Between Basics and Depth: Prometheus in Action

Ashish Kumar, Match 16, 2024

- CTO - Cloud and DevOps @ **CloudZenia**
- Cloud Engineer @ **Asecurecloud Inc**
- Organiser @ **CNCG New Delhi**
- Volunteer @ **AWS Delhi NCR User Group**
- AWS Community Builder

Ashish Kumar

TRIVIA TIME

Name three pillars of observability and how are they different?

Metrics
Logs
Traces

What is Prometheus?

Open source Monitoring tool and TSDB (Time Series Database):

- instrumentation
- metrics collection and storage
- querying
- alerting
- dashboarding / graphing / trending

What Prometheus does not do?

- logging
- tracing
- automatic anomaly detection
- long term storage
- automatic scalability
- durability
- internal advanced access management

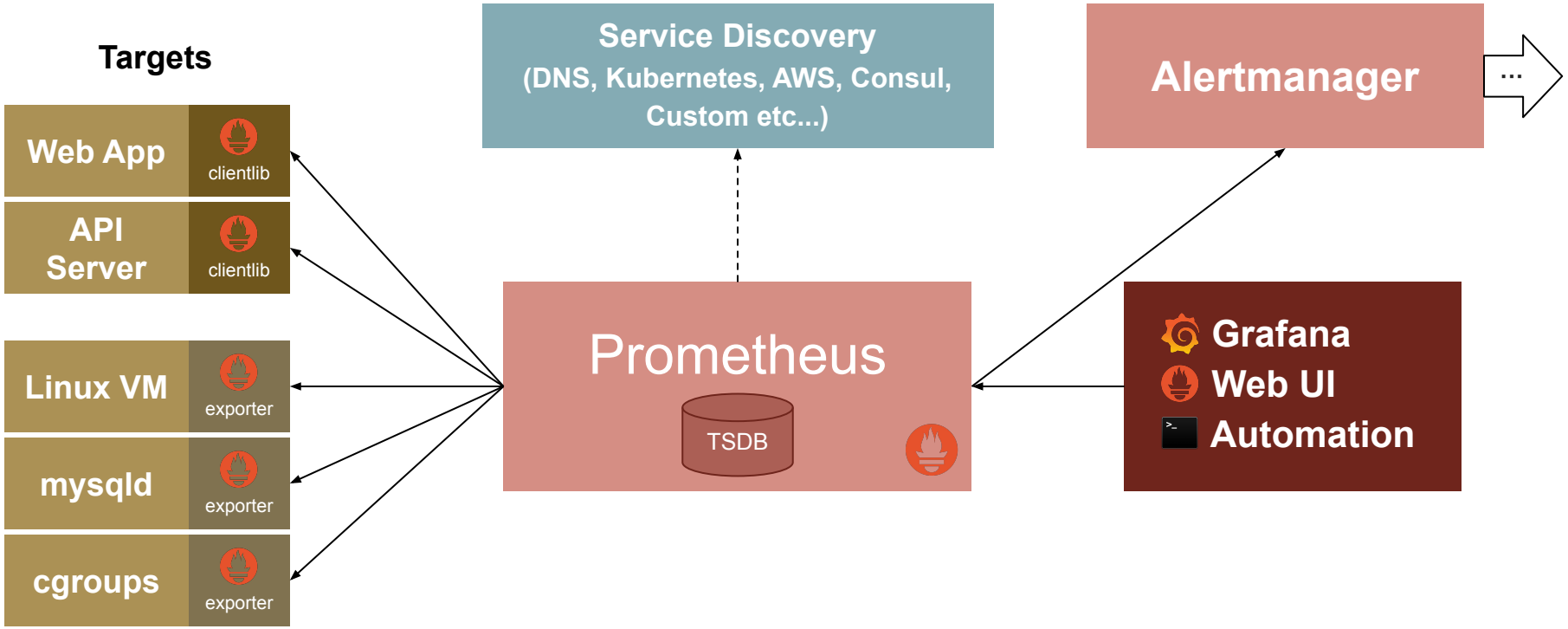
TRIVIA TIME

When and **Where** was Prometheus's inception?

