Getting started with Prometheus & Grafana

Seshagiri Sriram - Aug 26 2028

Prometheus

- A monitoring & alerting system, Inspired by Google’s BorgMon
- Originally built by SoundCloud in 2012
- Open Source, now part of the CNCF
- Simple text-based metrics format
- Multidimensional data model
- Rich, concise query language

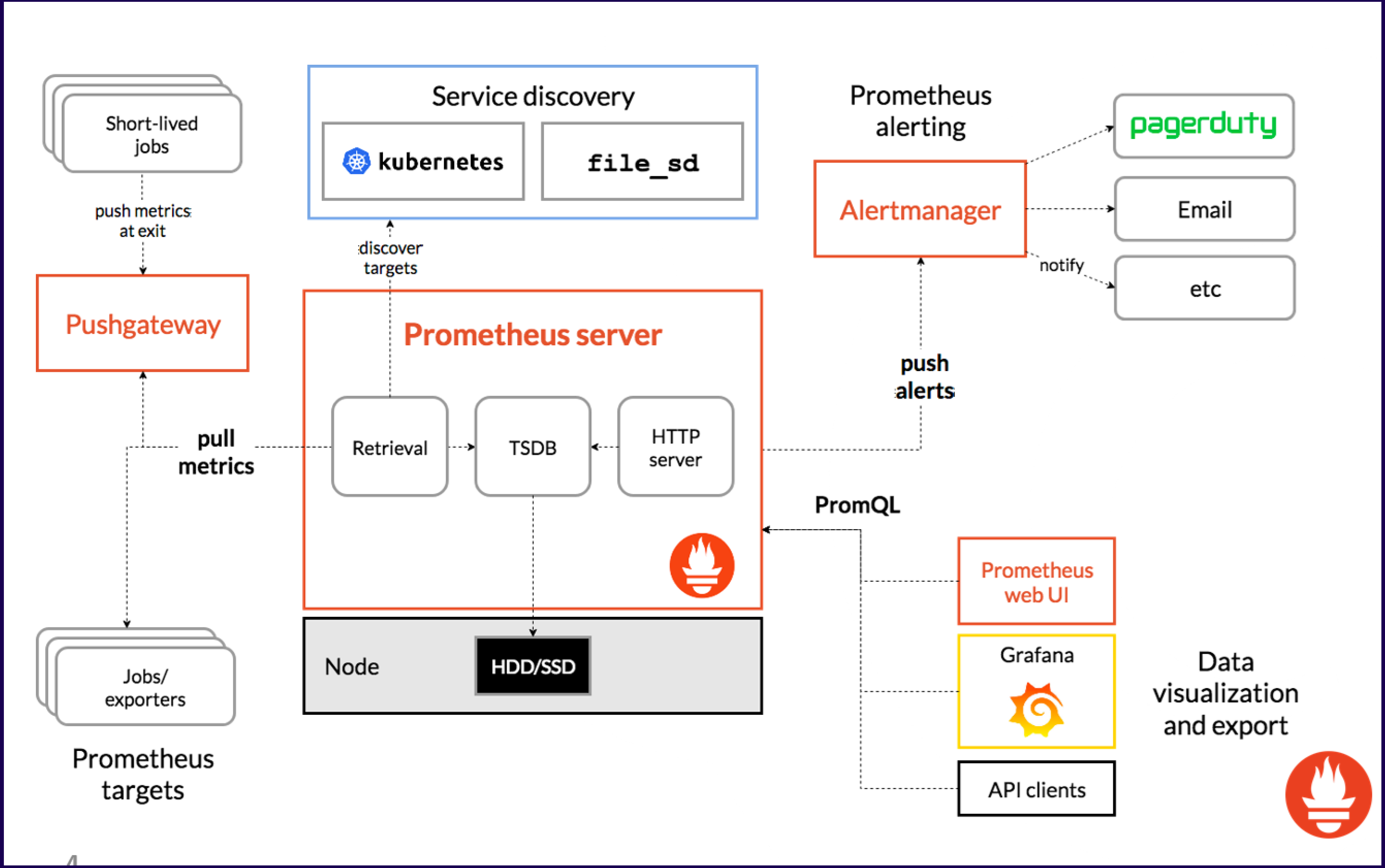
Prometheus

- A data scraper that pulls metrics data over HTTP periodically at a configured interval.
- A time-series database to store ala the metrics data.
- A simple user interface where you can visualize, query, and monitor all the metrics.
- Written in Go, fully published in 2015.

Prometheus

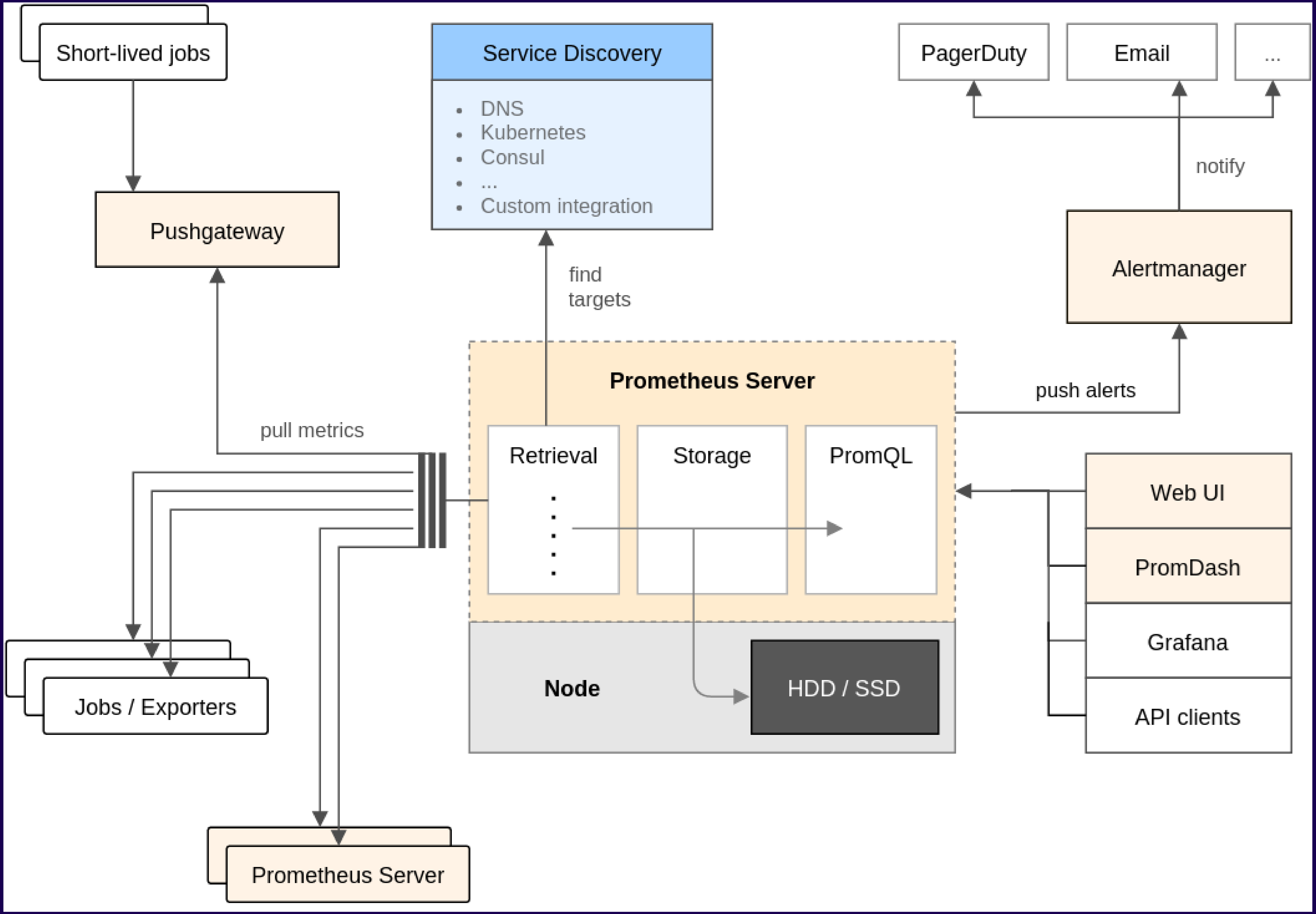
- Monitoring systems and TSDB
  - Instrumentation
  - Metrics collection and Storage
  - Querying
  - Alerting
  - Dashboarding / Graphing / Trending
- Focus on
  - Dynamic Cloud Environments
  - Operational Systems Monitoring

