



# Cloud native monitoring with Prometheus

## From traditional approach to observability



Beatriz Martínez

IBMer, insatiable learner, passionate about technology and innovation #AI #Cloud #devOps #CrossFit  
@beatrizmrg 

# Talk overview

## 1. Prometheus

Monitoring from the way we used to, to cloud-native world

## 2. Monitoring systems and applications

Metrics scraping, querying, graphing, instrumenting prometheus-native apps.

## 3. Alerting and Integrations

Decoupling, grouping, silencing, alerting.

## 4. Scaling and ecosystem

Federation, Thanos, k8s operator, Signifai

**commit**

MAD · NOV 22-23 · 2019

# 1. Prometheus

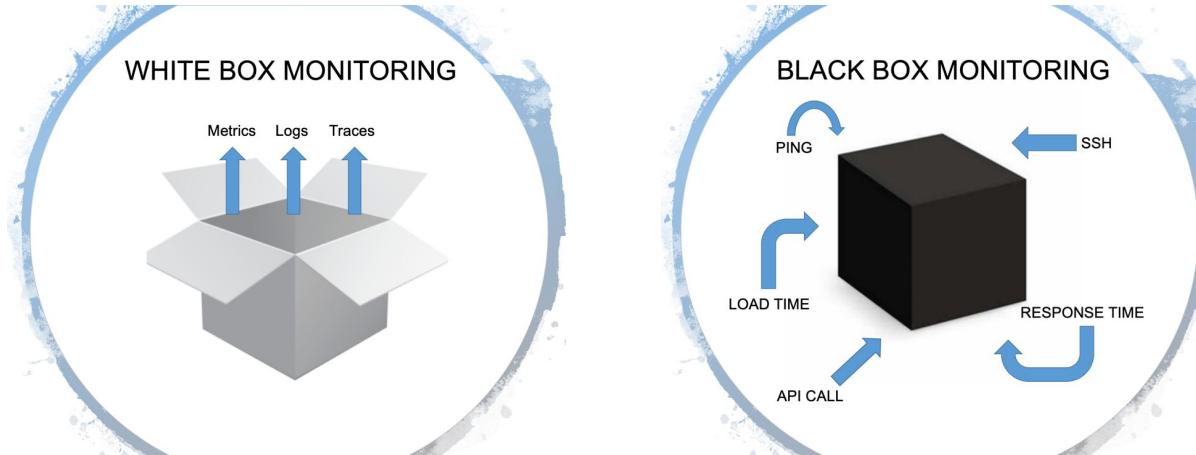
Monitoring from the way we used to, to cloud-native world

# 1.1 Monitoring

What do we understand for monitoring?

Ensure both systems and applications works as they should.

- Automatic or manual monitoring systems.
- Blackbox or Whitebox systems.



# 1.2 Monitoring systems space

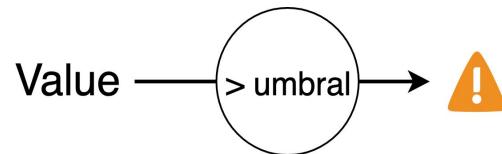
## Well known monitoring systems

We used to ensure the system health based on some basic checks available such as:

- CPU load
- Memory use
- Disk use
- Network use

Monitoring systems with this behaviour:

- Nagios (1999)
- Sensu



# 1.2 Monitoring systems space

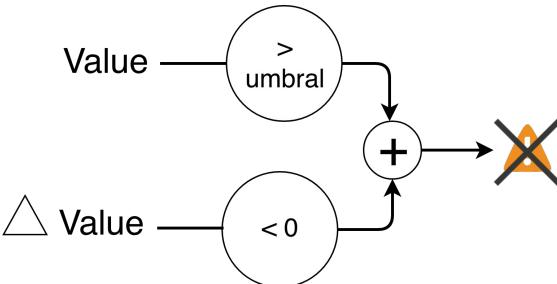
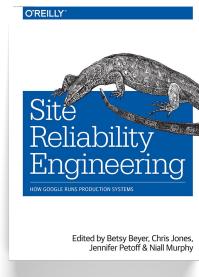
## Today necessities

Today must-have metrics we should be obtaining out of any system (RED method):

- Request rate
- Error rate
- Duration, response time (Latency)

Monitoring systems metrics based:

- Graphite
- Prometheus



Allow alerts based on more complex queries and correlated data:

High CPU System A	High Error Rate System B
High Duration	High Request Rate

# 1.2 Monitoring systems space

## Observability



- know when **things go wrong**
- Be able to **debug** and gain **insight**
- Trending to see changes over time, and **drive technical/business decisions**
- **Feed into other systems** (QA, security, automation, ...)

