Monitoring Kubernetes Applications with Prometheus

K Dimple Raja Vamsi
Yash Sethiya

17/06/2023

About Us

K Dimple Raja Vamsi

Senior Member of Technical Staff, VMware

GitHub, LinkedIn

Yash Sethiya

Member of Technical Staff, VMware

GitHub, LinkedIn

Agenda

- What is Monitoring
- What is Prometheus
- Prometheus Hands on
- What’s next?

Monitoring

Collecting, processing, aggregating, and displaying real-time quantitative data about a system, such as query counts and types, error counts and types, processing times, and server lifetimes.

Ref: https://sre.google/sre-book/monitoring-distributed-systems/

Why Monitoring?

- Analyzing trends
- Alerting
- Dashboarding
- Debugging

Ref: https://sre.google/sre-book/monitoring-distributed-systems/

What's and Why's

Monitoring should address what's broken and why it's broken

Ref: https://sre.google/sre-book/monitoring-distributed-systems/

SLIs/SLOs/SLAs

SLA → SERVICE LEVEL AGREEMENT
the agreement you make with your clients or users

SLOs → SERVICE LEVEL OBJECTIVES
the objectives your team must hit to meet that agreement

SLIs → SERVICE LEVEL INDICATORS
the real numbers on your performance

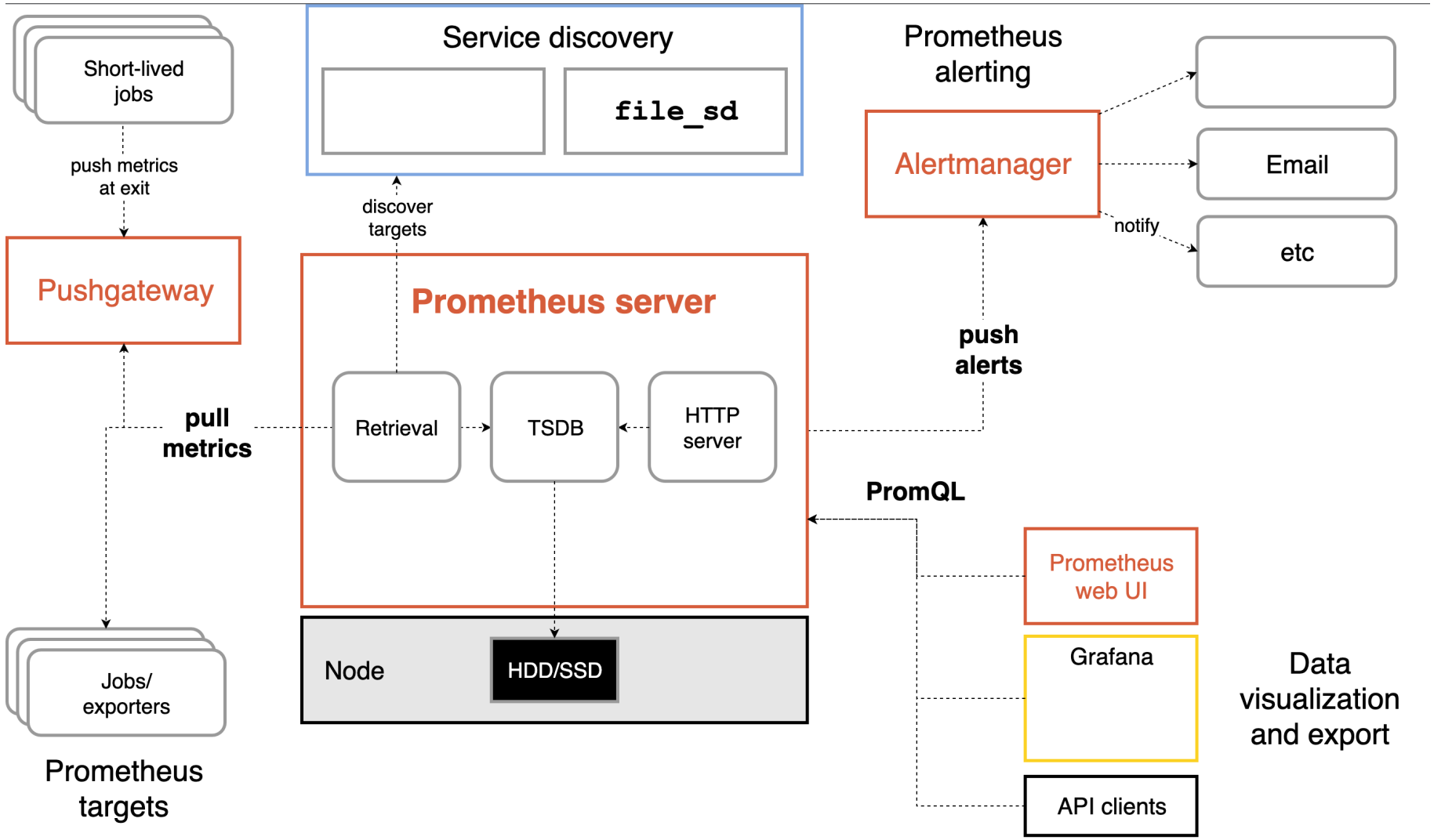
Ref: https://cloud.google.com/blog/products/devops-sre/sre-fundamentals-slis-slas-and-slos
https://www.atlassian.com/incident-management/kpis/sla-vs-slo-vs-sli

