Forward-Looking Statements

This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's current expectations, estimates and assumptions based on the information currently available to us. These forward-looking statements are not guarantees of future performance and involve significant risks, uncertainties and other factors that may cause our actual results, performance or achievements to be materially different from results, performance or achievements expressed or implied by the forward-looking statements contained in this presentation.

For additional information about factors that could cause actual results to differ materially from those described in the forward-looking statements made in this presentation, please refer to our periodic reports and other filings with the SEC, including the risk factors identified in our most recent quarterly reports on Form 10-Q and annual reports on Form 10-K, copies of which may be obtained by visiting the Splunk Investor Relations website at www.investors.splunk.com or the SEC's website at www.sec.gov. The forward-looking statements made in this presentation are made as of the time and date of this presentation. If reviewed after the initial presentation, even if made available by us, on our website or otherwise, it may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statement based on new information, future events or otherwise, except as required by applicable law.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. We undertake no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.

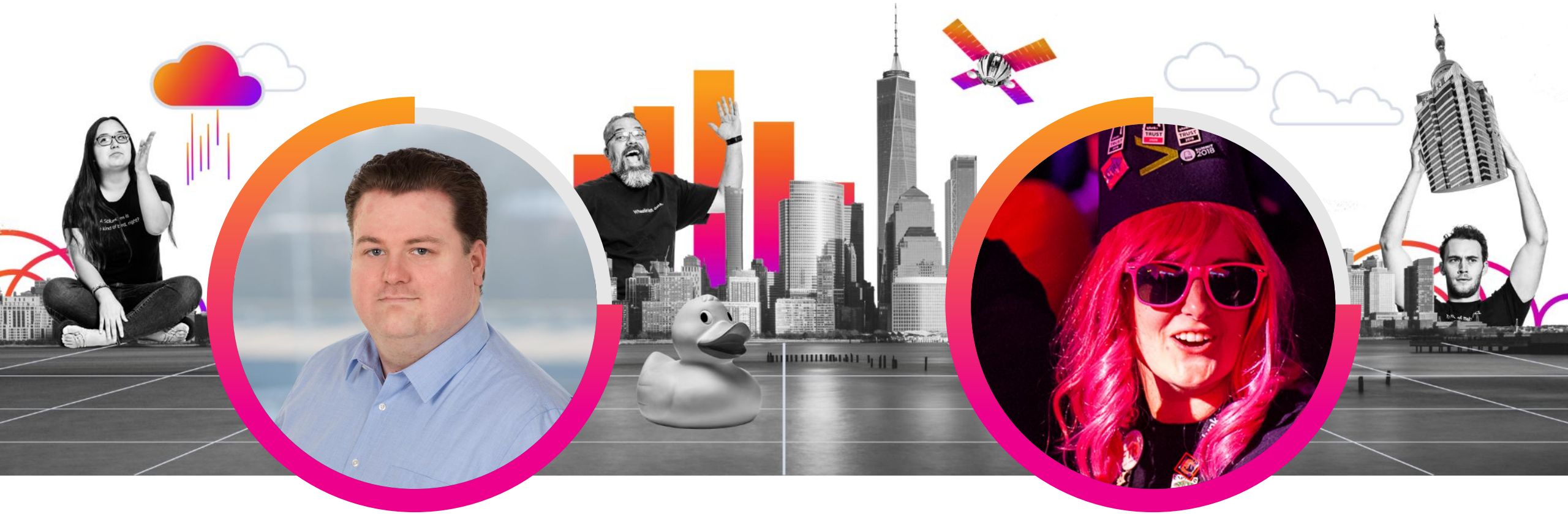
splunk> .conf21

Clara-Fication: More Tstats for Your Buckets

TRU1133B

Clara Merriman
Senior Splunk Engineer | Splunk

Martin Müller
Principal Consultant | Consist



Martin Müller

Principal Consultant
Consist Software Solutions GmbH

Clara Merriman

Senior Splunk Engineer | Splunk

splunk> .conf21

Agenda

```
| tstats summariesonly=t
values(conf.section)
from datamodel=conf
where conf.talk=TRU1133
by conf.section_number
```

1) Quick Reintroduction to tstats
   allow_old_summaries = true

2) Cool Things tstats Can Do
   TERM(cool=1)

3) Converting Searches to tstats
   | convert

4) “Pivot” Job Inspector
   | pivot

5) Optional Arguments
   append = true

splunk> .conf21

(Re-)Intro to | tstats

splunk> .conf21

Three Types of Stats-y Searches*

Event searches:

```
index=web
| stats count by host
```

- infinite flexibility
- speed penalties

Index-based tstats:

```
| tstats count
where index=web
by host
```

- limited to index-time
- very fast

Datamodel-based tstats:

```
| tstats count
from datamodel=Web
by host
```

- some flexibility at the expense of prep & rebuild efforts
- very fast with DMA

*There's also mstats for metrics indexes, that's a topic for another talk...