# 1.2 Monitoring systems space

## Working with metrics

In order to get metrics of the system or app insight:

- Logs based metrics
- Specific **metrics** systems

Even being able to graph same events with both, they **complement** and are used for specific goals.

Logs always have higher **computational cost**, so correlations and complex queries made with metrics would be preferable.

```
x.x.x.90 - - [13/Sep/2006:07:01:53 -0700] "PROPFIND /svn/[xxxx]/Extranet/branches/SOW-101 HTTP/1.1" 401 587
x.x.x.90 - - [13/Sep/2006:07:01:51 -0700] "PROPFIND /svn/[xxxx]/[xxxx]/trunk HTTP/1.1" 401 587
x.x.x.90 - - [13/Sep/2006:07:00:53 -0700] "PROPFIND /svn/[xxxx]/[xxxx]/2.5 HTTP/1.1" 401 587
x.x.x.90 - - [13/Sep/2006:07:00:53 -0700] "PROPFIND /svn/[xxxx]/Extranet/branches/SOW-101 HTTP/1.1" 401 587
```

# 1.3 Prometheus

The Development History of this instrumentation and monitoring stack.



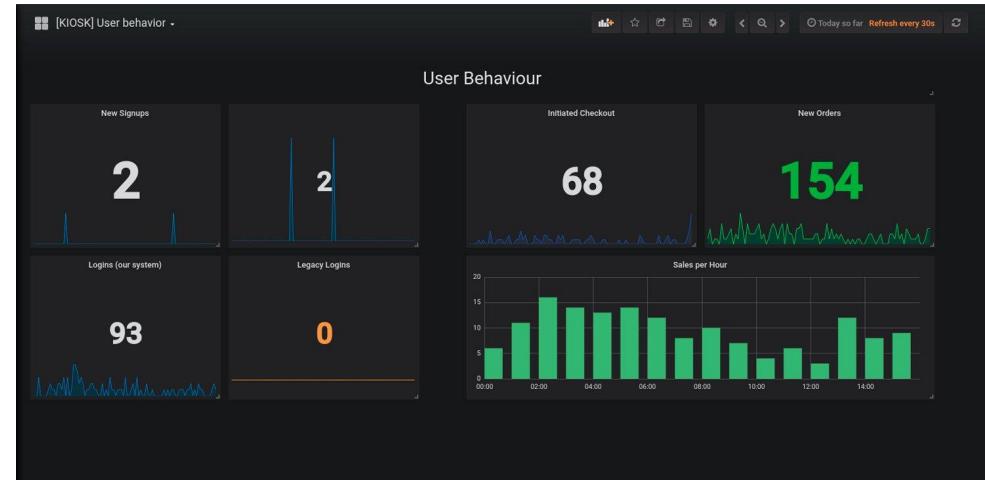
- Inspired by Google's Borgmon monitoring system.
- **Started in 2012** by ex-Googlers working in Soundcloud as an **open source** project
- Mainly written in **Go**.
- Publicly **launched** in early **2015**
- Today continues to be independent of any company.
- K8s and Prometheus, along with envoy were the only three **graduated CNCF projects**, coreDNS, containerd and Fluentd joined this year, more recently Jaeger and Vitess.

# 1.3 Prometheus

## Why Prometheus?

It has knowledge about what the world should look like (which endpoints should exists, what time series means trouble, ..) and actively tries to find faults.