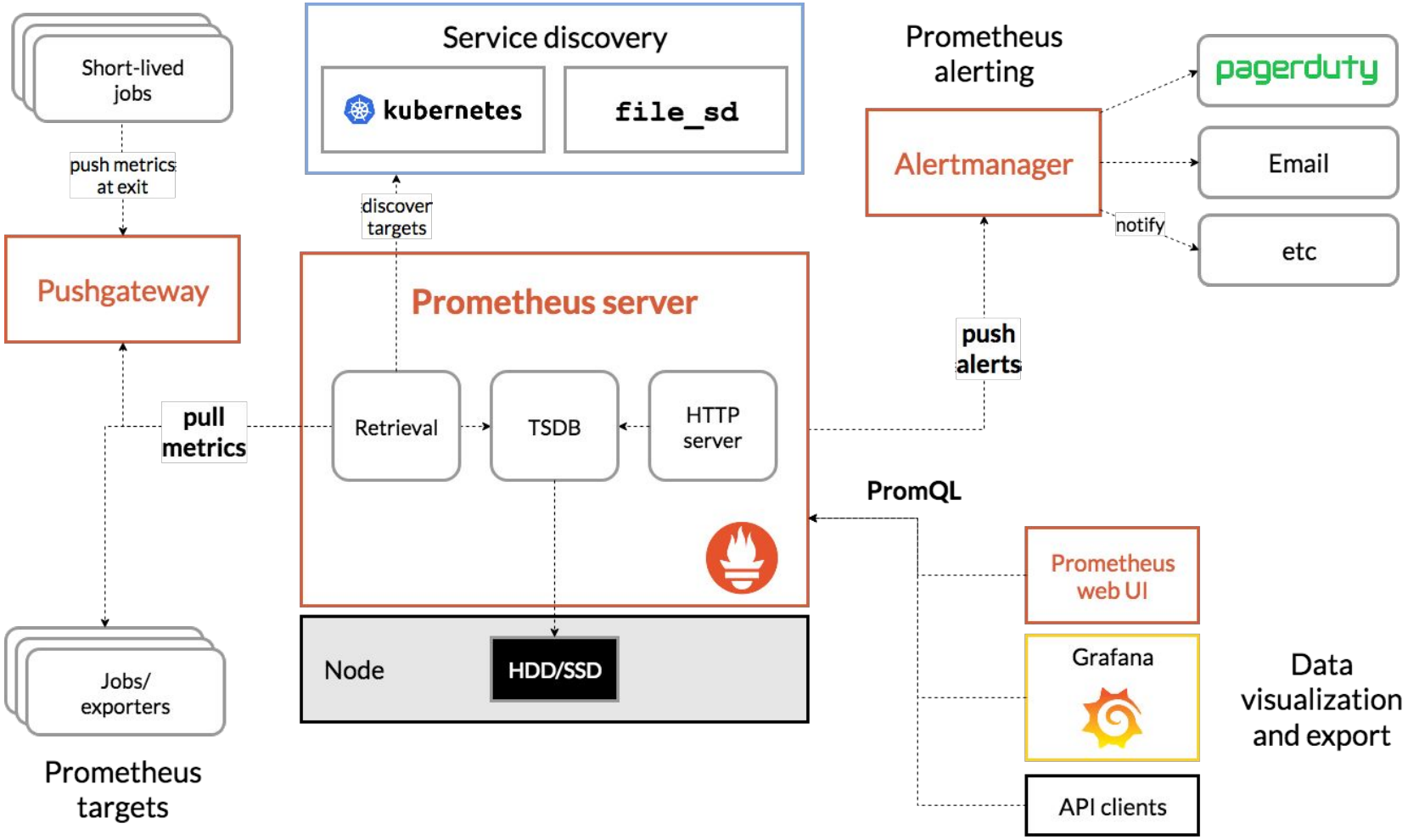
2012
SoundCloud
By Matt and Julis

FYI - 2nd project to be accepted in CNCF and also 2nd to graduate

Step by Step Architecture



Architecture



And that's how we query it

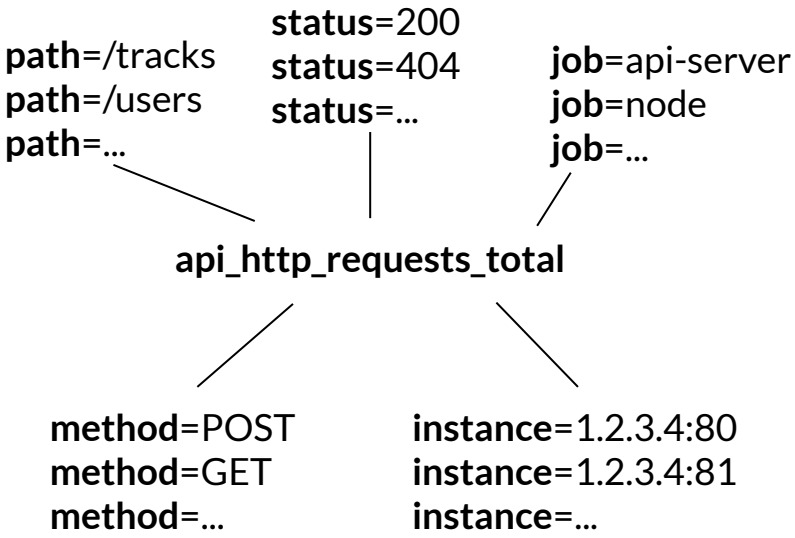
```
# HELP http_requests_total The total number of HTTP requests.
# TYPE http_requests_total counter
http_requests_total{method="post",code="200"} 1027
http_requests_total{method="post",code="400"} 3

# HELP http_request_duration_seconds A histogram of the request duration.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{le="0.05"} 24054
http_request_duration_seconds_bucket{le="0.1"} 33444
http_request_duration_seconds_bucket{le="0.2"} 100392
http_request_duration_seconds_bucket{le="0.5"} 129389
http_request_duration_seconds_bucket{le="1"} 133988
http_request_duration_seconds_bucket{le="+Inf"} 144320
http_request_duration_seconds_sum 53423