Prometheus



Short-lived jobs
push metrics at exit
Pushgateway
pull metrics
Jobs/ exporters
Prometheus targets
Service discovery
kubernetes
file_sd
discover targets
Prometheus server
Retrieval
TSDB
HTTP server
Node
HDD/SSD
Prometheus alerting
Alertmanager
pagerduty
Email
etc
notify
push alerts
PromQL
Prometheus web UI
Grafana
API clients
Data visualization and export

Prometheus

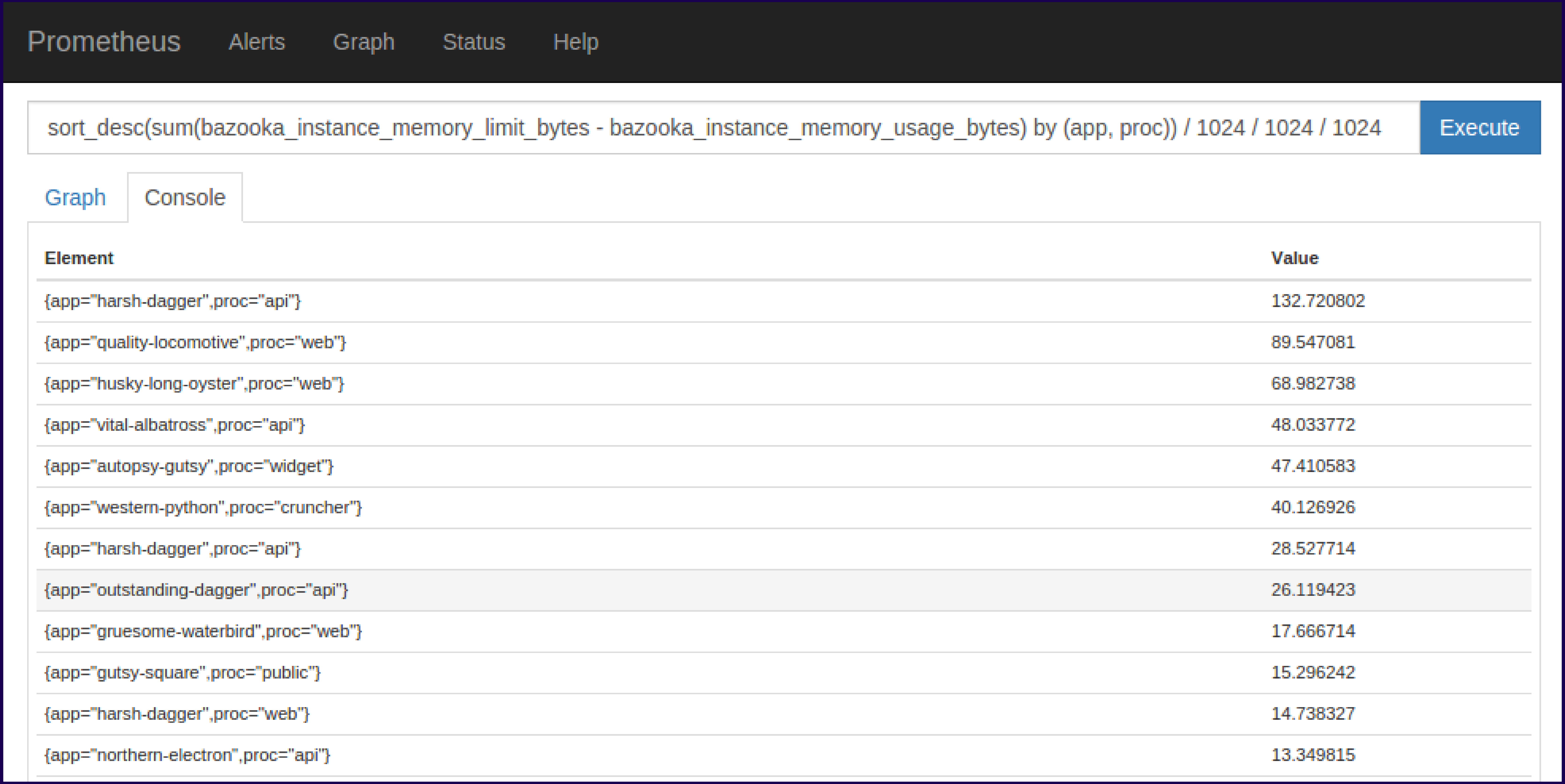

Short-lived jobs
Service Discovery
DNS
Kubernetes
Consul
...
Custom integration
PagerDuty
Email
...
notify
Pushgateway
find targets
Alertmanager
Prometheus Server
push alerts
pull metrics
Retrieval
Storage
PromQL
Web UI
PromDash
Grafana
API clients
Jobs / Exporters
Node
HDD / SSD
Prometheus Server

Prometheus

What I can do	What it cannot do
Dimensional Data Model	Raw Log/event Collection
Powerful Query Language	Request Tracing
Efficiency	Anomaly Detection
Operational Simplicity	Automatic horizontal scaling