- Whitebox: Instrumenting applications and systems.
- Metrics based: **gaining insights** both systems and **business**.
- Manageable and Reliable
- Advanced service discovery
- Efficient: single server can handle millions of metrics, hundreds of thousands of datapoints per second
- Scalable
- Easy to integrate with
- ....



# 1.3 Prometheus

Becoming the standard

Prometheus metrics:

- Databases
- Hardware related
- Messaging systems
- Storage
- HTTP
- APIs
- Logging
- ....

Official exporters:



Native:



# 1.3 Prometheus

## Becoming the standard

### Open Metrics Project

- Standard based on prometheus exposing format.
- Exploit metrics beyond Prometheus.
- Accepted by **Cloud Native Computing Foundation**. Sandbox (early-stage project) Mars 2018.
- Aims to bring useful metrics to cloud-native deployments
- At its core, is an effort to develop a neutral metrics exposition format.
- Define an **efficient way to transport the metrics over the wire**, and a flexible way to attach information to them: label sets.

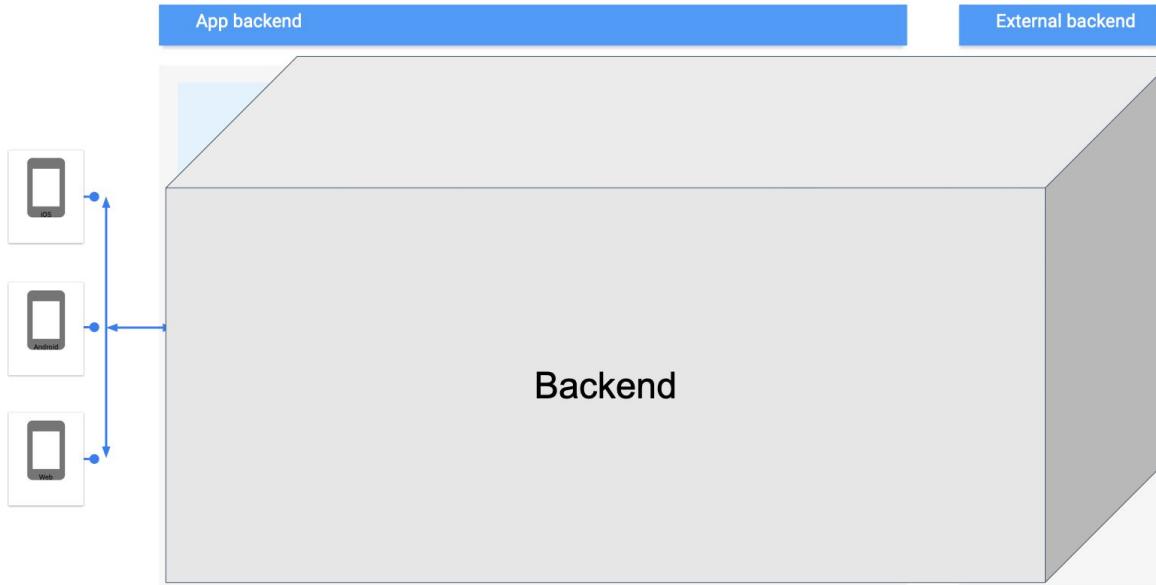


**OPEN METRICS**

An effort to create an open standard for transmitting metrics at scale, with support for both **text representation** and **Protocol buffers**.