http_request_duration_seconds_count 144320

# HELP rpc_duration_seconds A summary of the RPC duration in seconds.
# TYPE rpc_duration_seconds summary
rpc_duration_seconds{quantile="0.01"} 3102
rpc_duration_seconds{quantile="0.05"} 3272
rpc_duration_seconds{quantile="0.5"} 4773
rpc_duration_seconds{quantile="0.9"} 9001
rpc_duration_seconds{quantile="0.99"} 76656
rpc_duration_seconds_sum 1.7560473e+07
rpc_duration_seconds_count 2693
```

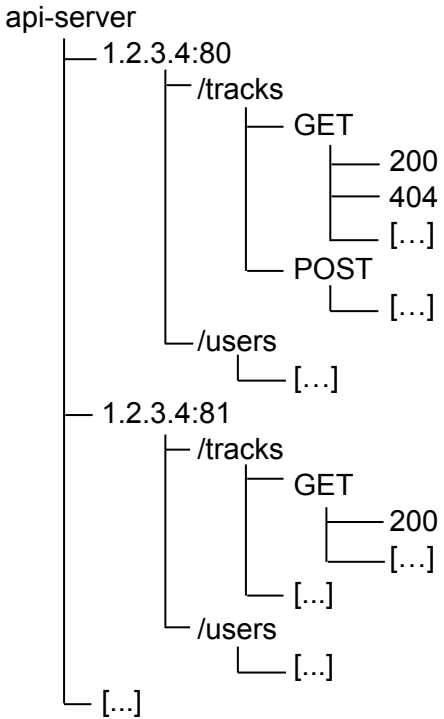
Why Prometheus?

- Dimensional data model
- Powerful query language
- Simple & efficient server
- Service discovery integration

Data Model

Labels > Hierarchy





Labels > Hierarchy

```
api_http_requests_total{method="post"}
```

vs.

```
api-server.*.*.post.*
```

- more flexible
- more efficient
- explicit dimensions

Data Model

What identified a time series?

<identifier> → [(t0, v0), (t1, v1), …]