Three Types of Stats-y Searches*

Event searches:

```
index=web
| stats count by host
```

- infinite flexibility
- speed penalties

Index-based tstats:

```
| tstats count
where index=web
by host
```

- limited to index-time
- very fast

Datamodel-based tstats:

```
| tstats count
from datamodel=Web
by host
```

- some flexibility at the expense of prep & rebuild efforts
- very fast with DMA

*There's also mstats for metrics indexes, that's a topic for another talk...

Three Types of Stats-y Searches*

Event searches:

```
index=web
| stats count by host
```

- infinite flexibility
- speed penalties

Index-based tstats:

```
| tstats count
where index=web
by host
```

- limited to index-time
- very fast

Datamodel-based tstats:

```
| tstats count
from datamodel=Web
by host
```

- some flexibility at the expense of prep & rebuild efforts
- very fast with DMA

*There's also mstats for metrics indexes, that's a topic for another talk...

Cool Examples

Filtering

- Filter by indexed fields

```
| tstats count where index=_internal source=*metrics.log*
```

- Filter by simple indexed terms

```
| tstats count where index=_internal source=*metrics.log* typingqueue
```

- Filter by complex indexed terms

```
| tstats count where index=_internal source=*metrics.log* "blocked=true"
```

- 😥 Error in 'TsidxStats': WHERE clause is not an exact query

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true)
```

- 😅 `TERM()` forces Splunk to treat `blocked=true` as one indexed term, ignoring the minor segmenter =

Filtering

- Filter by indexed fields

```
| tstats count where index=_internal source=*metrics.log*
```

- Filter by simple indexed terms

```
| tstats count where index=_internal source=*metrics.log* typingqueue
```

- Filter by complex indexed terms

```
| tstats count where index=_internal source=*metrics.log* "blocked=true"
```

- 😥 Error in 'TsidxStats': WHERE clause is not an exact query

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true)
```

- 😅 `TERM()` forces Splunk to treat `blocked=true` as one indexed term, ignoring the minor segmenter =

Filtering

- Filter by indexed fields

```
| tstats count where index=_internal source=*metrics.log*
```

- Filter by simple indexed terms

```
| tstats count where index=_internal source=*metrics.log* typingqueue
```

- Filter by complex indexed terms

```
| tstats count where index=_internal source=*metrics.log* "blocked=true"
```

- 😥 Error in 'TsidxStats': WHERE clause is not an exact query

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true)
```

- 😅 `TERM()` forces Splunk to treat `blocked=true` as one indexed term, ignoring the minor segmenter =

Filtering

- Filter by indexed fields

```
| tstats count where index=_internal source=*metrics.log*
```

- Filter by simple indexed terms

```
| tstats count where index=_internal source=*metrics.log* typingqueue
```

- Filter by complex indexed terms

```
| tstats count where index=_internal source=*metrics.log* "blocked=true"
```

- 😥 ❗ Error in 'TsidxStats': WHERE clause is not an exact query

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true)
```

- 😅 `TERM()` forces Splunk to treat `blocked=true` as one indexed term, ignoring the minor segmenter =

PREFIX() Fun

- This does not work, there is no indexed field `name`:

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true) by name
```

- `PREFIX(name=)` will ad-hoc-interpret the indexed term `name=typingqueue` as a key-value pair (v8.0+)

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true) by PREFIX(name=)
```

- 🤯

```
| tstats count where index=_internal source=*metrics.log* TERM(group=queue) by PREFIX(name=)
```

Last 24 hours

✓ 19,134 events (07/07/2021 16:00:00.000 to 08/07/2021 16:06:45.000) No Event Sampling Job Smart Mode

Events Patterns Statistics (18) Visualization

20 Per Page Format Preview

name=	count
aeq	1063
aggqueue	1063
aq	1063
auditqueue	1063

PREFIX() Fun

- This does not work, there is no indexed field `name`:

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true) by name
```

- `PREFIX(name=)` will ad-hoc-interpret the indexed term `name=typingqueue` as a key-value pair (v8.0+)

```
| tstats count where index=_internal source=*metrics.log* TERM(blocked=true) by PREFIX(name=)
```

- 🤯

```
| tstats count where index=_internal source=*metrics.log* TERM(group=queue) by PREFIX(name=)
```

Last 24 hours

✓ **19,134 events** (07/07/2021 16:00:00.000 to 08/07/2021 16:06:45.000) No Event Sampling Job Smart Mode

Events Patterns **Statistics (18)** Visualization

20 Per Page Format Preview

name=	count
aeq	1063
aggqueue	1063
aq	1063
auditqueue	1063

PREFIX() Magic

- "Traditional" search to chart queue sizes:

```
index=_internal source=*metrics.log* TERM(group=queue)
| timechart span=15m max(current_size_kb) by name
```

Observed 80k Events/s per indexer in a real-world environment… not bad, but can tstats help?

- PREFIX() works for aggregations too!