Prometheus

- Open-source systems monitoring and alerting toolkit originally built at SoundCloud.
- Prometheus collects and stores its metrics as time series data.
- Time series collection happens via a pull model over HTTP.
- Pushing metrics through Push gateway.
- service discovery or static configuration for pulling the metrics from sources.
- PromQL to query the data

Ref: https://prometheus.io/docs/introduction/overview/

Prometheus Architecture



Ref: https://prometheus.io/docs/introduction/overview/

Prometheus Metrics

- Counter
- Gauge
- Histogram
- Summary

Ref: https://prometheus.io/docs/concepts/metric_types/

Prometheus Counter

- Always positive
- Increasing

Example: Want to measure how many times a particular API is invoked

```
api_invoke_count{api_name="beer",success="false"} 23
api_invoke_count{api_name="beer",success="true"} 7
api_invoke_count{api_name="car",success="true"} 30
api_invoke_count{api_name="delay",success="true"} 30
```


Ref: https://prometheus.io/docs/concepts/metric_types/#counter

Prometheus Gauge

- Increased or decreased.
- Set a value.

Example: How many active http connections

```
# HELP present_connections How many active http connections
# TYPE present_connections gauge
present_connections 3
```


Ref: https://prometheus.io/docs/concepts/metric_types/#gauge

Prometheus Histogram

- Samples the observations into buckets.
- By default will have a **Inf** bucket and total **sum** and observation **count**
- Prometheus histogram is **cumulative**
- Used for Quantile computation on the Server side.

```
# HELP delay_histogram Histogram distrubution of the delay in seconds
# TYPE delay_histogram histogram
delay_histogram_bucket{le="1"} 5
delay_histogram_bucket{le="2"} 9
delay_histogram_bucket{le="3"} 14
delay_histogram_bucket{le="4"} 18
delay_histogram_bucket{le="5"} 24
delay_histogram_bucket{le="6"} 24
delay_histogram_bucket{le="7"} 25
delay_histogram_bucket{le="8"} 27
delay_histogram_bucket{le="9"} 28
delay_histogram_bucket{le="+Inf"} 30
delay_histogram_sum 123
delay_histogram_count 30
```

Example: How many requests are getting processed below a particular threshold

Ref: https://prometheus.io/docs/concepts/metric_types/#histogram

Prometheus Summary

- Samples the observations into buckets(Quantiles calculated on the Client side).
- By default will have a total **sum** and observation **count**

```
# HELP delay_summary Summary of the delay in seconds
# TYPE delay_summary summary
delay_summary{quantile="0.5"} 8
delay_summary{quantile="0.9"} 10
delay_summary{quantile="0.99"} 10
delay_summary_sum 145
delay_summary_count 33
```

Example: How much time it took to process 50%(50th quantile/0.5) of the requests.

