Metric Indexes

Architecture and Usage

Allan Yan - Principal Engineer, Search Technologies

Steve Zhang – Chief Scientist

October 2018

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements we may make. 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. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Agenda

- Why are metrics special?
- Metric data model
- Using metrics
  - Ingest
  - Manage
  - Query
- Counters
- Performance

Why Metric Indexes?

Metrics Data Fundamentally Different From Log Data

- High Volume
- Low Latency
- Structured
- Data Emitted At Consistent Interval

- Constrained Query Interface
- Higher Tolerance for Loss/Approximations

The Splunk Metric Store presents a constrained query interface and leverages the structured nature of metrics data to meet higher volume and lower latency demands

Metrics As Logs

```
09-27-2018 10:55:32.618 INFO Metrics - group=pipeline, name=parsing, processor=utf8,
cpu_seconds=0.012845, executes=66, cumulative_hits=301958
```

- Metrics often found in log form
  - Splunk event indexes could handle these naturally (e.g. metrics.log in index=_internal)
  - Could even render metrics as logs to store in Splunk
- Weakness in Approach
  - Analytics over textual events relatively slow (~50K events/node/sec)
  - Leveraging indexed fields & tstats is significantly better, but not optimal
    - Storing metric **values** in keyword lexicon is inefficient and unnecessary
    - Values for a single metric series should be co-located whenever possible

Metrics Data Model

Metric Data Point

- The atomic event representing a **single** measurement in time

Dimensions (key=value<string>)

- Key/Value pairs associated with a particular metric data point
- Aside from metric_name, these are **Optional**
- Examples: app=Solitaire, host=linux-1, datacenter=west

Metric Name (metric_name=<string>)

- **Required** dimension for all metric data points.
- Examples: cpu.idle, io.util, temp, page_hits

Measurement (_value=<number>)

- **Required** numerical field for all metric data points, stored as 64 bit float

Timestamp (_time=<number>)

- **Required** time field for all metric data points, stored as 32 bit integer

Metric Time Series

- A series of metric data points over time with the same metric_name and exact same dimension key/value pairs

Metric Time Series Examples

Server A

Server B

Dimensions				
metric_name	host	app	_time	_value
cpu.idle	A	foo	1	4.2
mem,free	B	bar	2	7.3
cpu.idle	A	foo	3	8.4
mem.free	A	baz	4	32

Different Colors Represent Distinct **Metric Time Series**. Each Row is a Single **Metric Data Point**, made up of a timestamp, measurement and a set of required and optional dimensions

Both of these data points belong to the same **Metric Time Series** because they share the exact same set of required and optional dimension key/value pairs:

metric_name=cpu.idle
host=A
app=foo

Temp Sensor A

Temp Sensor B

Dimensions				
metric_name	host	room	_time	_value
temperature	A	Tupac	1	4.2
temperature	B	Biggy	2	7.3
temperature	A	Joplin	3	8.4
temperature	B	Tupac	4	32

Note that different Metric Time Series may have a completely separate set of **Optional Dimensions**. For example, the time series from temperature sensors have a “room” dimension. The ones from servers above have a “app” dimension.

Metrics Input/Parsing

Numerous Ways of getting Metrics Into System

- Http Event Collector
  - Send structured data over HTTP
- Modular/Scripted Input
  - Send structured data over Modular Input Subsystem
- Statsd over tcp/udp
  - Native support for parsing statsd events
- Collectd
  - Native support for parsing collectd events
- Csv/Json
  - File based structured data (csv/json) with required metric fields
- Logs to Metrics
  - Ingest time extractions via props/transforms.conf to convert log events to metric events
  - New for 7.2: event splitting, UI support
  - **Go to Metrics Ingestion session for details. (Thursday 1:30 PM)**