```
| tstats max(PREFIX(current_size_kb=)) as max_size_kb
  where index=_internal source=*metrics.log* TERM(group=queue)
  by _time span=15m PREFIX(name=)
| timechart span=15m max(max_size_kb) by "name="
```

Same environment returned 2-3M EPS per indexer, 30x speedup

- But remember, your raw event layout matters: `name=typingqueue` works while `name="typingqueue"` breaks `PREFIX()` due to segmentation

PREFIX() Magic

- "Traditional" search to chart queue sizes:

```
index=_internal source=*metrics.log* TERM(group=queue)
| timechart span=15m max(current_size_kb) by name
```

Observed 80k Events/s per indexer in a real-world environment… not bad, but can tstats help?

- PREFIX() works for aggregations too!

```
| tstats max(PREFIX(current_size_kb=)) as max_size_kb
  where index=_internal source=*metrics.log* TERM(group=queue)
  by _time span=15m PREFIX(name=)
| timechart span=15m max(max_size_kb) by "name="
```

Same environment returned 2-3M EPS per indexer, 30x speedup

- But remember, your raw event layout matters: `name=typingqueue` works while `name="typingqueue"` breaks `PREFIX()` due to segmentation

PREFIX() Magic

- "Traditional" search to chart queue sizes:

```
index=_internal source=*metrics.log* TERM(group=queue)
| timechart span=15m max(current_size_kb) by name
```

Observed 80k Events/s per indexer in a real-world environment… not bad, but can tstats help?

- PREFIX() works for aggregations too!

```
| tstats max(PREFIX(current_size_kb=)) as max_size_kb
  where index=_internal source=*metrics.log* TERM(group=queue)
  by _time span=15m PREFIX(name=)
| timechart span=15m max(max_size_kb) by "name="
```

Same environment returned 2-3M EPS per indexer, 30x speedup

- But remember, your raw event layout matters: `name=typingqueue` works while `name="typingqueue"` breaks `PREFIX()` due to segmentation

From Search to tstats

splunk> .conf21

Basic Conversion

Event Search

```
index=main
| stats count by sourcetype
```

tstats Search

```
| tstats count where index=main by
sourcetype
```

Slightly More Complex Conversion

Event Search

```
index=pan_logs
| stats count
```

tstats Search

```
| tstats count where index=pan_logs
```

Slightly More Complex Conversion

Event Search

```
index=pan_logs
| stats count by host
```

tstats Search

```
| tstats count where index=pan_logs
by host
```

Slightly More Complex Conversion

Event Search

```
index=pan_logs
| timechart span=1d count by host
```

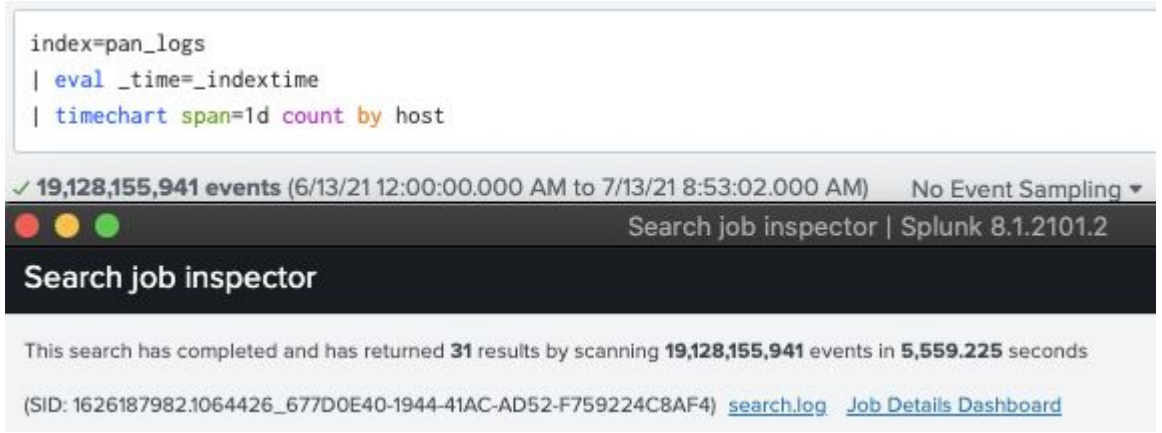
tstats Search