http_requests_total{job="nginx",instance="1.2.3.4:80",path="/home",status="200"}

metrics name

labels

- Flexible
- No hierarchy
- Explicit dimensions

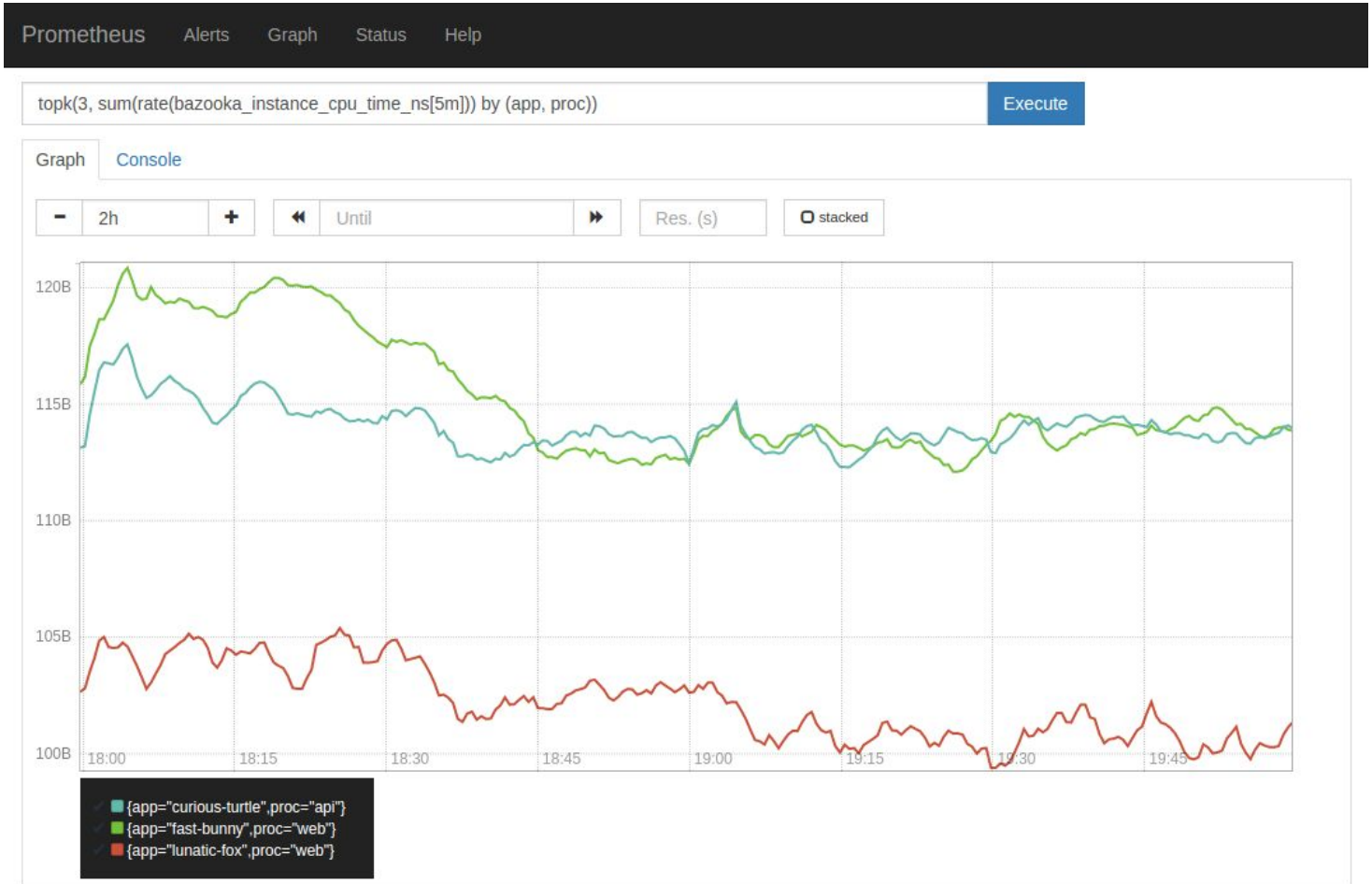
Querying

PromQL

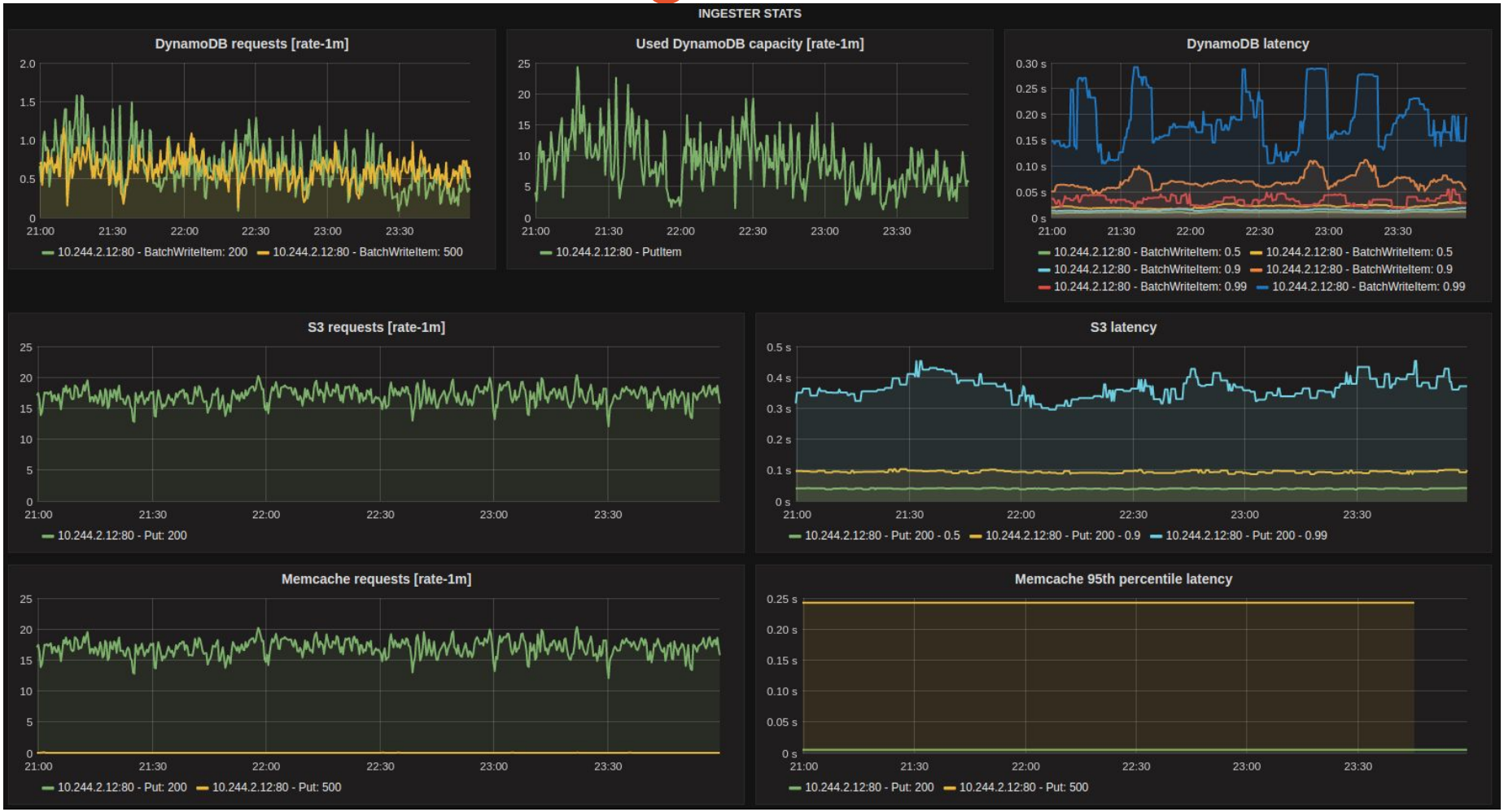
- New query language
- Great for time series computations
- Not SQL-style

- We will see it Demo

Built-in Graphing

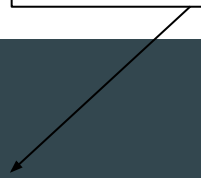