Ref: https://prometheus.io/docs/concepts/metric_types/#summary

Labels and Cardinality

- To differentiate the characteristics of the thing that is being measured.
- Initialize the metrics to zero value to avoid missing metrics.
- Each label set (key value pairs) is a time series that consumes RAM, CPU and Disk
  - Don't overuse labels

```
api_invoke_count {api="beer"} 2
api_invoke_count {api="car"} 3
-----------------------------
api_invoke_count_beer 2
api_invoke_count_car 3
```


Ref: https://prometheus.io/docs/practices/naming/#labels
https://prometheus.io/docs/practices/instrumentation/#do-not-overuse-labels

Counter vs Gauge

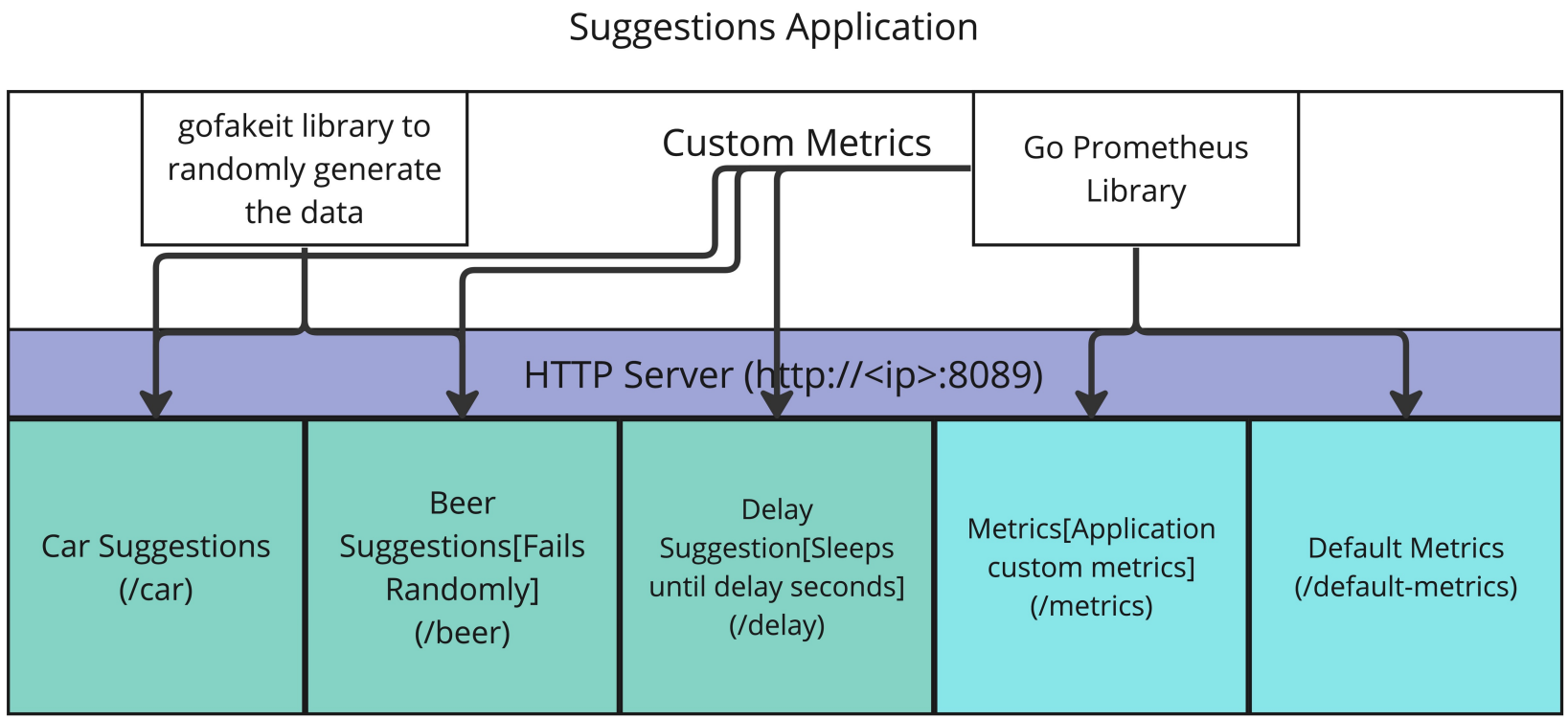
- Gauge can also be used as a counter.
  - Why is a counter needed?
- Gauge represents a state.
- Counter is used to to compute rate at which it is increasing.
  - Using gauge will work but it shouldn't decrease (violates the constraints to compute rate of increase)

Histogram vs Summary

1. If you need to aggregate, choose histograms.
2. Otherwise, choose a histogram if you have an idea of the range and distribution of values that will be observed. Choose a summary if you need an accurate quantile, no matter what the range and distribution of the values is.