User Management and authentication has to be handled separately

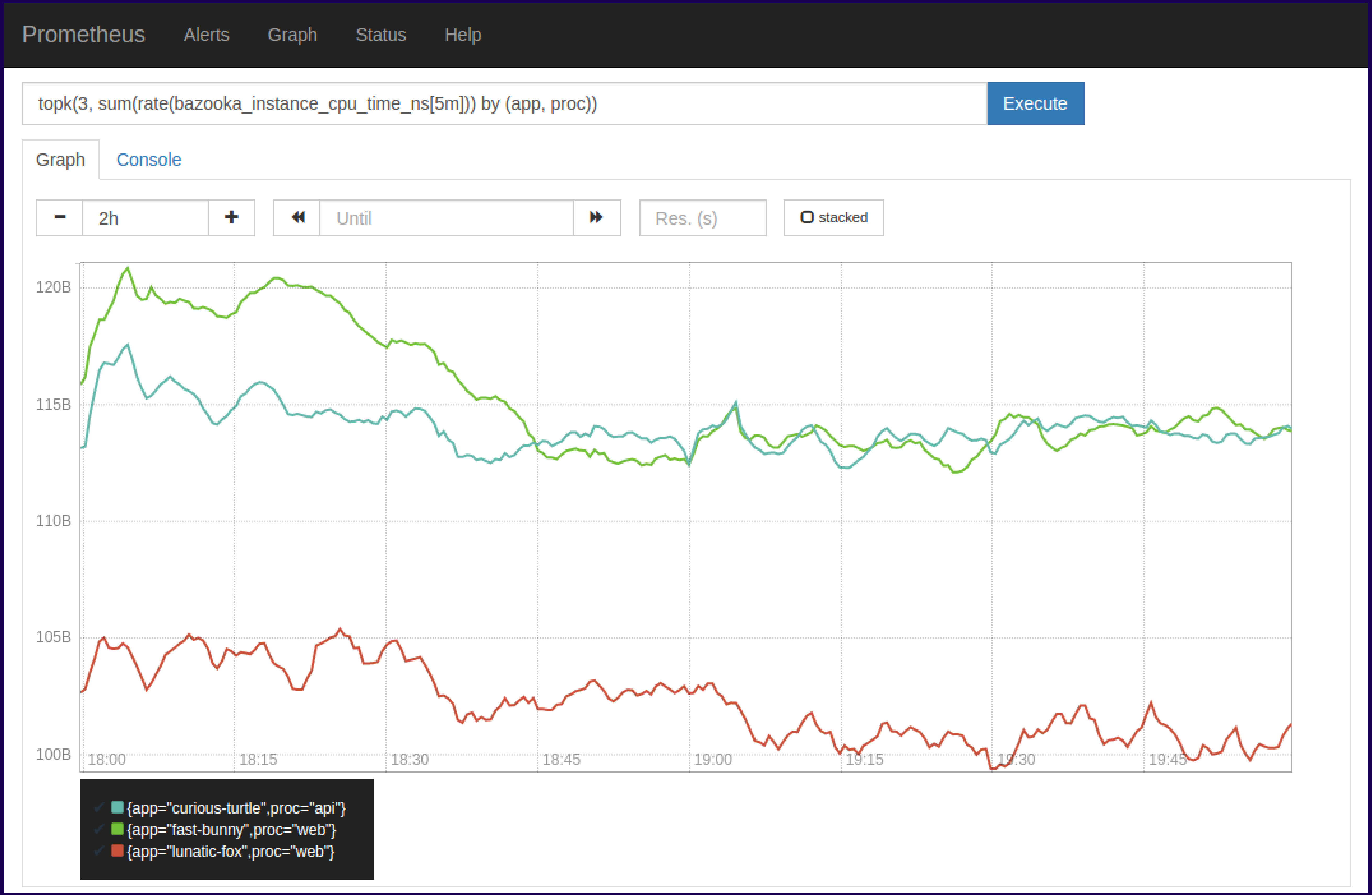
Expression Browser



```
sort_desc(sum(bazooka_instance_memory_limit_bytes - bazooka_instance_memory_usage_bytes) by (app, proc)) / 1024 / 1024 / 1024
```

Element	Value
{app="harsh-dagger",proc="api"}	132.720802
{app="quality-locomotive",proc="web"}	89.547081
{app="husky-long-oyster",proc="web"}	68.982738
{app="vital-albatross",proc="api"}	48.033772
{app="autopsy-gutsy",proc="widget"}	47.410583
{app="western-python",proc="cruncher"}	40.126926
{app="harsh-dagger",proc="api"}	28.527714
{app="outstanding-dagger",proc="api"}	26.119423
{app="gruesome-waterbird",proc="web"}	17.666714
{app="gutsy-square",proc="public"}	15.296242
{app="harsh-dagger",proc="web"}	14.738327
{app="northern-electron",proc="api"}	13.349815

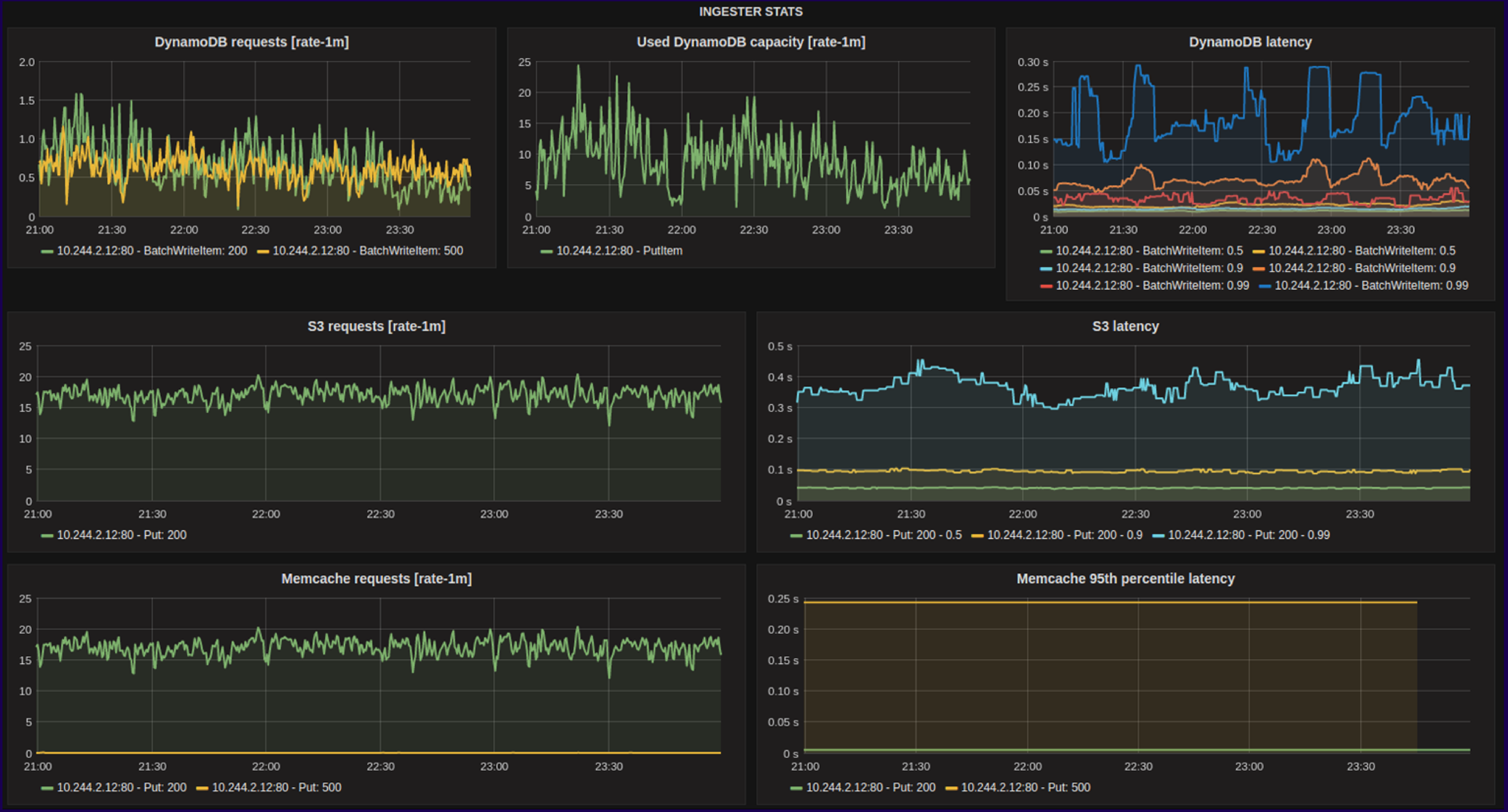
Built in Graphing



```
topk(3, sum(rate(bazooka_instance_cpu_time_ns[5m])) by (app, proc))
```

