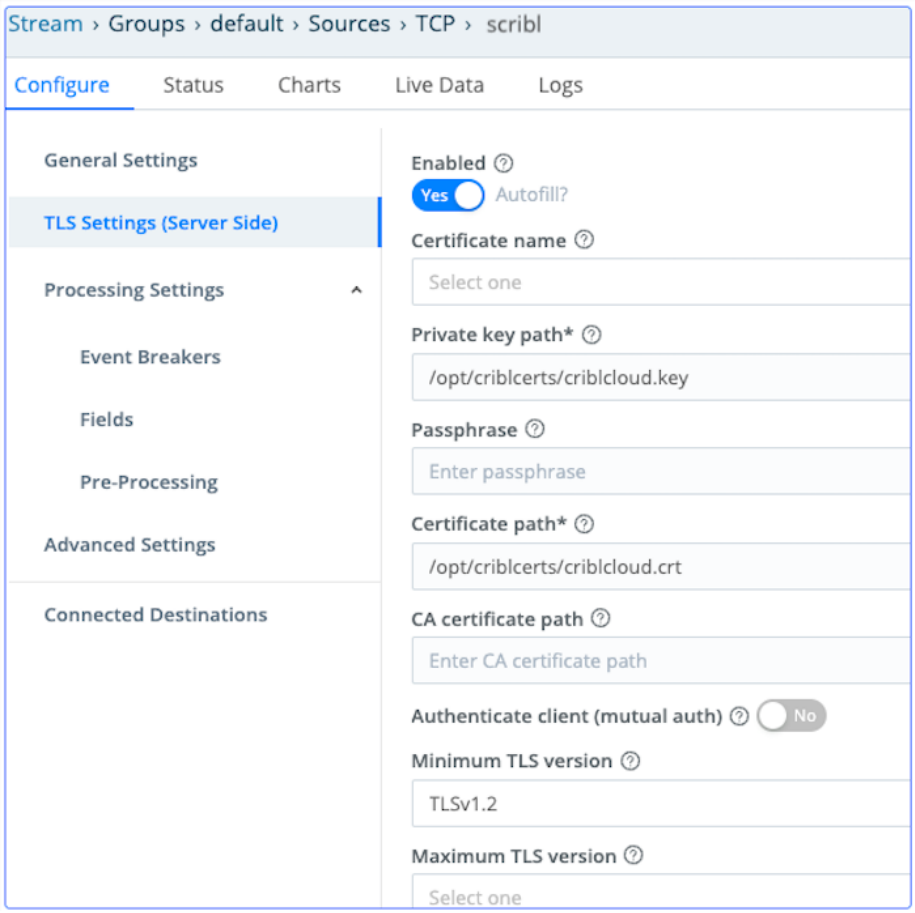
Once you have satisfied the above requirements (CLI, nc, and firewall) on your Splunk indexers, grab the exporttool.py script from the github repo and copy it over to each indexer. The only thing in the script that is hard coded is the default install location of Splunk (/opt/splunk) which you can easily modify if you are running a non-default config. Keep in mind that we are running the script directly on the Splunk indexers and a python binary is kept under $SPLUNK_HOME/bin.

The github repo also contains a Cribl Pack for Scribl which contains a few sanity checks dealing with possible unexpected large events that exceed line breakers and a couple data transforms. Download the pack and load it upas described below.

Add a new Source



Enable and Config TLS (recommended):



Create a new Event Breaker

See the below caveat regarding event sizes. You may need to increase the Max Event Bytes (default=500000) depending on your local sourcetype and the associated event sizes. I changed it to 500000 in this example.