Dashboarding (Not Prometheus)



Alerting

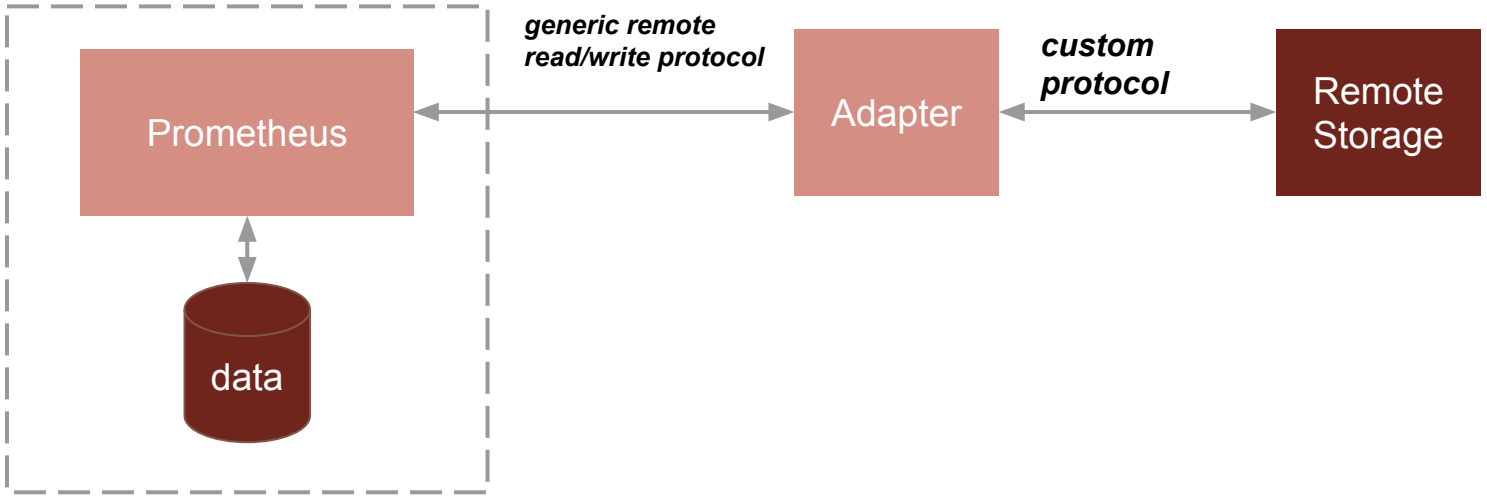
generate an alert for each path with an error rate of >5%



```
alert: Many500Errors
expr: |
  (
      sum by(path) (rate(http_requests_total{status="500"}[5m]))
    /
      sum by(path) (rate(http_requests_total[5m]))
  ) * 100 > 5
for: 5m
labels:
  severity: "critical"
annotations:
  summary: "Many 500 errors for path {{$labels.path}} ({{$value}}%)"
```