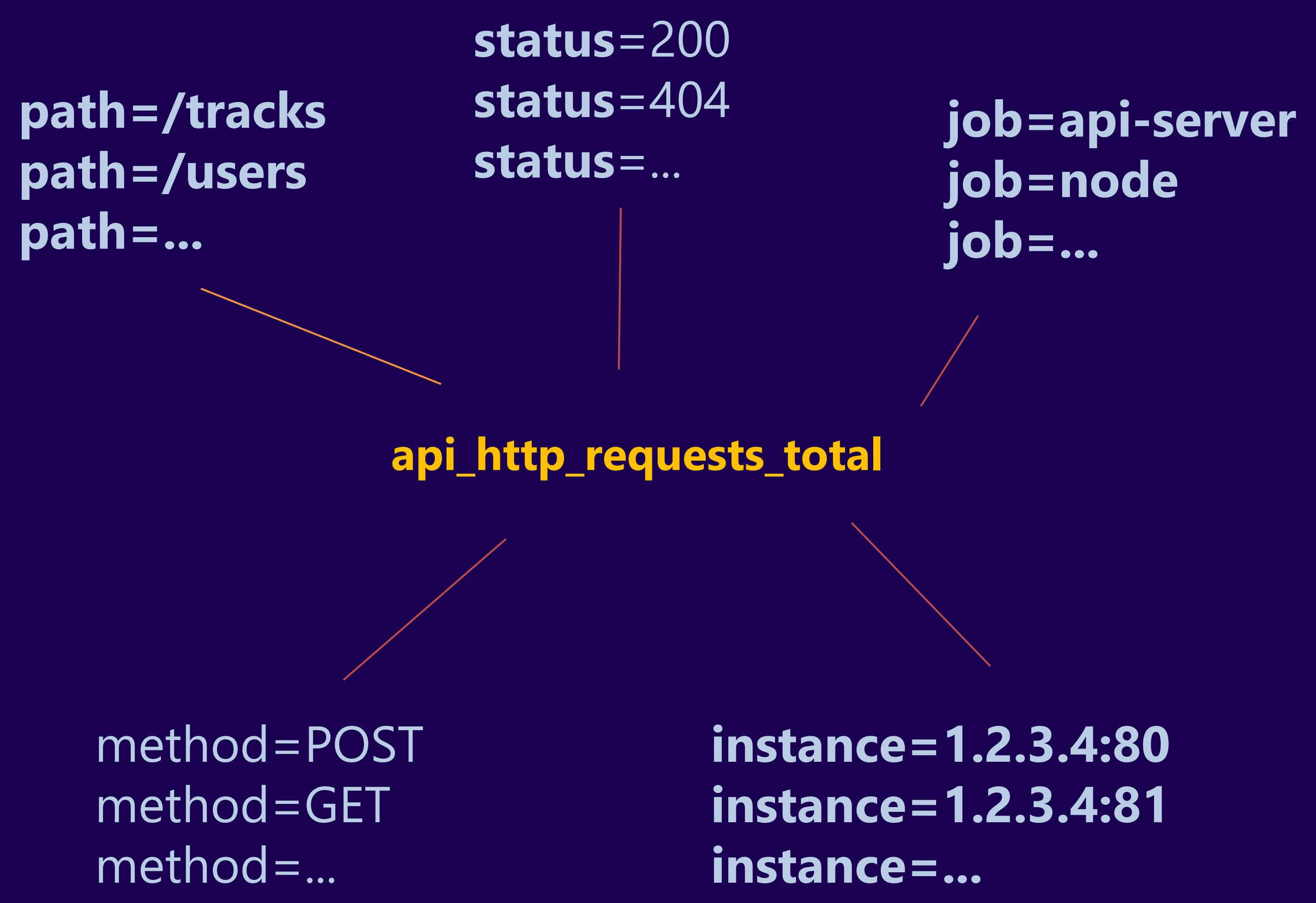
✓ 🟦 {app="curious-turtle",proc="api"}
✓ 🟩 {app="fast-bunny",proc="web"}
✓ 🟥 {app="lunatic-fox",proc="web"}

Grafana Support


INGESTER STATS
DynamoDB requests [rate-1m]
2.0
1.5
1.0
0.5
0
21:00
21:30
22:00
22:30
23:00
23:30
10.244.2.12:80 - BatchWriteItem: 200
10.244.2.12:80 - BatchWriteItem: 500
Used DynamoDB capacity [rate-1m]
25
20
15
10
5
0
10.244.2.12:80 - PutItem
DynamoDB latency
0.30 s
0.25 s
0.20 s
0.15 s
0.10 s
0.05 s
0 s
10.244.2.12:80 - BatchWriteItem: 0.5
10.244.2.12:80 - BatchWriteItem: 0.5
10.244.2.12:80 - BatchWriteItem: 0.9
10.244.2.12:80 - BatchWriteItem: 0.9
10.244.2.12:80 - BatchWriteItem: 0.99
10.244.2.12:80 - BatchWriteItem: 0.99
S3 requests [rate-1m]
10.244.2.12:80 - Put: 200
S3 latency
0.5 s
0.4 s
0.3 s
0.2 s
0.1 s
0 s
10.244.2.12:80 - Put: 200 - 0.5
10.244.2.12:80 - Put: 200 - 0.9
10.244.2.12:80 - Put: 200 - 0.99
Memcache requests [rate-1m]
10.244.2.12:80 - Put: 200
10.244.2.12:80 - Put: 500
Memcache 95th percentile latency
0.25 s
0.20 s
0.15 s
0.10 s
0.05 s
0 s
10.244.2.12:80 - Put: 200
10.244.2.12:80 - Put: 500

Data Model

Labels > Hierarchy



```
api-server
    1.2.3.4:80
        /tracks
            GET
                200
                404
                [...]
            POST
                [...]
        /users
            [...]
    1.2.3.4:81
        /tracks
            GET
                200
                [...]
            [...]
        /users
            [...]
    [...]
```