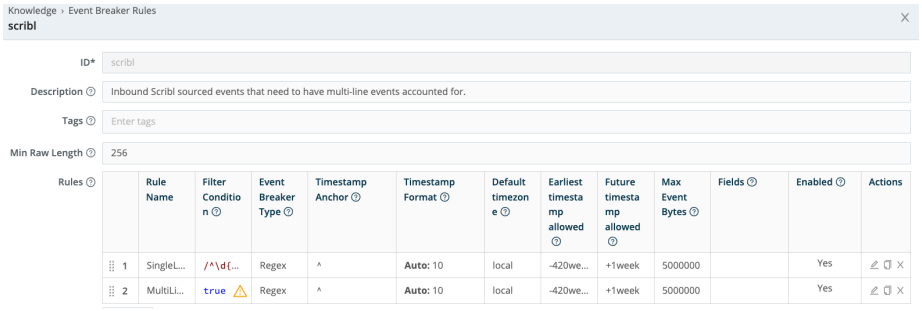
Processing/Knowledge → Knowledge → Event breaker Rules → +Add Ruleset

Click the Manage as JSON button, paste the below, , click OK, then Click Save.

```
{
  "lib": "custom",
```

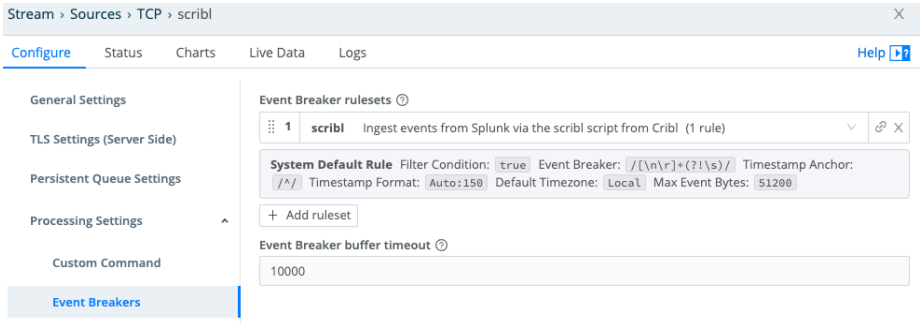
```
  "minRawLength": 256,
  "id": "scribl",
  "rules": [
    {
      "condition": "/^\\d{10}/.test(_raw) && _raw.includes('punct::') ==
true",
      "type": "regex",
      "timestampAnchorRegex": "/^/",
      "timestamp": {
        "type": "auto",
        "length": 10
      },
      "timestampTimezone": "local",
      "timestampEarliest": "-420weeks",
      "timestampLatest": "+1week",
      "maxEventBytes": 5000000,
      "disabled": false,
      "parserEnabled": false,
      "eventBreakerRegex": "/[\\n\\r]+(?!\\s)/",
      "name": "SingleLine"
    },
    {
      "condition": "true",
      "type": "regex",
      "timestampAnchorRegex": "/^/",
      "timestamp": {
        "type": "auto",
        "length": 10
      },
      "timestampTimezone": "local",
      "timestampEarliest": "-420weeks",
      "timestampLatest": "+1week",
      "maxEventBytes": 5000000,
      "disabled": false,
      "parserEnabled": false,
      "eventBreakerRegex": "/[\\n\\r]+(?=\\d{10},\\\")/",
      "name": "MultiLine"
    }
  ],
  "description": "Inbound Scribl sourced events that need to have multi-line
events accounted for."
}
```

Your new Scribl Event Breaker should contain 2 rules and look like this:



For performance reasons, you should notice that the event breaker searches top-down and we are first checking to see if both the epoch timestamp and “punct::” string appears in the event indicative of a single line event from scribl. If so, it’s a nice performant line break. If it’s not a single-line event, we fall back into a line breaker which is a little less performant since it needs to effectively reassemble multiple inbound lines into a single event.

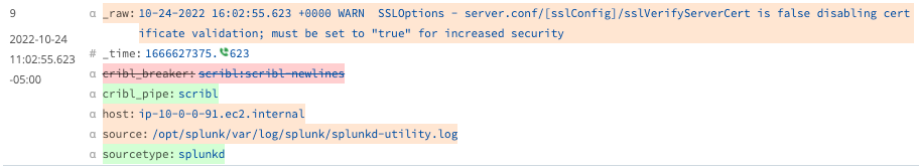
Attach the new Event Breaker to your Source