Logs to Metrics at Search Time

mcollect & meventcollect

- Re-ingest already indexed log events into metrics index (Splunk 7.1+)
  - Similar to summary indexing using `collect`
  - No additional license cost
- Best practice: using `meventcollect` for simple events search, use `mcollect` turning report output to metrics (e.g. `stats, timechart`)

mcollect	meventcollect
Use in any search	Only after distributable commands (e.g. `search`, `where`, `eval`)
Runs on Search Head	Runs on indexers
Store on SH or forward to indexers	Store on indexers

Logs to Metrics at Search Time (Cont.)

```
mcollect/meventcollect index=<string> [split=<bool>] [prefix_field=<string>]
[<field-list>]
```

- Single metric per event/row (split=false)
  - Requires metric_name and _value fields. Other fields treated as dimensions (or explicitly specified in <field-list>)

  -
    ```
    sourcetype=foo | stats count as _value by user city | eval
    metric_name=“foo.count” | mcollect index=my_metrics split=false user city
    ```
- Multiple metrics per event/row (split=true)
  - Must specify list of dimension fields. All other numerical fields treated as **metric values** (a metric data point is emitted for each one)

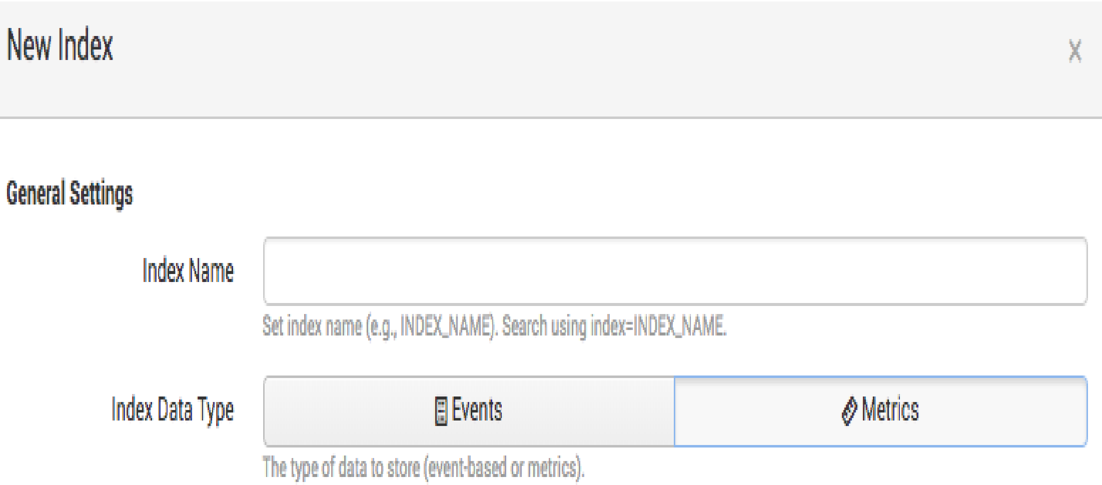
  -
    ```
    index=_internal source=*/metrics.log | eval prefix = group + “.” |
    meventcollect index=my_metrics split=true prefix_field=prefix name processor
    ```

```
09-27-2018 10:55:32.618 INFO Metrics - group=pipeline, name=parsing,
processor=utf8, cpu_seconds=0.012845, executes=66, cumulative_hits=301958
```

Metrics Data Management

Metrics Indexes

- Metric indexes very similar to event indexes
  - Data written to hot buckets. Buckets rolled to warm, cold, frozen, etc.
  - Internal storage format similar to event indexes (.tsidx)
- Most data management features for log indexes are supported for metrics
  - Replication, retention policy, access controls, remote storage, etc.
  - Exception: metrics does not support deleting individual data points.



New Index

General Settings

Index Name:

Set index name (e.g., INDEX_NAME). Search using index=INDEX_NAME.