```
| tstats prestats=t count where
index=pan_logs by _time span=1d host
| timechart span=1d count by host
```

Slightly More Complex Conversion

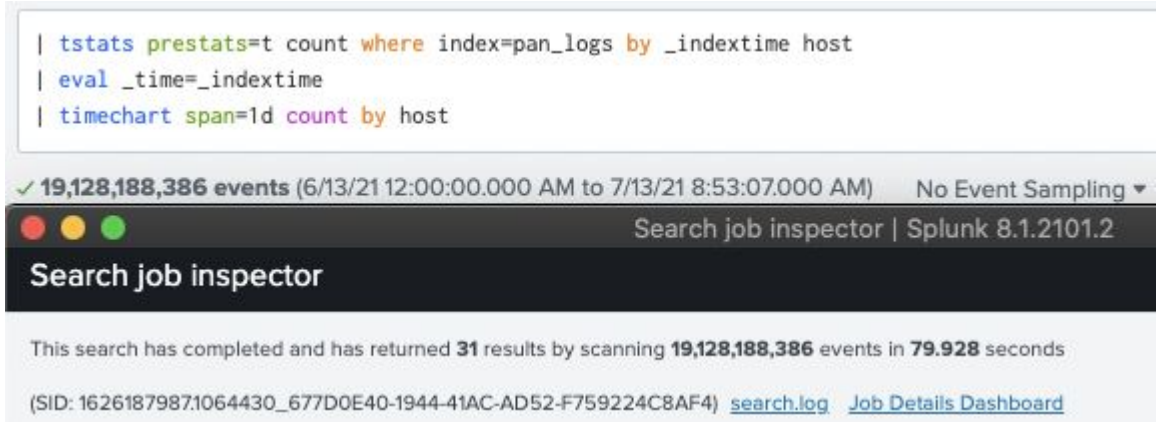
Event Search

```
index=pan_logs
| eval _time=_indextime
| timechart span=1d count by host
```



tstats Search

```
| tstats prestats=t count where
index=pan_logs by _indextime host
| eval _time=_indextime
| timechart span=1d count by host
```



Slightly More Complex Conversion

Event Search

```
index=pan_logs
| eval _time=_indextime
| timechart span=1d count by host
```

```
index=pan_logs
| eval _time=_indextime
| timechart span=1d count by host
```

✓ **19,128,155,941 events** (6/13/21 12:00:00.000 AM to 7/13/21 8:53:02.000 AM) No Event Sampling

Search job inspector | Splunk 8.1.2101.2

Search job inspector

This search has completed and has returned **31** results by scanning **19,128,155,941** events in **5,559.225** seconds

(SID: 1626187982.1064426_677D0E40-1944-41AC-AD52-F759224C8AF4) search.log Job Details Dashboard

70x Faster

stats Search

```
ts prestats=t count where
an_logs by _indextime host
al _time=_indextime
echart span=1d count by host
```