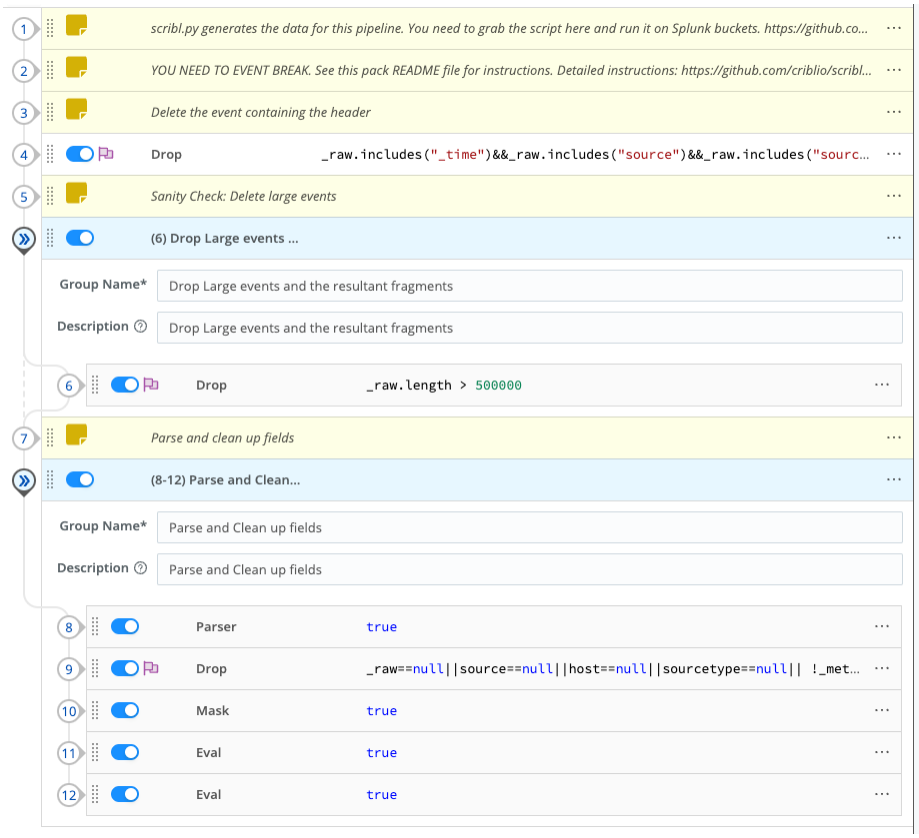
Create a pipeline to break out important fields

Sample data BEFORE the pipeline (In)

Sample data AFTER the pipeline (Out)



Download then import the [Cribl Pack for Scribl] into Stream and examine the functions in the Scribl pipelines.



Line 4: Drops the event containing the header that is sent by Splunk Exporttool at the beginning of every bucket export.

Line 6: If an event shows up that is greater than the line breaker max size, drop it.

Line 8: Breaks the event into fields based on the csv formatting. Important: if you add additional fields using the scribl arg -kv option, you need to account for the additional fields here.

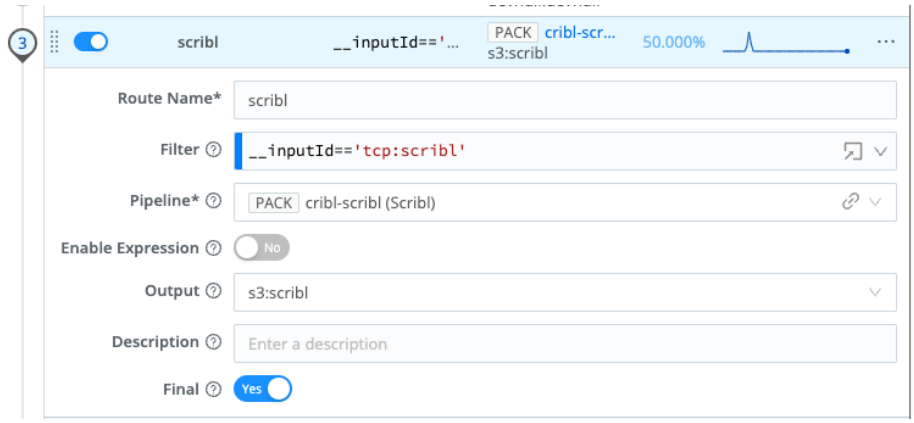
Line 9: If the line breaker runs into a field greater than it's max size, it's gonna break it and leave one or more dangling sections after the max size which need to be ignored.