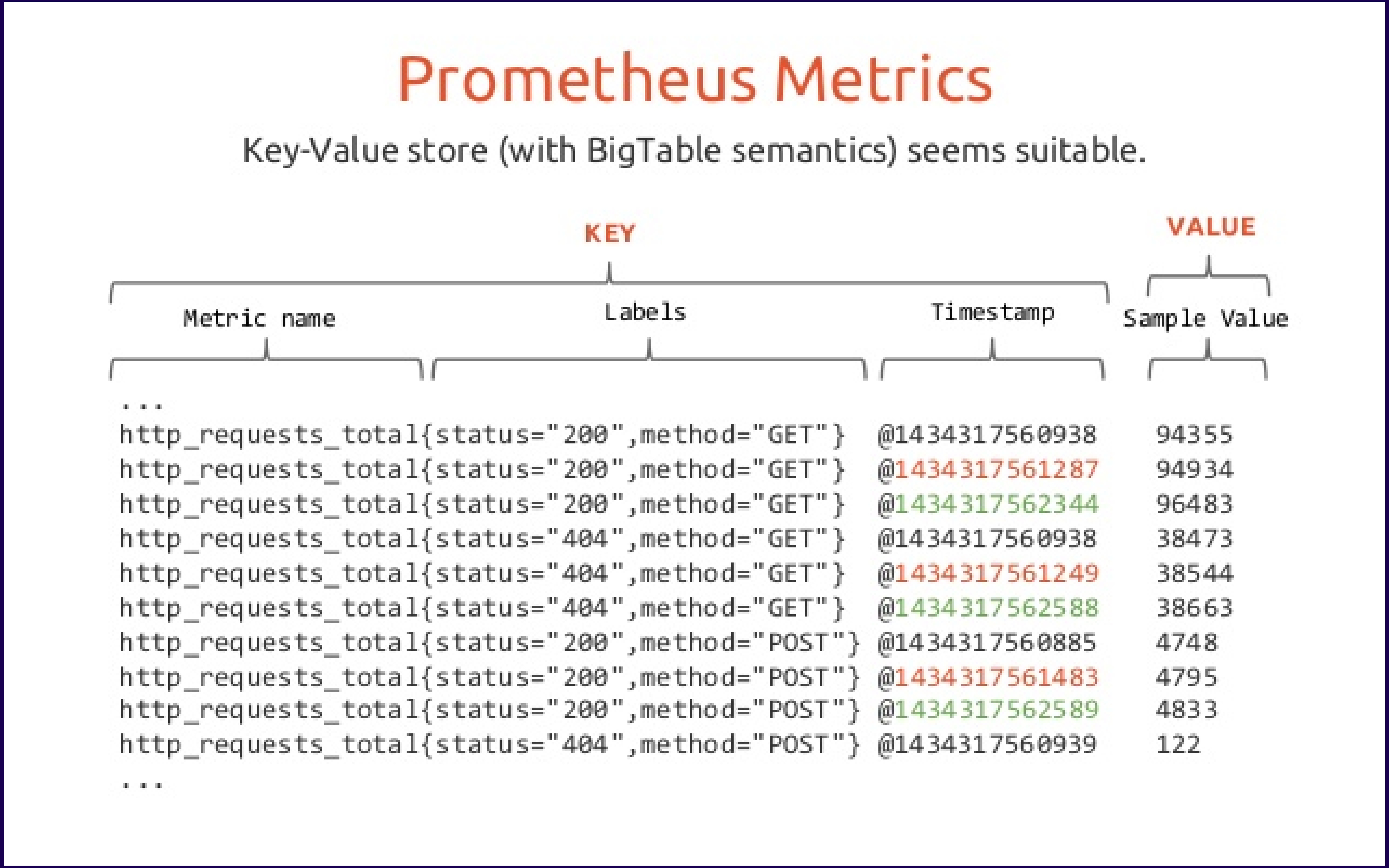
api_http_requests_total{method=”post”}
api-server.*.*.post.*

Prometheus

- Simple Data Model
  <identifier> -> (t_0,v_0), (t_1, v_1), ….. (t_n,v_n)
- Essentially a time series data
- Timestamps are in milliseconds
- Examples of these include: (See the series selectors below)

```
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/home", status="200"}
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/home", status="500"}
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/settings", status="200"}
http_requests_total{job="nginx", instances="1.2.3.4:80", path="/settings", status="502"}
```

Prometheus Metrics



```
...
http_requests_total{status="200",method="GET"}  @1434317560938   94355
http_requests_total{status="200",method="GET"}  @1434317561287   94934
http_requests_total{status="200",method="GET"}  @1434317562344   96483
http_requests_total{status="404",method="GET"}  @1434317560938   38473
http_requests_total{status="404",method="GET"}  @1434317561249   38544
http_requests_total{status="404",method="GET"}  @1434317562588   38663
http_requests_total{status="200",method="POST"} @1434317560885   4748
http_requests_total{status="200",method="POST"} @1434317561483   4795
http_requests_total{status="200",method="POST"} @1434317562589   4833
http_requests_total{status="404",method="POST"} @1434317560939   122
...
```

PROMQL 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 Query Language

- PromQL has a number of features.
- It can select a vector of values, use functions and
- Aggregate by dimension e.g.
  - *sum by (path) (rate(http_requests_total{job="nginx",status =~ "5.."}[1m]))*
- And do binary operations e.g.
  - *sum by (path) (rate(http_requests_total{job="nginx",status =~ "5.."}[1m])) / sum by (path) (rate(http_requests_total{job="nginx"}[1m])*