```
| tstats prestats=t count where index=pan_logs by _indextime host
| eval _time=_indextime
| timechart span=1d count by host
```

✓ **19,128,188,386 events** (6/13/21 12:00:00.000 AM to 7/13/21 8:53:07.000 AM) No Event Sampling

Search job inspector | Splunk 8.1.2101.2

Search job inspector

This search has completed and has returned **31** results by scanning **19,128,188,386** events in **79.928** seconds

(SID: 1626187987.1064430_677D0E40-1944-41AC-AD52-F759224C8AF4) search.log Job Details Dashboard

splunk> .conf21

Conversion Algorithm

From search to tstats

1) What is the end result?

2) Are the fields needed available at index time?
   - | walklex index="<index>" type=field
     | search NOT field=" *"
     | stats list(distinct_values) by field

3) Rearrange the search so that the aggregation commands are in the tstats command

4) Use prestats=true when wanting to run a timechart command

5) Add index, source, sourcetype, etc. filters into the WHERE clause

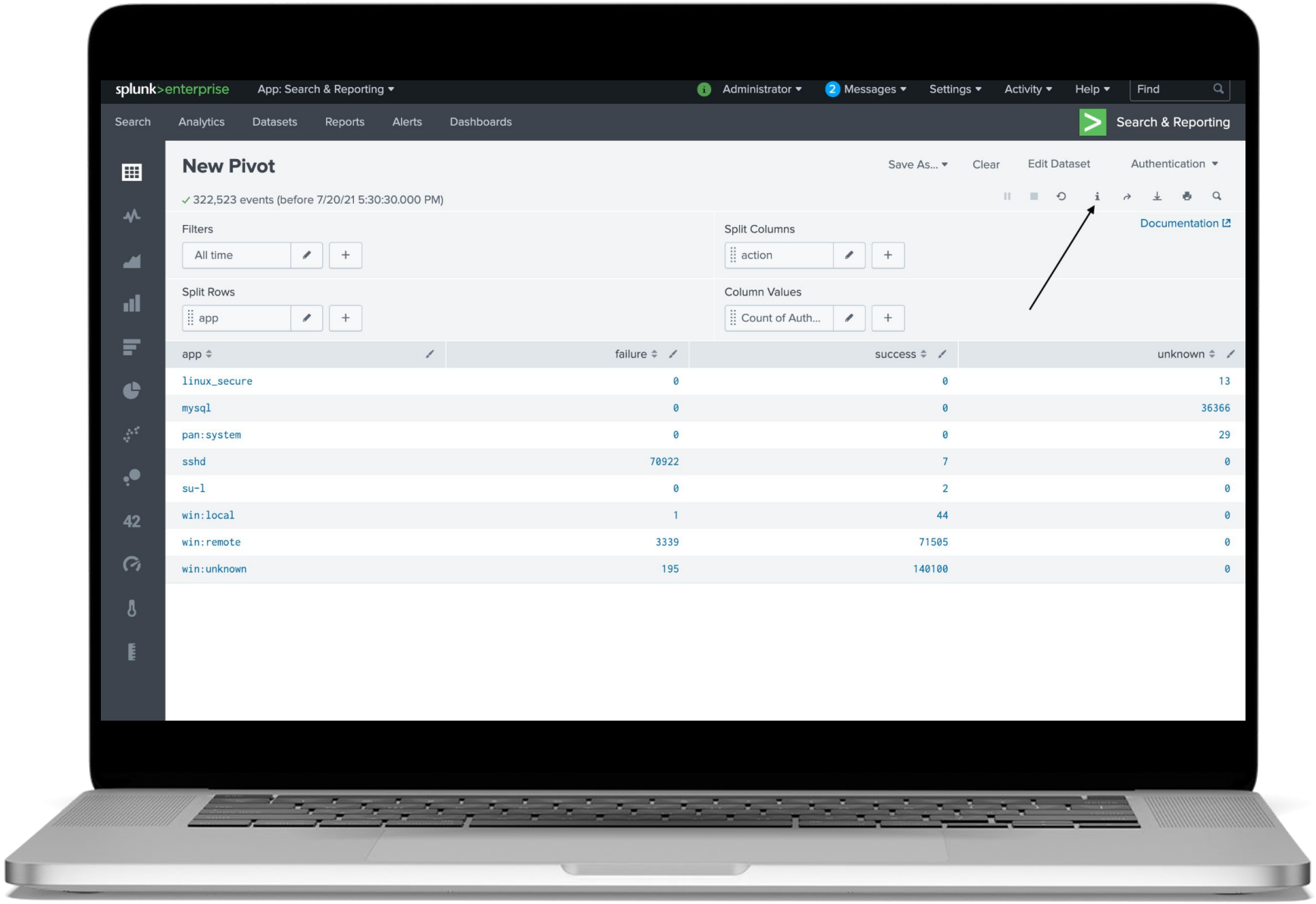
Let’s Pivot

splunk> .conf21

Data Models

Job Inspector to the Rescue

Find the | tstats search from a working Pivot using the Job Inspector

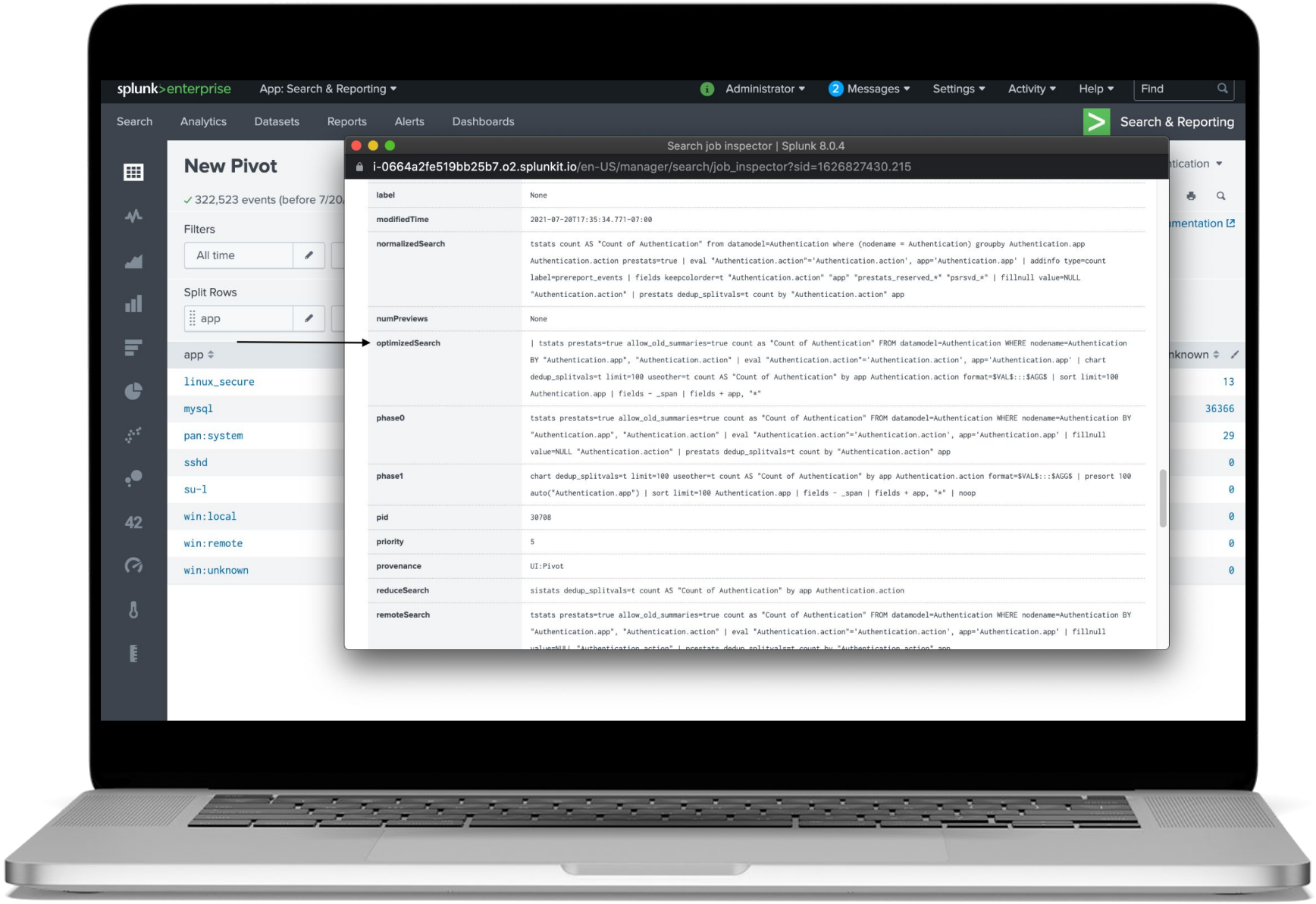


splunk> .conf21

Data Models

Job Inspector to the Rescue

Note that this only works when the DM is accelerated - pivot on an unaccelerated DM is converted to regular search



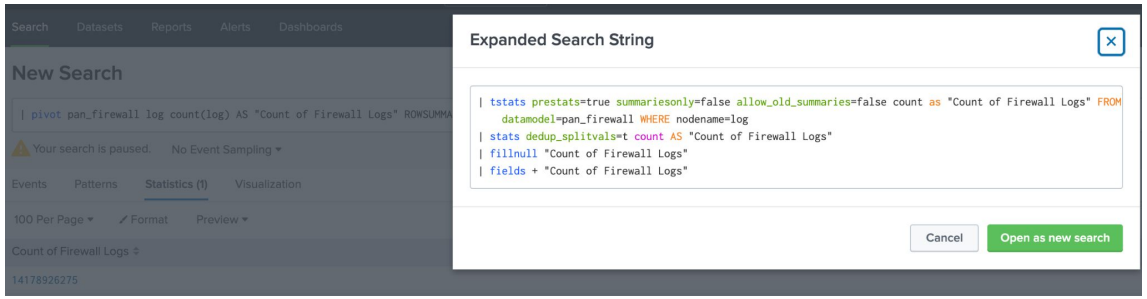
splunk> .conf21

Expanding Pivot

Run `pivot` in search

Mac: `cmd + shift + E`
Windows: `ctrl + shift + E`

`Pivot` will expand into `tstats` or generic `search`, depending on DM acceleration