Index Data Type: Events | Metrics

The type of data to store (event-based or metrics).

indexes.conf

```
[metrics_index]

# this is a metrics index
datatype = metric

# paths/other index params …
```

Querying Metrics

The mstats command

- mstats presents a Constrained Query Language similar to tstats
  - Projections, Group-By and Filter clauses
  - Cannot filter or group by _value field, only dimensions
- Original mstats syntax (7.0+)
  - Aggregations must be over _value field (sum, count, avg, dc, median, percX)
  - `metric_name` must be specified in the `WHERE` expression (wildcards ok)

```
|mstats <projections over _value>
         WHERE metric_name=<string> [<dimension filter predicates>|<index specification>]
         BY [<group-by dimensions>] [<span specification>]
Ex: |mstats avg(_value)
         WHERE index=metrics metric_name=cpu.idle host=splunk-*.com
         BY datacenter span=5m
Ex: |mstats sum(_value)
         WHERE index=metrics metric_name=cpu.idle host=splunk-*.com span=1h
```

Querying Metrics

The mstats command – enhanced syntax

- Specifying multiple metrics awkward in original syntax

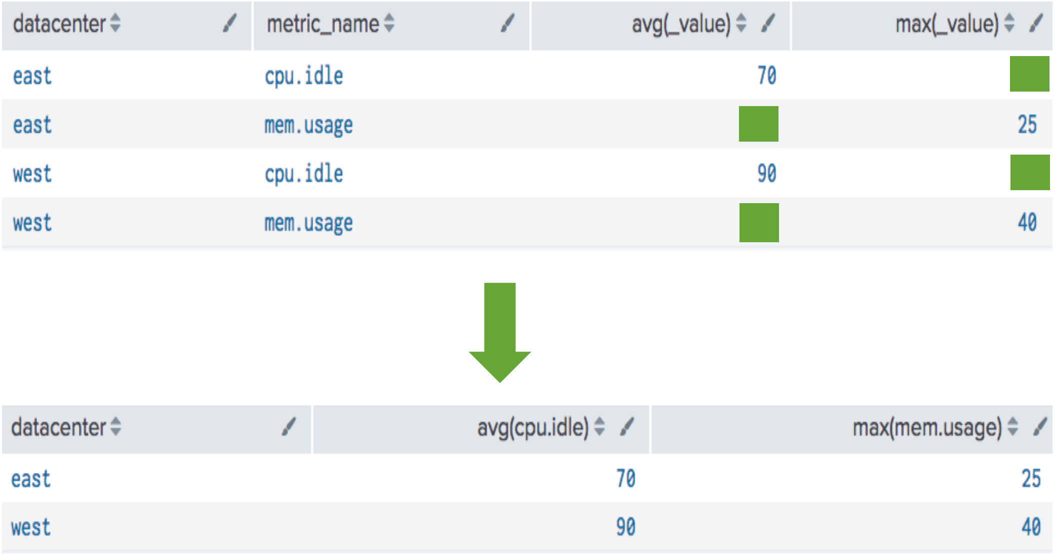
```
|mstats avg(_value) max(_value) WHERE index=metrics metric_name=cpu.idle OR
metric_name=mem.usage BY datacenter metric_name
```

  - Format of output is also awkward and likely needs further `eval/where/stats` commands to format properly.

- Enhanced mstats syntax (7.1+)
  - Allow for easier specification of multiple metrics.

```
|mstats <projections over metric_name>
        WHERE [<dimension filter predicates>|<index specification>]
        BY [<group-by dimensions>] [<span specification>]
Ex: |mstats avg(cpu.idle) max(mem.usage)
        WHERE index=metrics
        BY datacenter
```

  - Use `metric_name` instead of `_value`.
  - No `metric_name` allowed in the WHERE expression



datacenter	metric_name	avg(_value)	max(_value)
east	cpu.idle	70	🟩
east	mem.usage	🟩	25
west	cpu.idle	90	🟩
west	mem.usage	🟩	40

datacenter	avg(cpu.idle)	max(mem.usage)
east	70	25
west	90	40

- Original syntax better when you need to treat different metric_names as a single metric
  - `| mstats avg(_value) WHERE metric_name=“cpu.util” OR metric_name=“cpu.utilization” …`

