Finding the Golden Signals with Prometheus

Implementing SLOs, Histograms, and Burn Rates

Jack Neely
jjneely@42lines.net

October 20, 2020

42 Lines, Inc.

What Is Prometheus?

prometheus.io

33K Stars, 1.2K Watchers, 5.2K Forks

Cloud Native Time Series Database

Use Prometheus Effectively

Solve these three simple problems before your business scales:

- Methods of Common Instrumentation
- Service Level Objectives and Arbitrary Percentiles
- Effective Alerting and Burn Rates

Methods of Common Insturmentation

Goal of Site Wide Instrumentation

Work with your developers to integrate metrics into the base kit used to build a new project. Create and encourage a metric naming scheme!

Use the Prometheus Naming Best Practices as a guide. Use standard units of measurement.

Example PromQL: View a Graph of HTTP/API Hits by Service

```
> sum by (job) (
    rate(http_request_duration_seconds_count[1m])
  )
```

Use a Method!

The USE Method

- Brendan Gregg, 2012
- Hardware or Resource Focused
- Utilization, Saturation, Errors

The RED Method

- Tom Wilkie, 2015
- Services and APIs
- Rate, Errors, Duration

The 4 Golden Signals

- Google's Site Reliability Engineering Book, 2016
- RED + S
- Latency, Traffic, Errors, Saturation

Methods Create Service Level Indicators

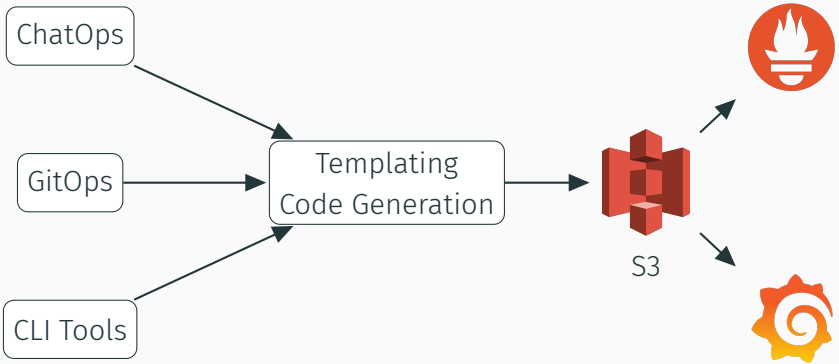
These metrics become a standard set of Service Level Indicators (SLIs). Actionable Service Level Objectives (SLOs) are constructed from SLIs.

- Latency → Performance SLO
- Traffic
- Errors

$\frac{SuccessRate}{TrafficRate}$ is Availability Ratio for Δt

- Saturation → Capacity Planing

An Ideal Workflow: SRE Abstraction Layer



Building Service Level Objectives with Prometheus

Articulating a Service Level Objective

1. Service Level Indicator: Usually from a Method
2. Threshold
    - $\leq$ 2 Seconds
    - Boolean True
3. Goal: 95% Availability
4. Time Window: 30 Days

Examples

- The service will have 95% availability over the past 30 days.
- The service will return HTTP API results within 2 seconds or less for 95% of requests over the trailing 30 days.

PromQL: Availability Ratio SLO Example

```
- record: job:http_requests_availability_sli_ratio:rate30d
  expr: >
    sum by (job) (
      rate(http_requests_total{status!~"5.."}[30d])
    )
    /
    sum by (job) (
      rate(http_requests_total[30d])
    )

- alert: ServiceAvailabilitySLO
  expr: >
    job:http_requests_availability_sli_ratio:rate30d < 0.95
  labels:
    severity: page
```