Line 10: Cleans up field names.

Lines 11 and 12: Drop fields.

Add a new route, attach the pipeline, add the destination

Send the data from your tcp input, route it through the pack, mark it as final, and redirect to the destination output of your choosing.



Cribl Stream Worker Load-Balancers

Scribl will create a TCP connection for each bucket that is exported and if you are instructing scribl to use X CPUs to export an index, your load-balancer will see ~X concurrent connects at any given time balancing them across the load-balanced worker nodes. Here are a few guidelines for configuring your load-balancer:

- Configure scribl to send to the external IP or FQDN of the load-balancer. Make sure you use nslookup to test that FQDN to make certain you don't have anything unexpected regarding multiple IPs. This has happened. We ran into an issue where someone configured DNS to return external IP addresses for load-balancers spread across multiple availability zones even though there were only workers in one of them.
- Verify the configured port on the external interface of the load-balancer is the one scribl is sending data to. If you opened up TLS/443, on the load-balancer, make sure you have

"-t -p 443" args being used with scribl. It's perfectly fine to terminate SSL at the load-balancer and have the load-balancer send data to non-TLS ports using a different port number (ex port 20000) on the workers.

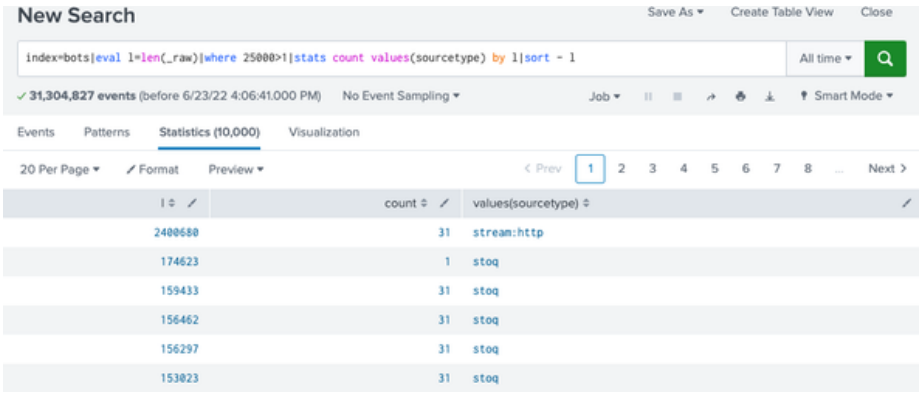
- If you into an issue, you can run scribl directly to a worker node if it can be reached.
- Double-check all firewall rules.
- Instead of using scribl during your testing which is pretty heavy-handed, you can test with this as described here:

Caveats / Comments:

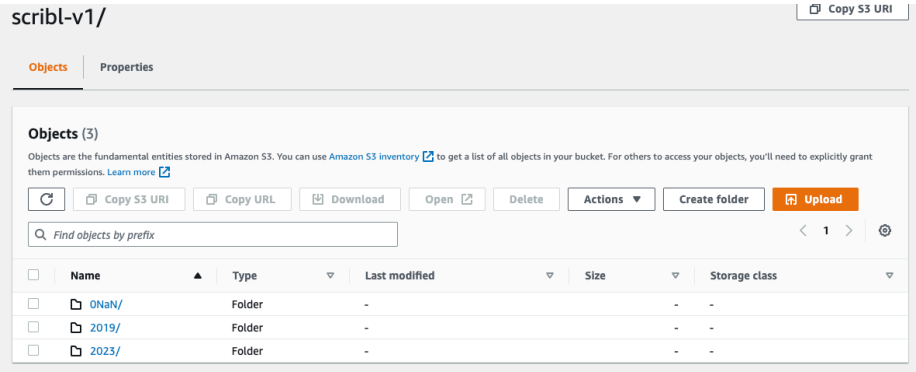
Splunk Event Sizes

You need to pay attention to event sizes in Splunk as it pertains to the Event breaking in Cribl. As noted above in the Event Breaker screenshot, the max event size has a default setting of 51200 bytes. If you use scribl to send events into Cribl Stream larger than that, things break. Either increase your event-breaking max event size, use the Cribl Stream Pipeline to drop the large events (example: by sourcetype), or do not use scribl to export the buckets containing the large events.