```
| pivot pan_firewall log count(log) AS "Count of Firewall Logs" ROWSUMMARY 0 COLSUMMARY 0 SHOWOTHER 1
```

BECOMES

```
| tstats prestats=true summariesonly=false allow_old_summaries=false count as "Count of Firewall Logs" FROM datamodel=pan_firewall WHERE nodename=log
| stats dedup_splitvals=t count AS "Count of Firewall Logs"
| fillnull "Count of Firewall Logs"
| fields + "Count of Firewall Logs"
```

Explore Your Options

splunk> .conf21

Datamodel-specific Options

`summariesonly=true`

- Use accelerated data only, do not fill in the gaps for the last few minutes
- Great for dashboards displaying 24 hours of data

```
| tstats summariesonly=true estdc(All_Traffic.src) from datamodel=Network_Traffic by All_Traffic.dvc
```

`allow_old_summaries=true`

- Allows tstats to use existing older accelerated data even if the DM changed, e.g. an eval-field was refined
- Favours speed over using the latest search-time knowledge objects
- Most relevant in combination with `summariesonly=true`, avoids entirely empty results after a change

```
| tstats summariesonly=true allow_old_summaries=true count from datamodel=Authentication by _time span=1h
```

Datamodel-specific Options

`summariesonly=true`

- Use accelerated data only, do not fill in the gaps for the last few minutes
- Great for dashboards displaying 24 hours of data

```
| tstats summariesonly=true estdc(All_Traffic.src) from datamodel=Network_Traffic by All_Traffic.dvc
```

`allow_old_summaries=true`

- Allows tstats to use existing older accelerated data even if the DM changed, e.g. an eval-field was refined
- Favours speed over using the latest search-time knowledge objects
- Most relevant in combination with `summariesonly=true`, avoids entirely empty results after a change

```
| tstats summariesonly=true allow_old_summaries=true count from datamodel=Authentication by _time span=1h
```

Good Options for All

`fillnull_value="Buttercup"`

- Tells `tstats` what do do when events don't have values for a group-by-field
- Default behaviour: Ignore those events!