PromQL: Latency SLO Example

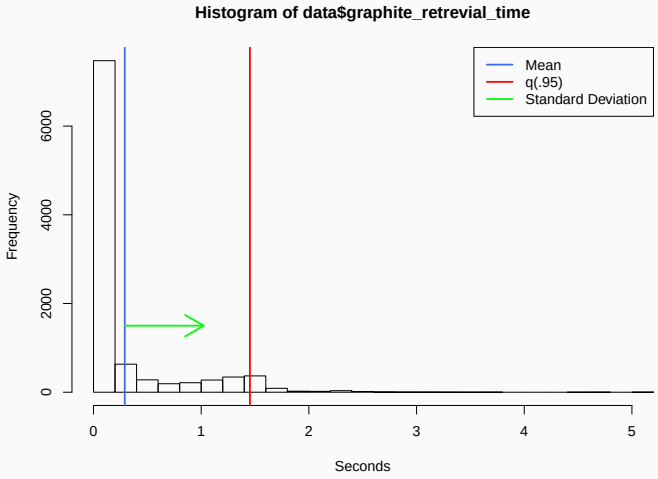
```
- rule job:http_request_duration_seconds_sli:rate30d_q95
  expr: >
    histogram_quantile(0.95, sum by (job, le) (
        rate(
          http_request_duration_seconds_bucket[30d]
        )
    ))

- alert: ServiceLatencySLO
  expr: >
    job:http_request_duration_seconds_sli:rate30d_q95 > 2
  labels:
    severity: page
```

But wait, what's the problem with Histograms?

Prometheus Histograms

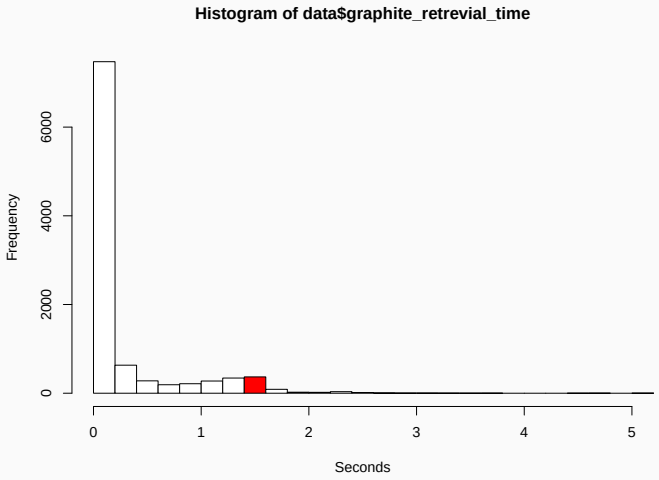
Histograms as a Sketch for Percentile Estimation



R Calculations on Raw Data

- Gamma Distribution
- $\mu = 0.2902809$
- $\sigma = 0.7330352$
- $q(.5) = 0.0487695$
- $q(.95) = 1.452352$
- Index of $q(.95) = 9,501$