Here is a quick Splunk search highlighting the large events that need to be dealt with:



If you happen to send events into Cribl larger than the Max Event Size configured in the source-specific event breaker, you will see malformed events being sent to your destination. If you are writing to an object store that has a Partitioning Expression utilizing the timestamp (directories based on YY, MM, and DD), you will see directories similar to the below named 0NaN (Not-a-Number).



To safeguard against these you can create a function to drop events larger that a specific size (`_raw.length > 500000`) or drop the event if the _time field is not a number (`isNaN(_time)`).

Bottlenecks

As mentioned above, the bottleneck you will most likely run into will be bndwidth in your data path or ingest rate at the final destination. Anything you can do to parallelize that final write will pay dividends. For example, you may want to use Cribl Stream's Output Router to write to multiple S3 buckets based on the original Splunk Index or Sourcetype if bandwidth is not your bottleneck.

Index Clusters and replicated buckets

See this for some background on what happens with bucket replication. This is the important part: *"The indexer cluster replicates data on a bucket-by-bucket basis. The original bucket copy and its replicated copies on other peer nodes contain identical sets of data, although only **searchable** copies also contain the index files."*

The name of replicated buckets start with "rb_" which scribl ignores preventing the duplicate indexing of replicated buckets within the index cluster. Scribl only operates on buckets whose names start with "db_".

Hot Buckets

Hot buckets are ignored. Similar to how replicated bucket names begin with "_rb", hot buckets start with "hot_" and they are both ignored within a simple if statement in the script. If you have a use case where you need to export hot buckets, feel free to modify the if statement.

Dynamic Data Self Storage (DDSS)

If you are leveraging the Splunk Dynamic Data Self Storage option, you should be able to mount your private buckets containing the data and mount it just as we do for the Smart Store config above.

Splunk SmartStore Support

SmartStore is an indexer capability that provides a way to use remote object stores, such as Amazon S3, Google GCS, or Microsoft Azure Blob storage, to store indexed data. At this point in time, Scribl has only been tested on AWS S3 Object Stores. The below process was used to test Scribl.

S3 Object Store

Create your S3 Object Store (bucket) in AWS S3 and make sure your indexer has the proper permission to access the store. In this example, we create an IAM role granting proper S3 permissions and attached it to an indexer EC2 instance.

Upload a file to the bucket and validate your permission with:

```
aws s3 ls s3://smart-store-scribl
```

SmartStore config on the indexer /opt/splunk/etc/system/local/indexes.conf:

```
[default]
remotePath=volume:ecs_store/$_index_name

[volume:ecs_store]
storageType = remote
path = s3://smart-store-scribl/scribl
```

Restart Splunk. If you need to force a roll from hot to warm buckets in Splunk, use the below command to roll the _internal index:

```
/opt/splunk/bin/splunk _internal call /data/indexes/_internal/roll-hot-
buckets
```