Pros pros and pros

- Operational Simplicity
  - Local storage, no clustering
  - HA by running two
  - Static binary
- Decoupled Remote Storage
  - For scalable, durable, long-term storage.
  - E.g.: Cortex, InfluxDB



Non-SQL Query Language

PromQL: `rate(api_http_requests_total[5m])`

SQL: `SELECT job, instance, method, status, path, rate(value, 5m) FROM api_http_requests_total`

PromQL: `avg by(city) (temperature_celsius{country="germany"})`

SQL: `SELECT city, AVG(value) FROM temperature_celsius WHERE country="germany" GROUP BY city`

PromQL: `rate(errors{job="foo"}[5m]) / rate(total{job="foo"}[5m])`

SQL:

```
SELECT errors.job, errors.instance, [...more labels...], rate(errors.value, 5m) /
rate(total.value, 5m) FROM errors JOIN total ON [...all the label equalities...] WHERE
errors.job="foo" AND total.job="foo"
```

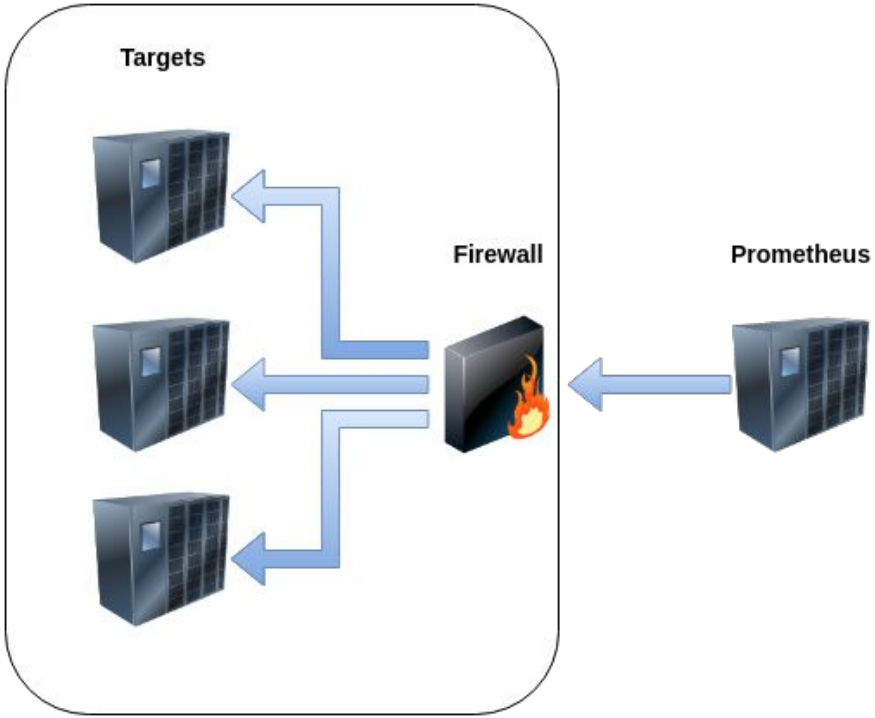
PromQL

- better for metrics computation
- only does reads

Pull vs. Push

- automatic upness monitoring
- horizontal monitoring
- more flexible
- simpler HA
- less configuration
- yes, it scales!

But but...



