Containing your Cardinality

Chris Marchbanks, @csmarchbanks

PromCon EU, 2019

splunk> turn data into doing™

About me

Engineer @ Splunk

Tech lead for internal observability platform for Splunk Cloud

Newest team member of Prometheus

Passionate about all things observability

Uphill skiing on the weekends (it's almost time again!)

Who is concerned about cardinality in their system?

Have you seen this warning?

CAUTION: Remember that every unique combination of key-value label pairs represents a new time series, which can dramatically increase the amount of data stored. Do not use labels to store dimensions with high cardinality (many different label values), such as user IDs, email addresses, or other unbounded sets of values.

What is Cardinality

The number of series held in Prometheus over a given timeframe.

Sources of cardinality:

1. Number of targets
2. Number of series a target exposes
3. Targets or the series changing over time (churn)

Cardinality is valuable, but excessive cardinality is expensive!

How does Cardinality hurt?

More resource usage

Slow queries

Eventually queries will be rejected or OOM your Prometheus

Long startup times as Prometheus loads series

Shared fate of HA Prometheus pairs

How much is too much?

For a single metric, queries begin to be painfully slow above ~100,000 series.
E.g. an instant query like `count(metric_a)`

- 100,000 series in ~1.5 seconds
- 200,000 series in ~5 seconds

A single Prometheus can handle >10,000,000 series in memory

- Startup can take 15+ minutes
- Heap can exceed 100 GB
- If a scale factor increases, can quickly topple due to multiplicative nature

Multiplicative Cardinality

You have a histogram, *`http_request_duration_seconds_bucket`*

100 instances * 10 buckets -> 1000 series
* 10 endpoints -> 10,000 series
* 10 response codes -> 100,000 series (maximum recommended cardinality)
* 4 http methods -> 400,000 series
* 100 tenants -> 40,000,000 series ☠☠☠

Types of data I see in Prometheus

Operational

a. Determine if your service is meeting its SLOs
b. Direct an operator to a problem with minimal exploration
c. Provide evidence that your service is operating correctly
d. Must be fast and reliable!

Telemetry

a. Allow investigation into how users are interacting with the application
b. Can be sliced and diced in many different ways to answer new questions
c. Less latency sensitive

Prometheus is designed for operational data, telemetry data is best sent to your favorite event system