Metrics

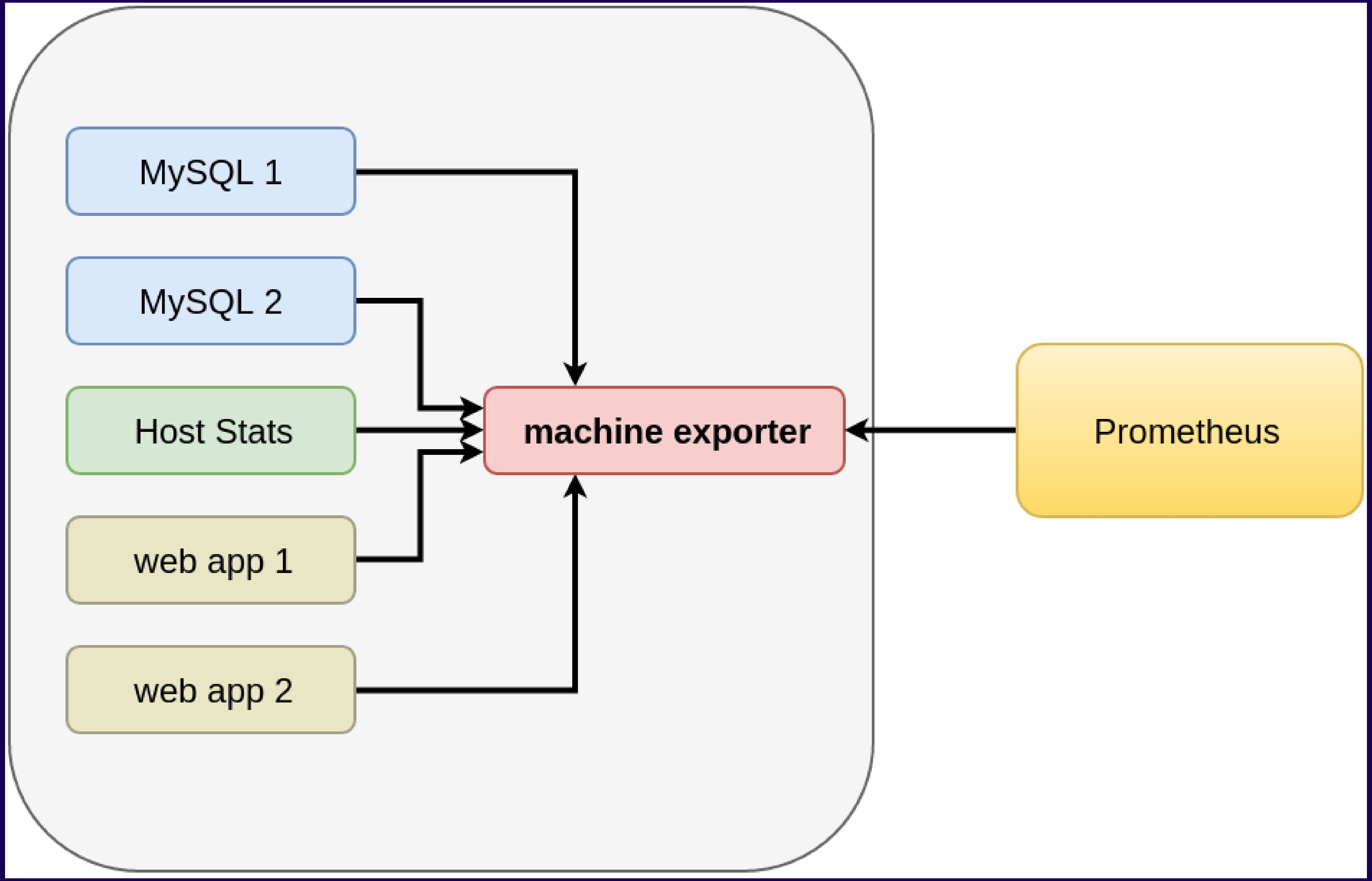
Category of Metrics

- USE
- RED
- AD-HOC

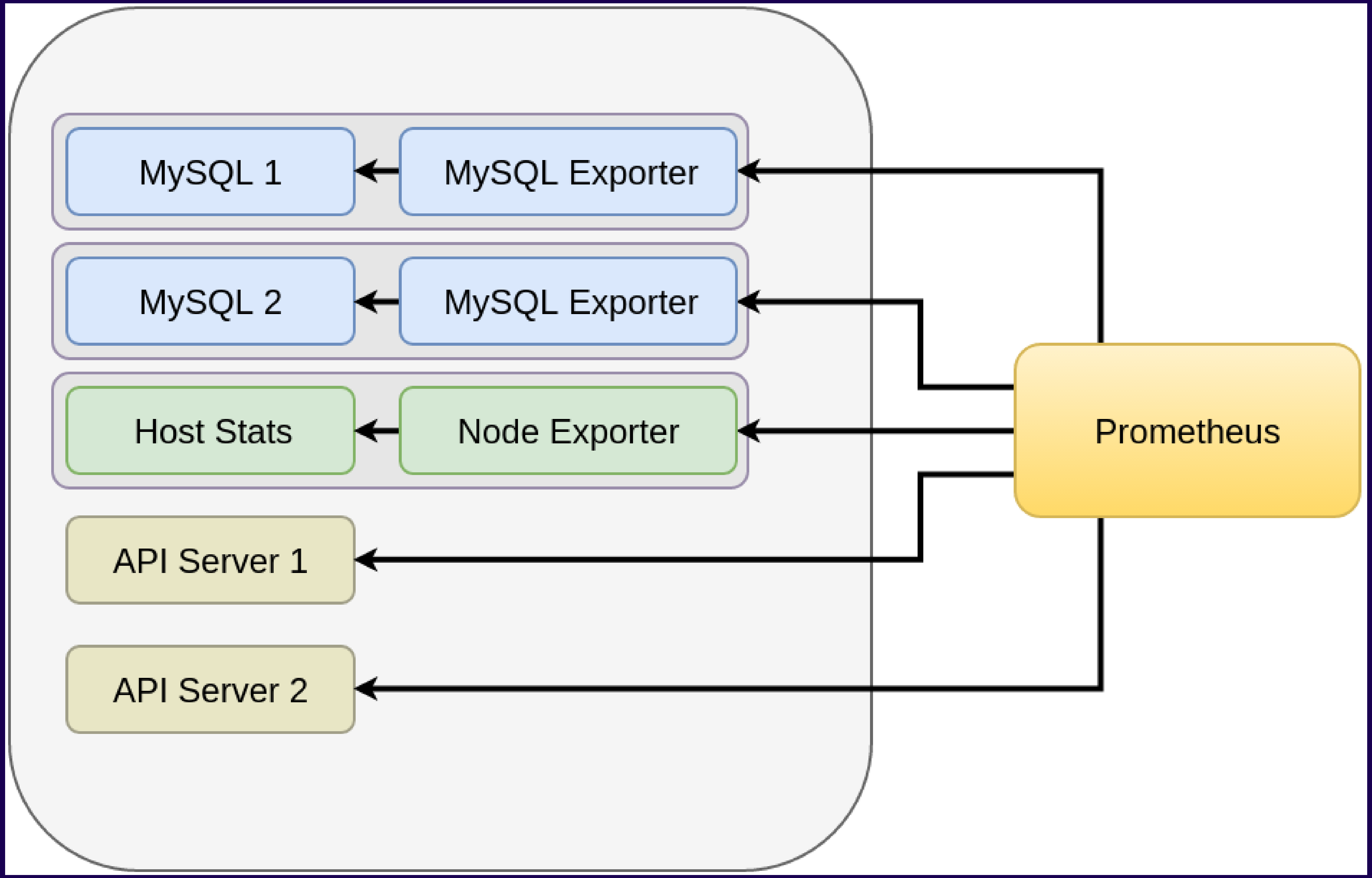
Type of Metrics

- Count
- Gauge
- Histogram

Exporters

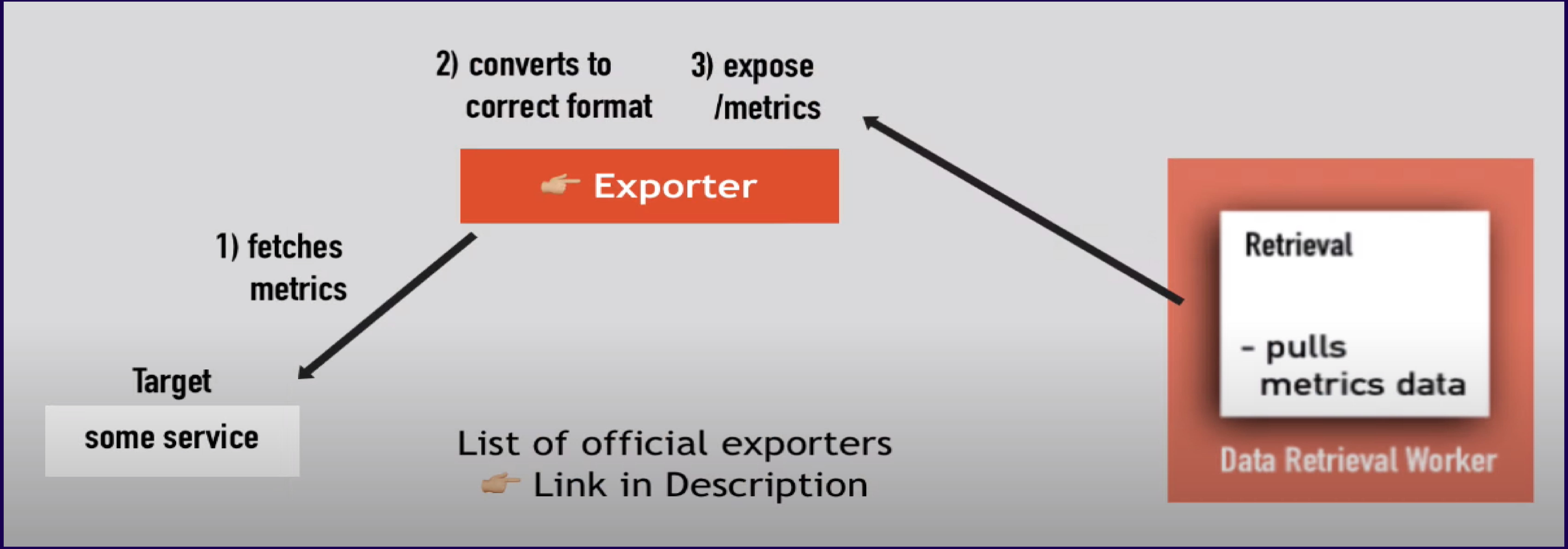

MySQL 1
MySQL 2
Host Stats
web app 1
web app 2
machine exporter
Prometheus

Exporters