Querying Metrics Catalog

The mcatalog command

- Constrained query language similar to tstats
  - Can ONLY list metrics catalog information, e.g. metric names, dimensions
  - convenient internal field: _dims
  - Cannot project, filter or group by _value field, only dimensions

```
|mcatalog values(metric_name|_dims|_catalog|<dimension>)
        WHERE [metric_name=<string>] [<dimension filter predicates>|<index specification>]
        BY [<group-by dimensions>]
Ex: |mcatalog values(_dims) WHERE index=metrics BY metric_name
Ex: |mcatalog values(metric_name) WHERE index=metrics
Ex: |mcatalog values(region) WHERE index=metrics BY metric_name
```

Querying Metrics Catalog

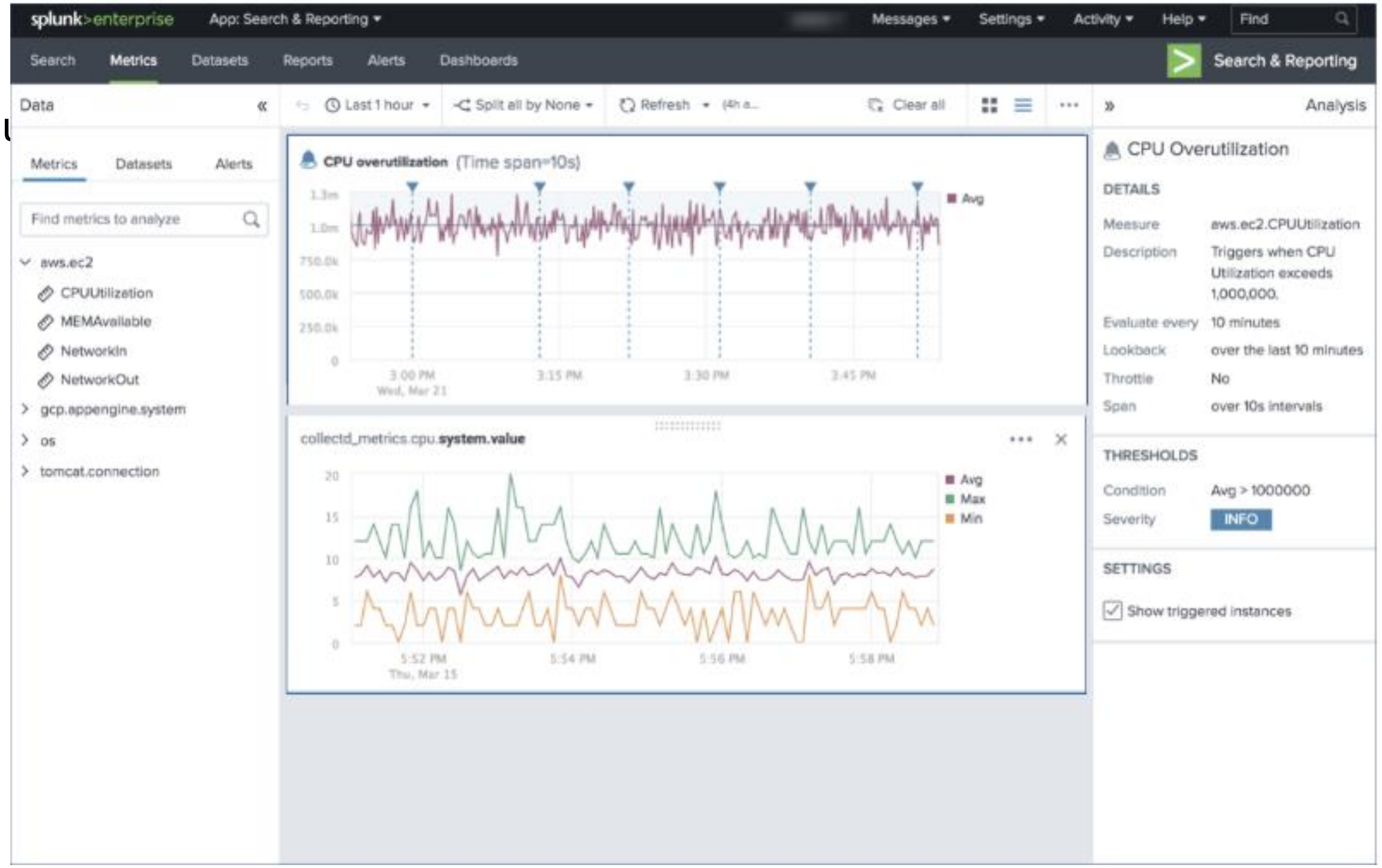
The metrics catalog endpoint

- Metrics catalog endpoints
  - List metric names: /services/catalog/metricstore/metrics
  - List dimension names: /services/catalog/metricstore/dimensions
  - List dimension values: /services/catalog/metricstore/dimensions/{dimension-name}/values
  - Filter results by index, dimension and dimension values (including wildcard):
    - /services/catalog/metricstore/metrics?filter=index=metrics&filter=dc