Alternatives and Workarounds

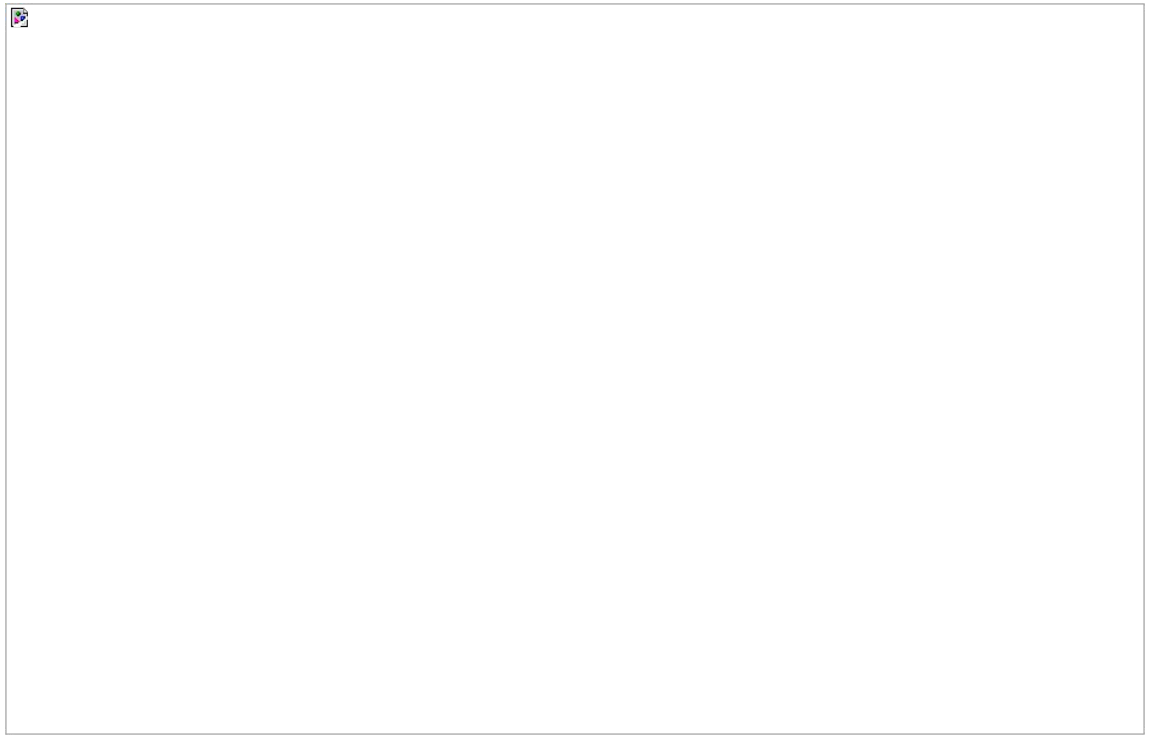
- run Prometheus on same network segment
- open port(s) in firewall / router
- open tunnel / VPN

Uber-Exporters

or...

Per-Process Exporters?