MySQL 1
MySQL Exporter
MySQL 2
MySQL Exporter
Host Stats
Node Exporter
API Server 1
API Server 2
Prometheus

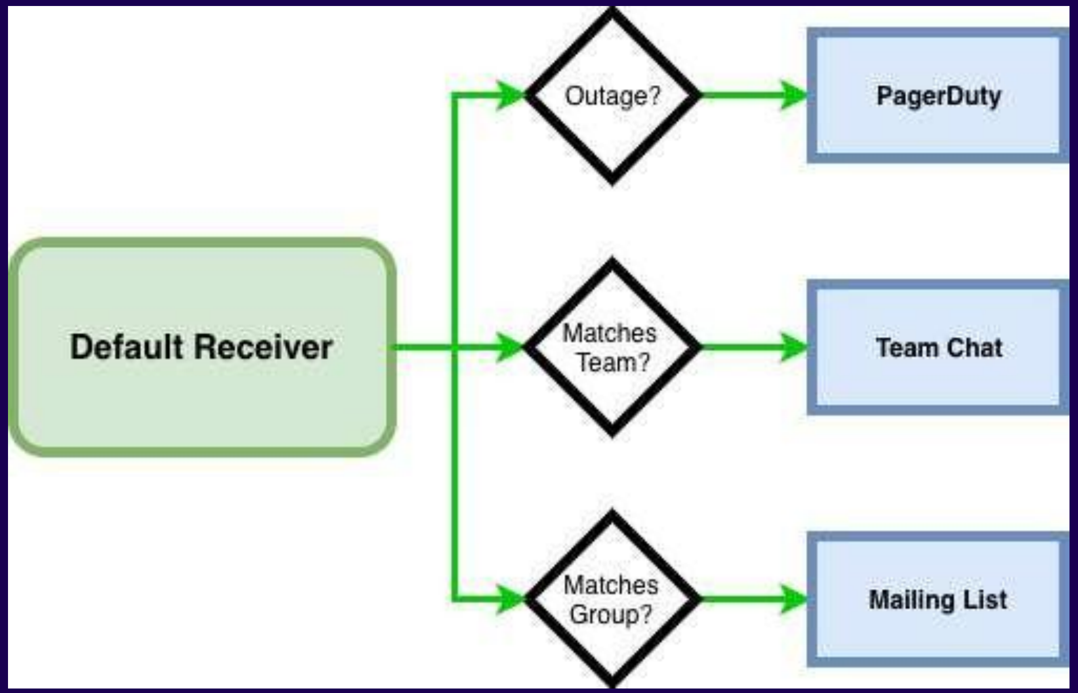
Exporters

A process that exposes Metrics for Prometheus to harvest



The available exporters can be find from here: https://prometheus.io/docs/instrumenting/exporters/