    - /services/catalog/metricstore/metrics?filter=index=metrics&filter=dc=east
    - /services/catalog/metricstore/dimensions?filter=index=metrics
    - /services/catalog/metricstore/dimensions?filter=index=metrics&filter=dc*

Querying Metrics and Catalog

The Analysis Workspace

- Try the Analysis Workspace!
  - Full featured GUI. No SPL knowledge requ
  - Allows for
    - drilldown
    - open in search
    - create dashboard
    - Others: alters, export etc.
  - https://splunkbase.splunk.com/app/3976/



CPU Overutilization

DETAILS

Measure aws.ec2.CPUUtilization

Description Triggers when CPU Utilization exceeds 1,000,000.

Evaluate every 10 minutes

Lookback over the last 10 minutes

Throttle No

Span over 10s intervals

THRESHOLDS

Condition Avg > 1000000

Severity INFO

SETTINGS

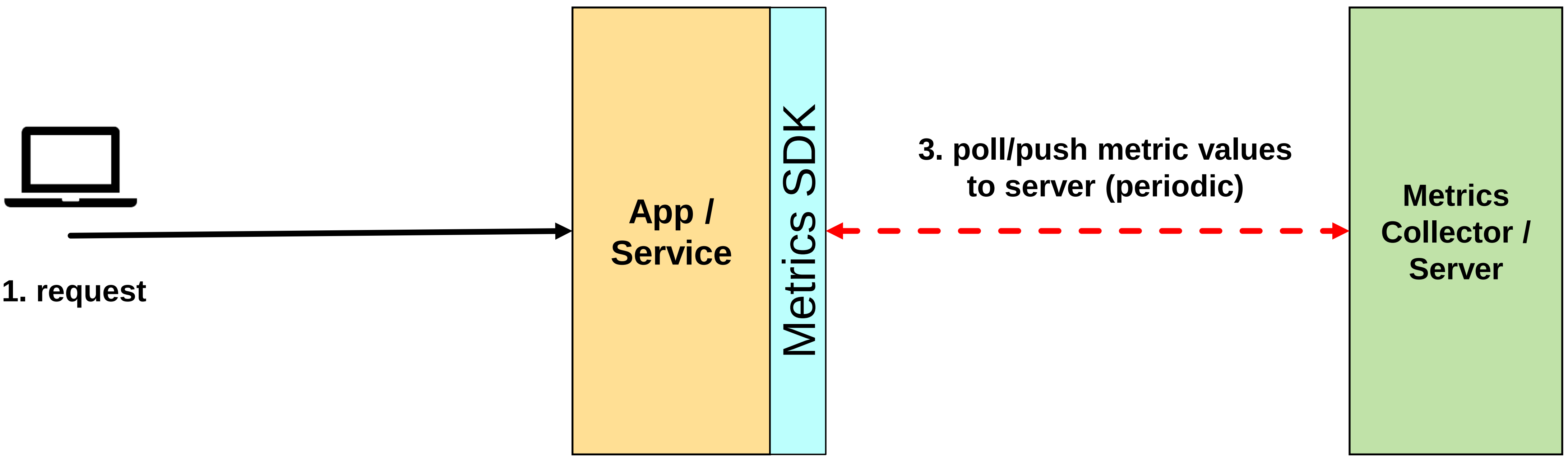
☑ Show triggered instances

Compare and Contrast stats Family Commands

	Historical Search	Realtime Search	Metric Index	Event Index	Aggregate on Dimensions / Index-time Fields	Aggregate on Search-time Fields	Aggregate on metric values (_value)
mstats	X	X	X				X
mcatalog	X		X		X		
tstats	X			X	X		
search + stats	X	X		X	X	X	