```
| tstats fillnull_value="Buttercup" count where index=_internal by user
```

`span=1h`

- Used when grouping by `_time`, almost like in the `timechart` and `bin` commands
- Aggregates slices of time together by the given span instead of automatically based on time range

```
| tstats count from datamodel=Authentication by _time span=1h
```

- Note: Unlike `timechart`, `tstats` will not generate rows for empty spans

Good Options for All

`fillnull_value="Buttercup"`

- Tells `tstats` what do do when events don't have values for a group-by-field
- Default behaviour: Ignore those events!

```
| tstats fillnull_value="Buttercup" count where index=_internal by user
```

`span=1h`

- Used when grouping by `_time`, almost like in the `timechart` and `bin` commands
- Aggregates slices of time together by the given span instead of automatically based on time range

```
| tstats count from datamodel=Authentication by _time span=1h
```

- Note: Unlike `timechart`, `tstats` will not generate rows for empty spans

Map-Reduce Shenanigans

`prestats=true`

- Allows you to specify the Map- and Reduce-parts of `tstats` independently from each other
- Enables lots of cool hacks coming up in a minute

```
| tstats prestats=true count from datamodel=Web by Web.src | 🧙 | stats count by Web.src
```

`append=true`

- Lets you concatenate multiple `tstats` result sets on the indexers, `prestats=true` only
- Allows 7.x-compatible emulation of `fillnull_value` without `append` command:

```
| tstats prestats=true count where index=_internal user=* by user
| tstats prestats=true append=true count where index=_internal NOT user=*
| fillnull value="Buttercup" user
| stats count by user
```

Map-Reduce Shenanigans

`prestats=true`

- Allows you to specify the Map- and Reduce-parts of `tstats` independently from each other
- Enables lots of cool hacks coming up in a minute

`| tstats prestats=true count from datamodel=Web by Web.src |` 🧙 `| stats count by Web.src`

`append=true`

- Lets you concatenate multiple `tstats` result sets on the indexers, `prestats=true` only
- Allows 7.x-compatible emulation of `fillnull_value` without `append` command:

```
| tstats prestats=true count where index=_internal user=* by user
| tstats prestats=true append=true count where index=_internal NOT user=*
| fillnull value="Buttercup" user
| stats count by user
```

`prestats=true` Basics

- Every search has a Map- and a Reduce-phase
- Job inspector: `phase0` & `phase1`
- Event search `phase0` is “everything including the first stats”, `phase1` is “everything from the first stats”
- `tstats` is its own generating command and stats combined, therefore `tstats` is split up automatically:

```
search: | tstats count where index=_internal by sourcetype

phase0: | tstats prestats=t <snip> count where index=_internal groupby sourcetype

phase1: | tstats <snip> count WHERE index=_internal BY sourcetype
```

- That’s equivalent to specifying both phases manually:

```
| tstats prestats=t count where index=_internal by sourcetype
| stats count by sourcetype
```

`prestats=true` Basics

- Every search has a Map- and a Reduce-phase
- Job inspector: `phase0` & `phase1`
- Event search `phase0` is “everything including the first stats”, `phase1` is “everything from the first stats”
- `tstats` is its own generating command and stats combined, therefore `tstats` is split up automatically:

```
search: | tstats count where index=_internal by sourcetype

phase0: | tstats prestats=t <snip> count where index=_internal groupby sourcetype

phase1: | tstats <snip> count WHERE index=_internal BY sourcetype
```

- That’s equivalent to specifying both phases manually:

```
| tstats prestats=t count where index=_internal by sourcetype
| stats count by sourcetype
```

`prestats=true` Under the Hood

- What if you “forget” the manual `phase1` / Reduce?