Alert Manager Rules

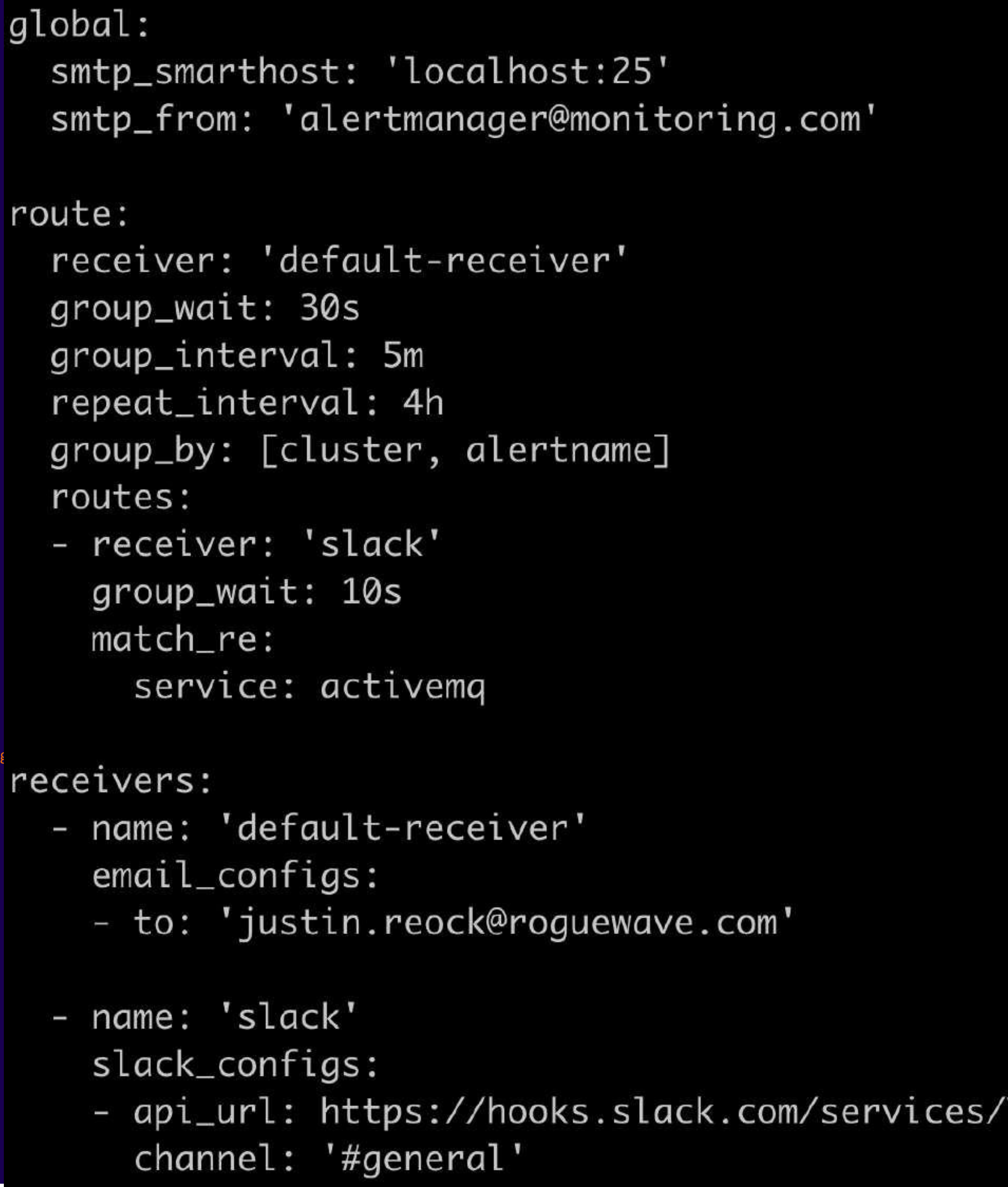


- AlertManager rules are conceptualized as routes, giving you the ability to write sophisticated sets of rules to determine where notifications should end up
- A default receiver should be configured for every notification, and then additional services can be configured through child routes which will match certain conditions

A full configuration reference is available here: https://prometheus.io/docs/alerting/configuration

Alert Manager Rules

- Our config YAML file will be responsible for **setting up routing rules** that will determine how events are triaged
- As mentioned before, all events should **start with a default receiver**, called default-receiver, which will be the starting point for any route
- From there, any number **of sub-receivers can be configured**
- **Sample Configuration one called 'slack'** which will be invoked when the "service" tag of the event that has been triggered matches "activemq"
- Next, **configure our receivers**
- Sample Slack receiver config will **contain WebHook** into Slack



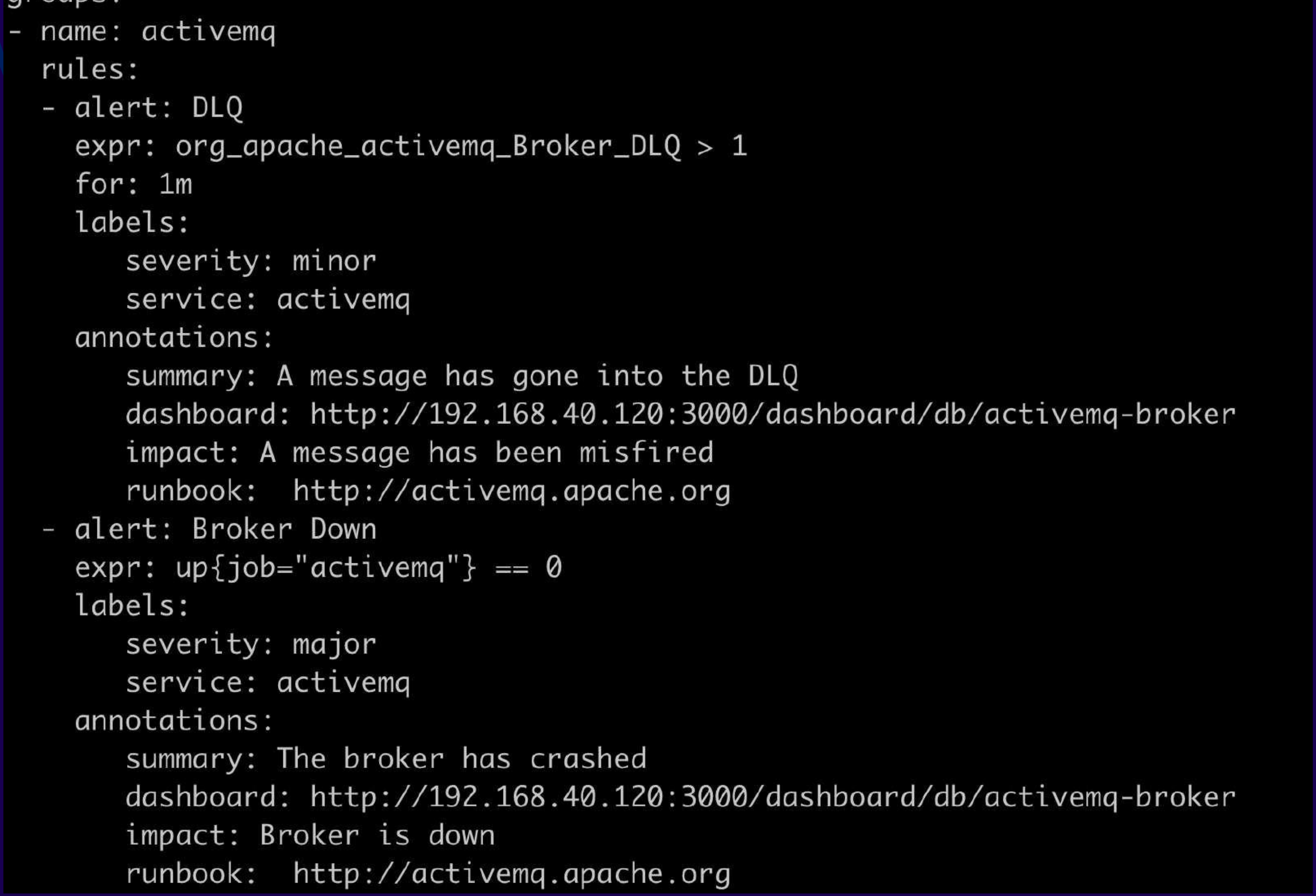
```
global:
  smtp_smarthost: 'localhost:25'
  smtp_from: 'alertmanager@monitoring.com'

route:
  receiver: 'default-receiver'
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  group_by: [cluster, alertname]
  routes:
  - receiver: 'slack'
    group_wait: 10s
    match_re:
      service: activemq

receivers:
  - name: 'default-receiver'
    email_configs:
    - to: 'justin.reock@roguewave.com'

  - name: 'slack'
    slack_configs:
    - api_url: https://hooks.slack.com/services/
      channel: '#general'
```

Alert Manager Rules

Configure Sample Rules

- **two simple events**, but, events can be created out of a **huge range of possible query configurations**

Alert M



```
groups:
- name: activemq
  rules:
  - alert: DLQ
    expr: org_apache_activemq_Broker_DLQ > 1
    for: 1m
    labels:
       severity: minor
       service: activemq
    annotations:
       summary: A message has gone into the DLQ
       dashboard: http://192.168.40.120:3000/dashboard/db/activemq-broker
       impact: A message has been misfired
       runbook:  http://activemq.apache.org
  - alert: Broker Down
    expr: up{job="activemq"} == 0
    labels:
       severity: major
       service: activemq
    annotations:
       summary: The broker has crashed
       dashboard: http://192.168.40.120:3000/dashboard/db/activemq-broker
       impact: Broker is down
       runbook:  http://activemq.apache.org
```

Integrating with Prometheus

15s

- **configure Prometheus to push** alert events into AlertManager
  - Add an alerting section to the Prometheus YAML file
- **Update prom- amq.yml** configuration file from earlier to integrate with our newly configured AlertManager instance
- Upon restarting Prometheus, we **should see our alerts** in the Prometheus dashboard