- mstats aggregate on metric values, mcatalog search for metadata
- real-time mstats much more efficient than real-time event search + stats
- tstats on index time fields only. search+stats on index time fields still slow due to reading raw data.

Counters



1. request

App / Service

Metrics SDK

3. poll/push metric values to server (periodic)

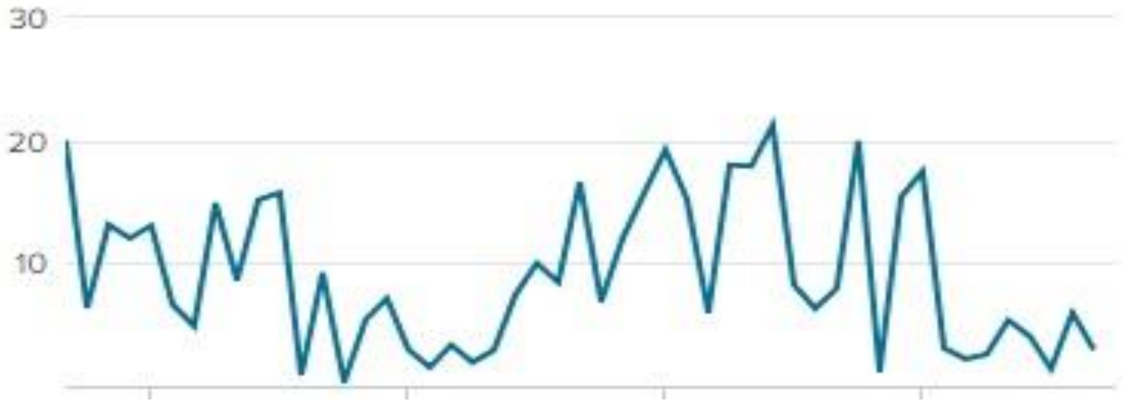
Metrics Collector / Server

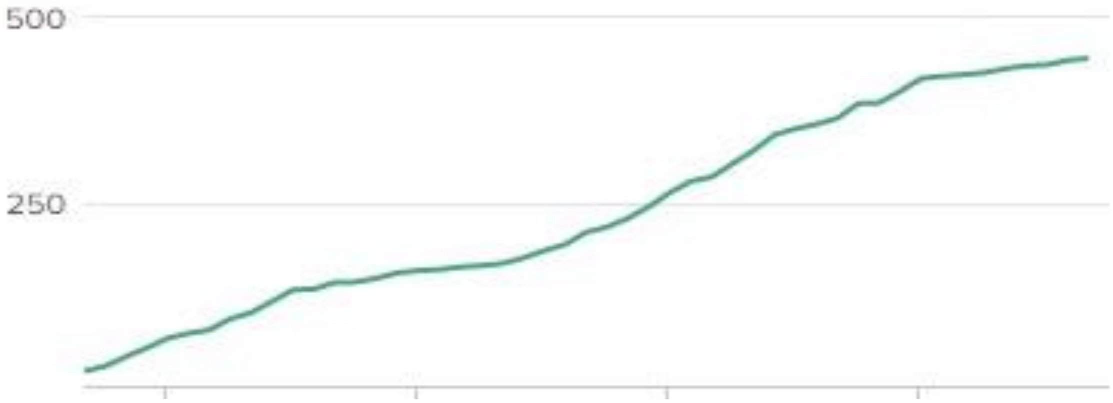
2. Invoke client api, e.g.
`increment(<counter_name>)`

Counters

To reset or not to reset?