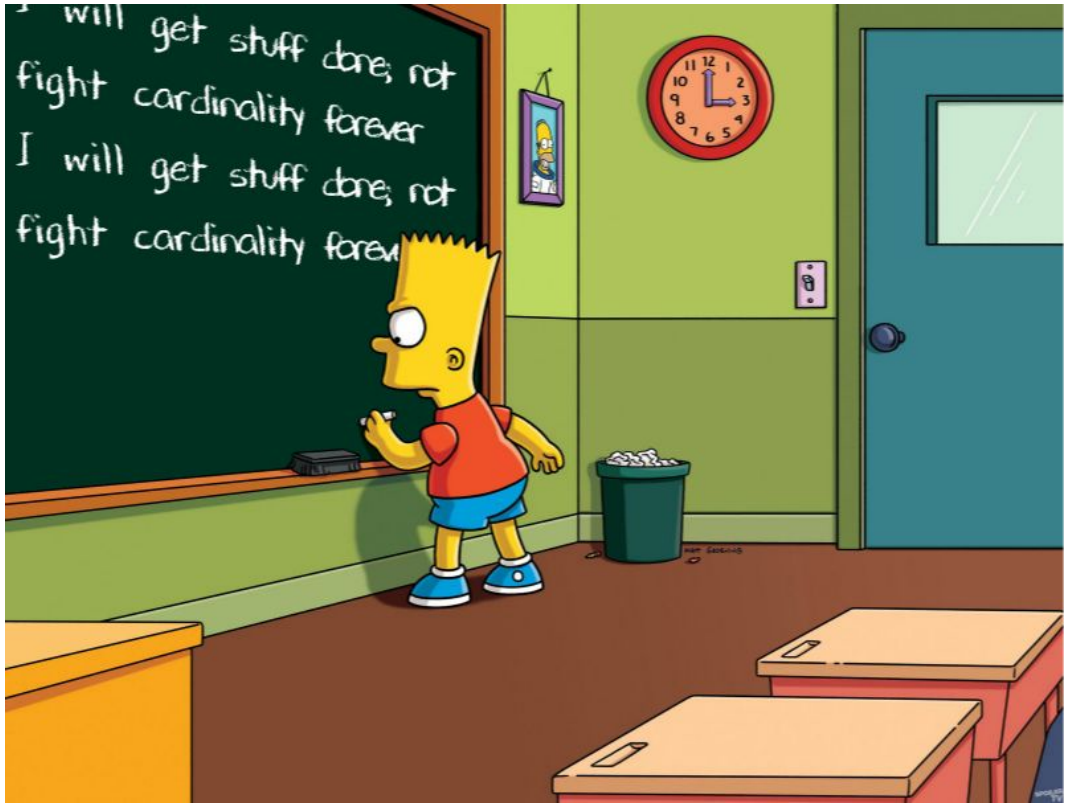
Tips to Contain your Cardinality

splunk> turn data into doing™

Tip #0: Be pragmatic

These are all suggestions, something else might make sense!

A few high cardinality series are ok.



#thoughtfullyconsider

Tip #1: How is a label used?

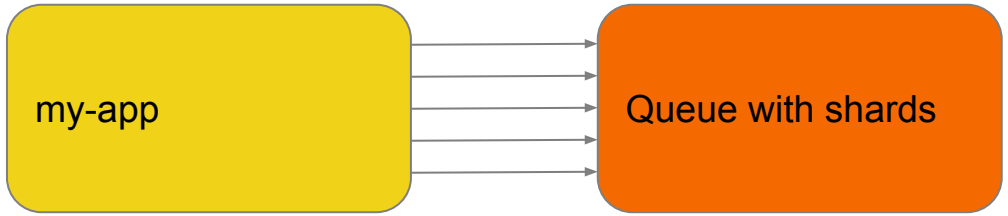
Does the label help someone woken up at 2am identify the problem?

Is there a predictable number of values for the label?

If the answer to the above questions is no, consider having the label on an event log instead of in Prometheus.

Tip #2: Model your system

Service that writes data onto a queue



Sharded by
tenant/user id

If you are woken up, what would you want to see?

1. Per-tenant metrics showing that some customers are having issues?
2. A per shard metric that shows you a single shard is overloaded?

Tip #3: Use multiple metrics

Two ways to model availability and latency

All the labels on one metric

http_request_duration_seconds_bucket

100 instances *
10 buckets *
10 response codes *
10 routes = **100,000 series**

Split the metric

http_request_duration_seconds_bucket

100 instances * 10 **buckets** * 10 routes = 10,000 series

http_requests_total

100 instances * 10 **response codes** * 10 routes = 10,000 series

Total: **20,000 series**

Tip #4: Isolate high cardinality

Only do this for high value sources.

E.g. Keep “tenant” labels only on high level API calls, not every metric.

Consider functional sharding and federation to keep high cardinality sources from bringing down all of your monitoring.

Tip #5: Some tools!

Prometheus v2.14.0 releases cardinality stats in the UI!

Useful queries:

1. *sum(scrape_series_added) by (job)*
2. *sum(scrape_samples_scraped) by (job)*
3. *prometheus_tsdb_symbol_table_size_bytes*

Highest Cardinality Metric Names

Name	Count
codelab_api_request_duration_seconds_bucket	8736000
codelab_api_request_duration_seconds_sum	336000
codelab_api_requests_total	336000
codelab_api_request_duration_seconds_count	336000
codelab_api_request_errors_total	168000
go_gc_duration_seconds	120030
go_memstats_mcache_sys_bytes	24006
go_memstats_mspan_inuse_bytes	24006
go_memstats_next_gc_bytes	24006
go_memstats_heap_released_bytes	24006

Thanks!

Questions?

Chris Marchbanks, @csmarchbanks

splunk> turn data into doing™