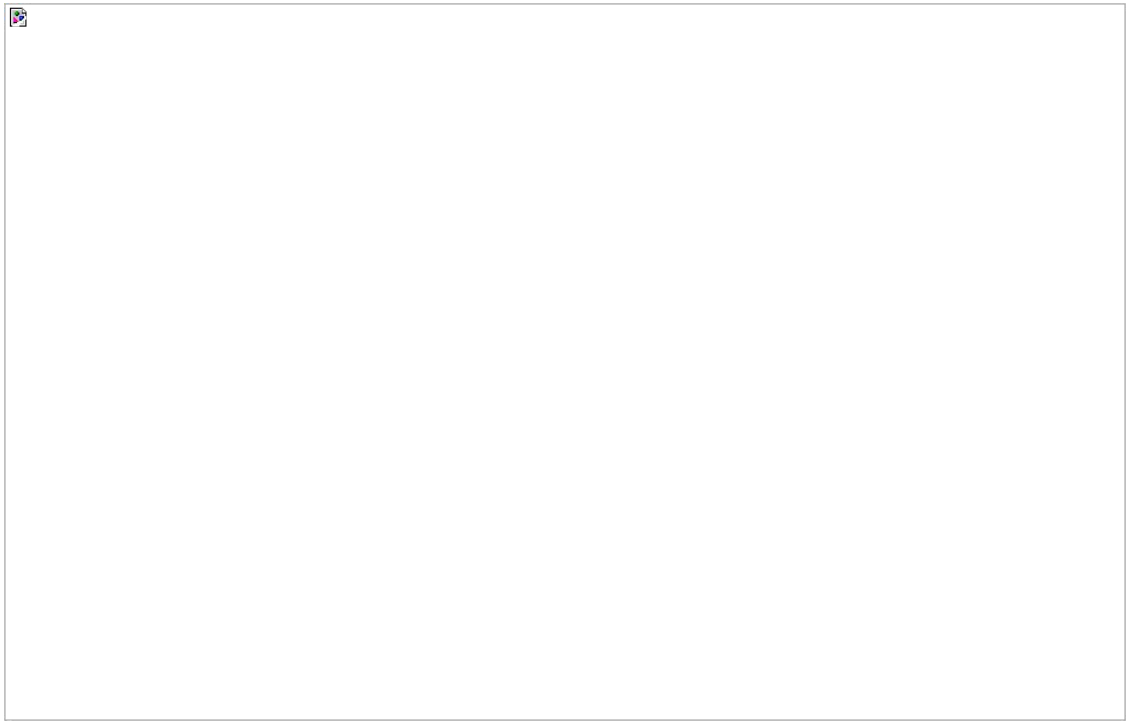
Per-Machine Uber-Exporters



Drawbacks

- operational bottleneck
- SPOF, no isolation
- can’t scrape selectively
- harder up-ness monitoring
- harder to associate metadata

One Exporter per Process



Why not JSON?

We optimized for two extremes:

Text format

- easy to construct
- relatively efficient
- readable
- streamable

Protobuf format

- very efficient
- robust
- streamable

JSON? Worse in all categories.

Relabeling - WTF?

```
relabel_configs:
- source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
  action: keep
  regex: true
- source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scheme]
  action: replace
  target_label: __scheme__
  regex: (https?)
- source_labels: [__meta_kubernetes_service_annotation_prometheus_io_path]
  action: replace
  target_label: __metrics_path__
  regex: (.+)
- source_labels: [__address__, __meta_kubernetes_service_annotation_prometheus_io_port]
  action: replace
  target_label: __address__
  regex: (.+)(?::\d+);(\d+)
  replacement: $1:$2
- action: labelmap
  regex: __meta_kubernetes_service_label_(.+)
- source_labels: [__meta_kubernetes_service_namespace]
  action: replace
  target_label: kubernetes_namespace
- source_labels: [__meta_kubernetes_service_name]
  action: replace
  target_label: kubernetes_name
```

Relabeling - OK...

- a new DSL
- steep learning curve
- ...but very flexible

The alternative:
many special config options.

Stateful Client Libraries

- client libs keep state
- but not much
- manage metrics for you
- pre-aggregation is more efficient

Everything is a float64...

This is crazy! I only need integers!
What about precision?

Everything is a float64...

- it's simpler
- we compress it incredibly well
- float64 integer precision until 2^53

To run into trouble:

Increment counter 1 million times per second for over 285 years

No Clustering?

- really hard to get right
- first thing to fail when you need it (e.g. during network outage)
- keep it simple, focus on operational monitoring
- HA for alerting still easy

Auth or Multi-User?

- focus on great monitoring
- too many different ways
- solve auth externally
- multitenancy is more difficult

Multi cluster Monitoring with Prometheus

- Federated Prometheus
- Mirroring
- Remote Write (**Demo**)
- We can also set it up using other external tools and managed solutions

Feature/Aspect	Federated Prometheus	Mirroring	Prometheus Remote Write
Concept	A global Prometheus scrapes selected metrics from local Prometheus instances.	Each Prometheus scrapes metrics from its own and other clusters.	One Prometheus pushes metrics to another Prometheus instance.
Configuration Complexity	Moderate (Need to specify which metrics to federate)	High (Each Prometheus needs config for all clusters)	Moderate (Configure sender only)
Data Duplication	Low (Only federated metrics are duplicated)	High (All metrics are stored in each Prometheus)	Moderate (Metrics are duplicated between sender and receiver)
Network Load	Moderate (Depends on the number of federated metrics)	High (Each Prometheus scrapes all clusters)	Moderate to High (All metrics are sent to the receiver)
Global View	Partial (Only federated metrics)	Complete (All metrics available everywhere)	Complete at the receiver side
Storage Impact	Low to Moderate (Depends on federated metrics)	High (All metrics stored multiple times)	Moderate (Metrics stored in both sender and receiver)
Scalability	Scales well if only key metrics are federated	Might not scale well with many clusters due to duplication	Scales well, but depends on the capacity of the receiver
Use Case	Aggregating key metrics for a global overview	Complete visibility in every cluster	Centralizing metrics from multiple clusters

Thanks!

REFERENCES

References

- https://github.com/cncf/presentations/tree/main/prometheus
- Prometheus docs
- ChatGPT 4