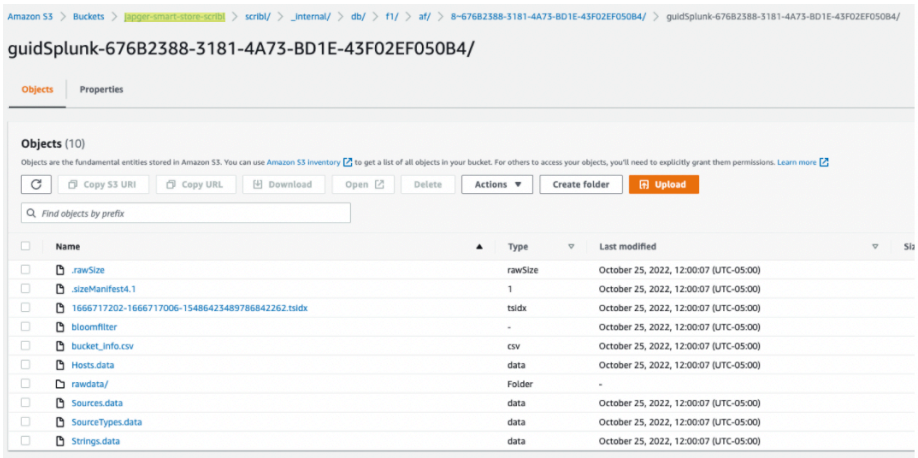
You will notice that the local indexer bucket names may differ slightly from the standard config when using Smart Store but they still start with db_.

```
ls -l /opt/splunk/var/lib/splunk/_internaldb/db
total 16
-rw------- 1 root root   10 Oct 24 16:03 CreationTime
drwx--x--- 3 root root  271 Oct 25 16:55 db_1666627667_1666627364_0_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root 4096 Oct 25 16:55 db_1666634954_1666627665_1_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root  271 Oct 25 16:55 db_1666635850_1666634954_2_676B2388-
3181-4A73-BD1E-43F02EF050B4
```

```
drwx--x--- 3 root root  271 Oct 25 16:55 db_1666636181_1666635850_3_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root  271 Oct 25 16:55 db_1666639559_1666636180_4_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root  295 Oct 25 16:55 db_1666639746_1666639558_5_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root  271 Oct 25 16:55 db_1666716888_1666638001_6_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root  271 Oct 25 16:57 db_1666717006_1666716888_7_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root  295 Oct 25 17:00 db_1666717202_1666717006_8_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 3 root root  294 Oct 26 13:57 db_1666792676_1666717204_9_676B2388-
3181-4A73-BD1E-43F02EF050B4
drwx--x--- 2 root root    6 Oct 24 16:03 GlobalMetaData
drwx--x--- 3 root root 4096 Nov  2 21:04 hot_v1_10
drwx--x--- 3 root root 4096 Nov  2 21:04 hot_v1_11
```

S3 Object Store Structure

The directory structure within the S3 bucket is different than what is local to the indexer and will resemble something similar to the below. This will be important as we will be mounting this bucket within Linux and pointing Scribl at this new structure to access buckets for exporting events. The index name (_internal in the below example) is the piece scribl needs to see. You might want to mount the entire ‘scribl' directory in the below example to make sure you have access to all indexes in this object store.



Mount the Smart Store bucket and export

We opted to use S3fs-fuse to mount the S3 bucket in Linux. Follow the direction in the preceding link to install S3fs and mount the directory. Once the directory is mounted, use scribl by pointing

it at the index and scribl will figure out where the buckets are and filter as needed if you specified time constraints as arguments.

Since scribl is only performing reads against the S3fs mounted directories, it is highly recommended that you disable caching. We have run into environments where caching was not disabled and the default caching directory or /tmp was being filled up causing errors and performance issues.

Note: listing directories on an S2fs mounted system is SLOW. If you are using scribl against a bucket with a very large number of tsidx files, you may consider modifying the scribl script to obtain the collection of buckets to be exported from a flat file instead of the directory traversal process. It was taking roughly 10 minutes to perform the bucket discovery each time scribl was initiated against an index containing ~250k buckets.

To Do:

-
-
-
-
- Look into getting transport built into the script, including TLS. Netcat works so well, though.
- Provide an option for using a flat file containing txidx files or buckets to be exported when SmartStore is involved to get around the SLOW bucket discovery process when using a mounted (S3fs) drive.
- Consider adding et/lt for event time in addition to bucket selection. This would help prevent duplicate data issues where buckets overlap.

Deleted: Add min and max times to determine which buckets should be exported (complete 6/2022)

Deleted: Add context as scribl pertains to index clusters and replicated buckets

Deleted: Build SmartStore support

Deleted: Build a Scribl Pack to speed deployment and capture lessons learned as more deployments are completed