Histograms as a Sketch for Percentile Estimation

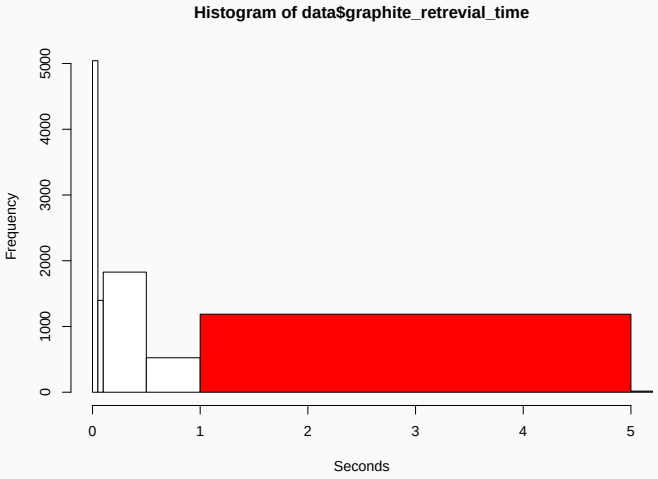


Prometheus Estimation

- $sample = 9,501$
- 8th bucket
- 93rd sample of 368

144 Buckets Stopping at $t = 29$

Histograms as a Sketch for Percentile Estimation

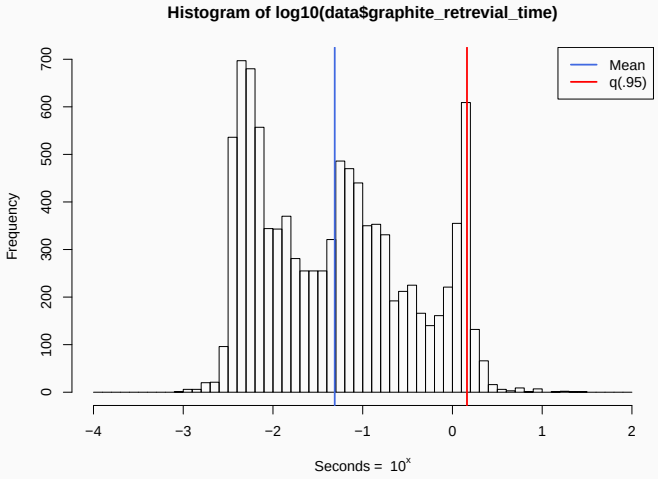


Human Defined Histogram Metric

- $sample = 9,501$
- 5th bucket
- 711th sample of 1187
- Error = 234%

Buckets = $\{.05, .1, .5, 1, 5, 10, 50, 100\}$

Histograms as a Sketch for Percentile Estimation



Log-Linear Bucketing Algorithm

Working Around Prometheus's Limitations

Make the SLO a parameter to the application and export it!

```
http_request_slo_seconds 2
```

Use these 3 Counters:

```
http_requests_total{handler="foo",client="iOS"} 42
http_requests_errors_total{handler="foo",client="iOS"} 3
http_requests_over_slo_total{handler="foo",client="iOS"} 1
```

SLO alert for 95% of events processed under 2 seconds for last 30 days:

```
1 - rate(http_requests_over_slo_total[30d]) /
 rate(http_requests_total[30d])
< 0.95
```

Alerting on SLOs

Alerting Terminology

Reset Time: The time between fixing an incident and the monitoring system resolving the alert. Did the actions really fix the problem?

Detection Time: The time between the start of an incident and generating an alert.

Precision: The proportion of alerts for a significant event compared to false positives. Low traffic conditions can cause a loss of precision.

Recall: The proportion of significant events that resulted in alerts.

Error Budgets

Definition

$$ErrorBudget = 1 - SLO$$

Example

$$0.05 = 1 - 0.95$$

Example in Percentages

$$5\% = 100\% - 95\%$$

5% of requests in the 30 day time window can respond with durations longer than 2 seconds.

Calculating Error Burn Rates

Definition

$$BurnRate = \frac{ErrorRatio\Delta t}{ErrorBudget}$$

Ratio of Error Budget Consumed in Δt

$$BudgetConsumedRatio\Delta t = \frac{t}{SLO\ TimeWindow} \times BurnRate$$

Solve for *BurnRate* to create an alert that fires if 2% of the Error Budget is consumed in the last hour.

PromQL: Burn Rate Example

```
- rule: job:http_requests_over_slo_ratio:rate1h
  expr: >
    sum by (job) (
      rate(http_requests_over_slo_total[1h])
    ) /
    sum by (job) (
      rate(http_requests_total[1h])
    )

- alert: BurnRateHigh
  expr: >
    job:http_requests_over_slo_ratio:rate1h > 14.4 * 0.05
  labels:
    severity: page
  annotations:
    summary: 2% of Error Budget consumed in the last hour
```

Conclusions

Takeway Points

Build a Schema and Kit as guard rails against garbage in garbage out patterns. This produces a rule library.

SLO based measuring has many positive and powerful aspects. As they are based on percentiles we need to get the math right to base good business decisions on good data. Prometheus presents us with challenges here.

SLOs are actually *reports*. More like period progress updates than real time alerting. Also, alerting should fire before we violate our SLO goals.

References and Links

Google's Site Reliability Engineering Book, 2016

Google's Site Reliability Workbook, 2018

42 Lines Consulting, Jack Neely, jjneely@42lines.net

Monarch: Google's Planet-Scale In-Memory Time Series Database, August 2020

Thank You!

devops@42lines.net

Practical Operations Podcast: operations.fm