Periodic Counters	Accumulating Counters
Counter value resets to 0 every time it is reported to server	Counter value resets only when service is reset
Counter value represents increments since last report	Needs at least 2 measurements to compare because only deltas are meaningful
`statsd, collectd` **`ABSOLUTE`**`, collectd` **`DERIVE`** `(storerates=true)`	`Prometheus, collectd` **`COUNTER`**`, collectd` **`DERIVE`** `(storerates=false)`





Consistency is key! (between client/server/query)

Counters

How to query

- Gauge values (e.g. current temperature)
  - `min()`, `max()`, `avg()`, `perc()`
- Periodic counters
  - `sum()`
- Accumulating counters
  - `rate()` new for Splunk 7.2: similar to `rate()` in Prometheus, `derivative()` in Influxdb
  - Use `latest()` and `streamstats` before 7.2 or to customize treatment of resets, rollovers, or missing data

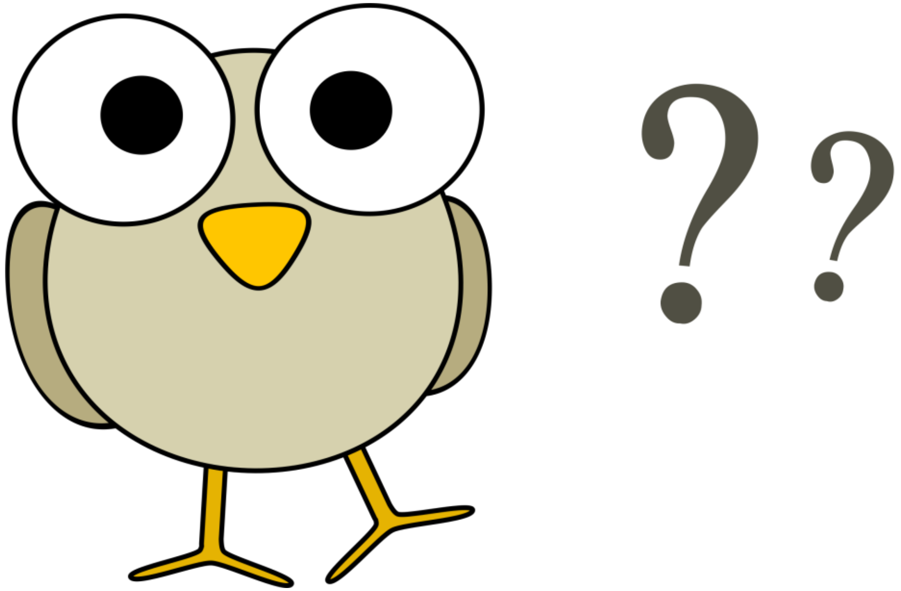
Performance Characteristics

- Search Performance **significantly better** in 7.1+ (**~5-6x** for typical searches)
  - Better co-location of data points for a metric series.
- Indexing performance fairly **consistent** across workload characteristics
  - 100K EPS/node for average hardware (2x12 Xeon 2.3Ghz, 64GB RAM, 15K disks)
- Search performance highly **sensitive** to cardinality of metric series
  - Each metric series **shares** the metric_name and exact set of dimension values
  - **Magic ratio** = # metric data points / # metric series (for each bucket)
  - Performance tends to **quickly degrade** when ratio is < 100 due to per metric series overhead.
- Tuning Parameters
  - Bucket size (larger is better for search)
  - Hashing on metric_name (touch fewer buckets during searches over specific metrics. Tricky tradeoffs; not usually recommended.)

Future Work

- Improve (accumulating) counter computation
  - rate() to account for rollover and resets
- Metric Rollups
  - Rollup metrics to longer time intervals (e.g. 1 hour, 1 day) for longer retention
  - Similar to summary indexing
- Add support for linked metrics
  - Optimize storage and querying for multiple metric values with the same set of dimension values

Questions?



Thank You

Don't forget to rate this session in the .conf18 mobile app

.conf18

splunk>