Ref: https://prometheus.io/docs/practices/histograms/
https://grafana.com/blog/2022/03/01/how-summary-metrics-work-in-prometheus/

Suggestions Application Overview



Suggestion Application SLIs/SLAs/SLOs

- API Success rate should be greater than 70%
- No of Active connections shouldn't be more that 5

Code and Slides

https://github.com/DimpleRajaVamsi/kcd-prometheus-workshop

PromQL

- Refer to https://github.com/DimpleRajaVamsi/kcd-prometheus-workshop/prometheus/promql_basics.md

Rules

Prometheus supports two types of rules which may be configured and then evaluated at regular intervals

1. Recording rules
2. Alert rules

 Ref: https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/

Recording rules

Allows you to precompute frequently needed or computationally expensive expressions and save their result as a new set of time series

 Ref: https://prometheus.io/docs/prometheus/latest/configuration/recording_rules/

Alerting rules

allow you to define alert conditions based on Prometheus expression language expressions and to send notifications about firing alerts to an external service

 Ref: https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

Alert Manager

Handles alerts sent by client applications such as the Prometheus server. It takes care of deduplicating, grouping, and routing them to the correct receiver integration such as email, PagerDuty etc... It also takes care of silencing and inhibition of alerts.

Ref: https://prometheus.io/docs/alerting/latest/alertmanager/

Alert Manager

1. Configure the Alerting rules
2. Configure Alertmanager
3. Configure prometheus

Ref: https://prometheus.io/docs/alerting/latest/alertmanager/

Alert Manager Configuration

- Routes
- Receivers
- Inhibitors

Alert Manager Receiver Configuration

```yaml
receivers:
  - name: "team-suggestion"
    slack_configs:
      - channel: "#prometheus-alert-manager"
        # Whether to send resolved alerts
        send_resolved: true
```

Alert Manager Route Configuration

```yaml
route:
  group_by: ["severity"]
  receiver: "team-suggestion"
  routes:
    - matchers:
        - api_name="beer"
      receiver: "team-beer"
    - matchers:
        - api_name="car"
      receiver: "team-car"
```

Ref: https://prometheus.io/docs/alerting/latest/configuration/#route
https://prometheus.io/webtools/alerting/routing-tree-editor/

Alert Manager Inhibition Rules

```yaml
inhibit_rules:
  - source_matchers:
      - severity="blocker"
    target_matchers:
      # multiple matchers are ANDed together
      - severity=~"critical|risk"
    # Need to suprees same alert for the same api
    equal: ["alertname", "api_name"]
  - source_matchers:
      - severity="critical"
    target_matchers:
      - severity="risk"
    equal: ["alertname", "api_name"]
```

Exporters

- Useful in cases where we can't instrument the given system to generate Prometheus metrics.
- Node exporter exposes kernel metrics as Prometheus metrics.

Ref: https://prometheus.io/docs/instrumenting/exporters/

Pushgateway

- Useful for for short lived/ephemeral jobs
- Needs to delete the pushed metrics

Ref: https://prometheus.io/docs/instrumenting/pushing/
https://prometheus.io/docs/practices/pushing/

Grafana

Used to Visualize the data and lot more like alerts etc...

Ref: https://grafana.com/oss/grafana/

Grafana Visualization

- Existing Dashboards
  - Node Exporter: https://grafana.com/grafana/dashboards/1860-node-exporter-full/
- Creating New Dashboards and exporting.

Ref: https://grafana.com/grafana/dashboards/

Prometheus Federation

- Allows to scrape metrics from other prometheus instance
- Useful to get a global view when having multiple prometheus instances.

 Ref: https://prometheus.io/docs/prometheus/latest/federation/

Limitations

- Scalability (Vertically scalable)
  - Data retention
- Dashboarding
- Global Visibility

What's Next

- Prometheus Operator
  - Kube Prometheus
  - Kubernetes way of deploying and managing prometheus stack
- Thanos
  - Unlimited storage
  - Global view

Thank you