```
| tstats prestats=t count where index=_internal by sourcetype
```

✓ **16 events** (04/07/2021 16:00:00.000 to 05/07/2021 16:20:16.000) No Event Sampling

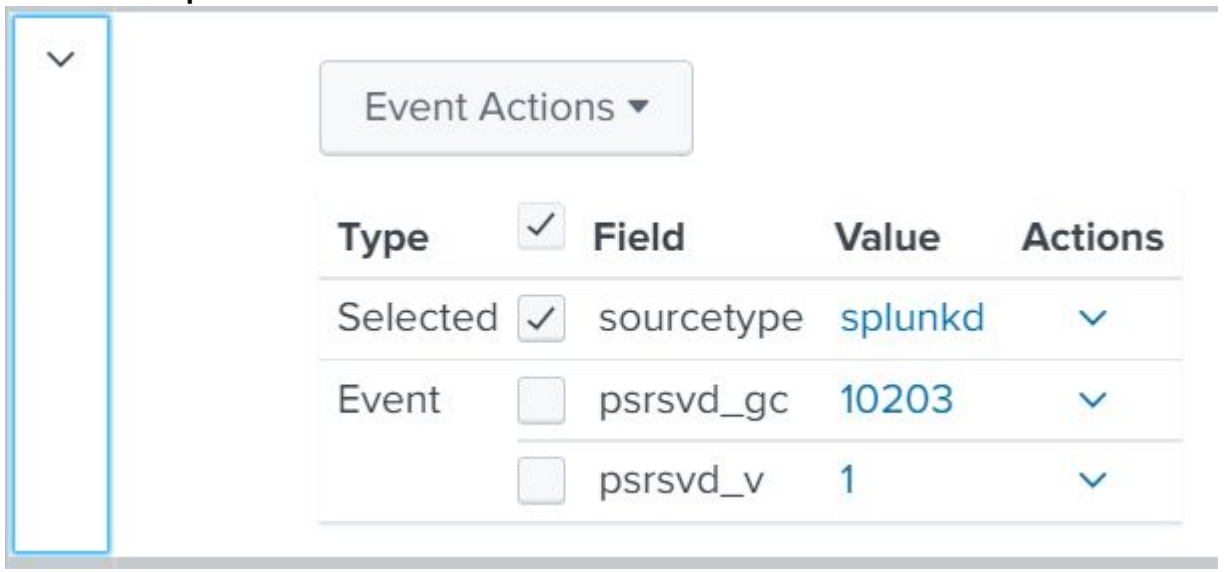
Events Patterns Statistics Visualization

List Format 20 Per Page

i	Time	Event
>		sourcetype = splunk_web_access
>		sourcetype = splunk_web_service
>		sourcetype = splunkd

`prestats=true` Under the Hood

- What if you “forget” the manual `phase1` / Reduce?



- `psrsvd_gc` = “prestats-reserved global count”, a partial contribution to the total count for `splunkd`
- You can do SPL with those fields! On the Indexers!
- Touching group-by fields (`sourcetype`) is usually safe, don’t touch aggregations (`count` / `psrsvd_gc`)

`prestats=true` Example: Lookups

- Let the Searchhead do the lookup work all on its own

```
| tstats sum(All_Traffic.bytes) as bytes from datamodel=Network_Traffic by All_Traffic.src_ip
| lookup dnslookup clientip as "All_Traffic.src_ip" OUTPUT clienthost
| search clienthost="*.splunk.com"
| stats sum(bytes) as splunk_bytes
```

Indexers
Searchhead

- Let the Indexers share the lookup work between themselves in parallel

```
| tstats prestats=t sum(All_Traffic.bytes) from datamodel=Network_Traffic by All_Traffic.src_ip
| lookup dnslookup clientip as "All_Traffic.src_ip" OUTPUT clienthost
| search clienthost="*.splunk.com"
| stats sum(All_Traffic.bytes) as splunk_bytes
```

Indexers
Searchhead

- Note: Renames with `prestats=t` are ignored, the reduce-`stats` has to refer to the original name

`prestats=true` Example: Naïve Approach

Enterprise Security Content Update: SQL Injection with Long URLs

- Load all URLs from the Web datamodel to the searchhead, apply complex filtering

```
| tstats summariesonly=t count from datamodel=Web by Web.url
| eval num_sql_cmds = <complex eval to count SQL-y looking things>
| where num_sql_cmds > 3
```

This search has completed and has returned **22** results by scanning **536,216,726** events in **38.779** seconds

	51.42	dispatch.stream.remote	1,182	-	1,081,017,538

- Over 1GB of data returned to SH, 2.5M rows need to be merged and sorted before `eval`
- Single SH doing filtering

prestats=true Example: Smart Approach

Enterprise Security Content Update: SQL Injection with Long URLs

- Offload the complex filtering to the indexers, only load matching URLs to the searchhead

```
| tstats summariesonly=t prestats=t count from datamodel=Web by Web.url
| eval num_sql_cmds = <complex eval to count SQL-y looking things>
| where num_sql_cmds > 3
| stats count by Web.url
```

This search has completed and has returned **22** results by scanning **536,216,726** events in **6.881** seconds

	60.88	dispatch.stream.remote	1,182	-	9,072,147

- Under 10MB of data returned to SH
- Parallel filtering on every indexer
- 6x speedup

Key Takeaways

splunk> .conf21

What to Remember

- `tstats` works on indexed fields and datamodels
- `ctrl+shift+E` or `cmd+shift+E` will expand the `pivot` search and show `tstats`
- `prestats=t` is great when additional aggregations follow
- `FROM` specifies the datamodel
- `WHERE` filters events
- `PREFIX()` and `TERM()` are magic
- Be mindful of high cardinality intermediate results

Resources

splunk> .conf21

Helpful Links

- More on the Job Inspector: https://conf.splunk.com/watch/conf-online.html?search=TRU1143C
- `tstats` Reference Docs: https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Tstats
- `pivot` Reference Docs: https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Pivot
- Additional Background on Map-Reduce: https://en.wikipedia.org/wiki/MapReduce

Thank You

Please provide feedback via the

SESSION SURVEY

splunk> .conf21