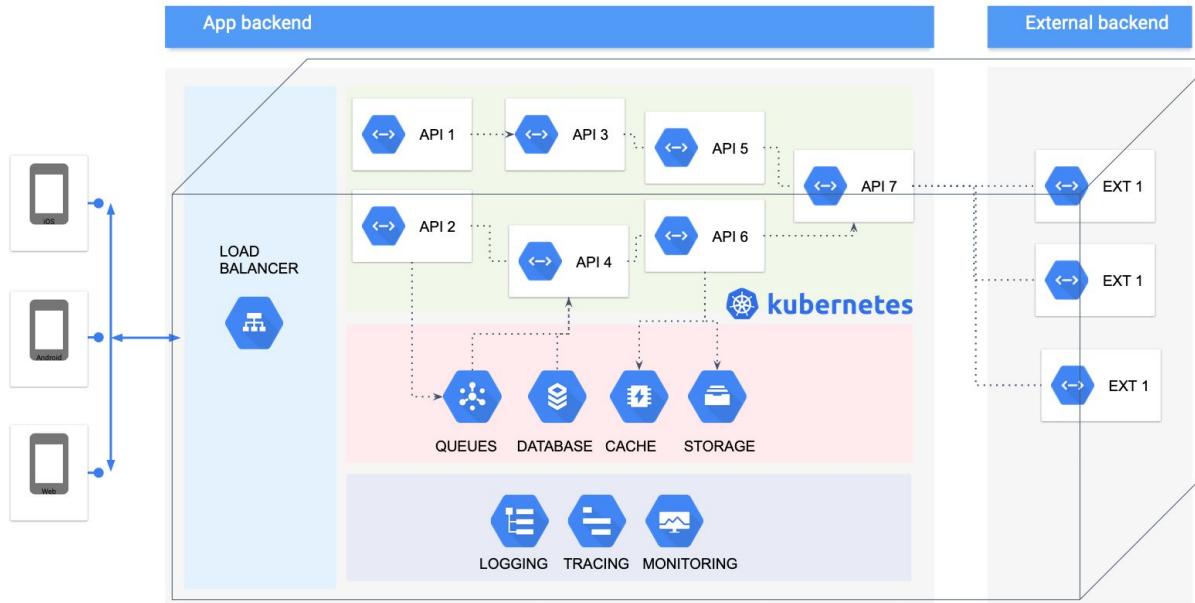
# 1.4 Cloud-native monitoring

## Microservices observability



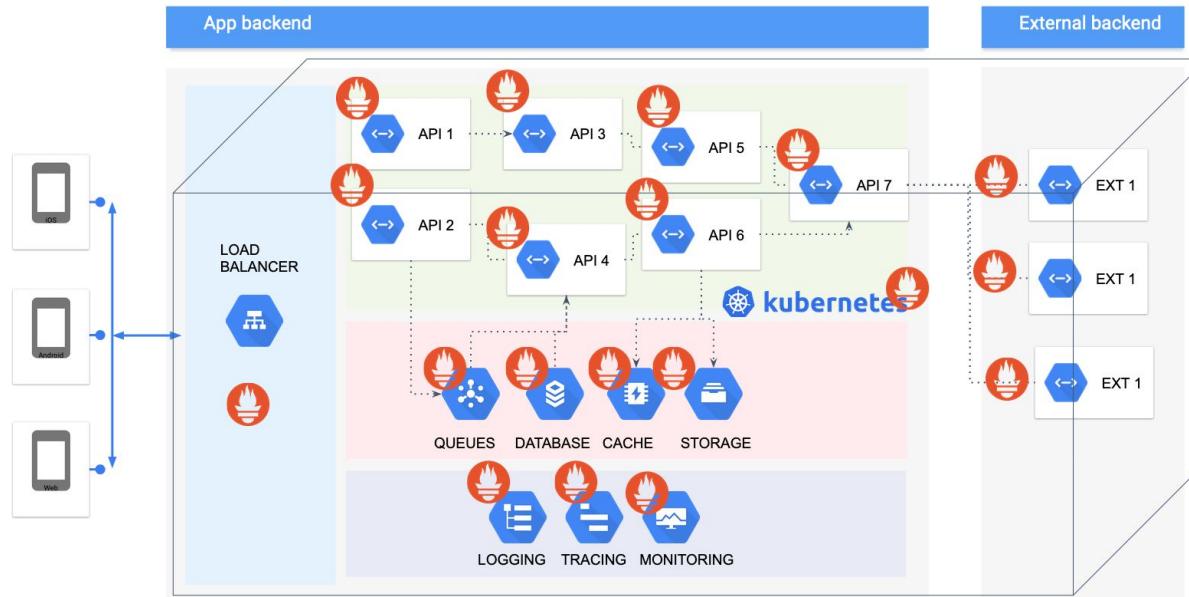
# 1.4 Cloud-native monitoring

## Microservices observability



# 1.4 Cloud-native monitoring

## Microservices observability

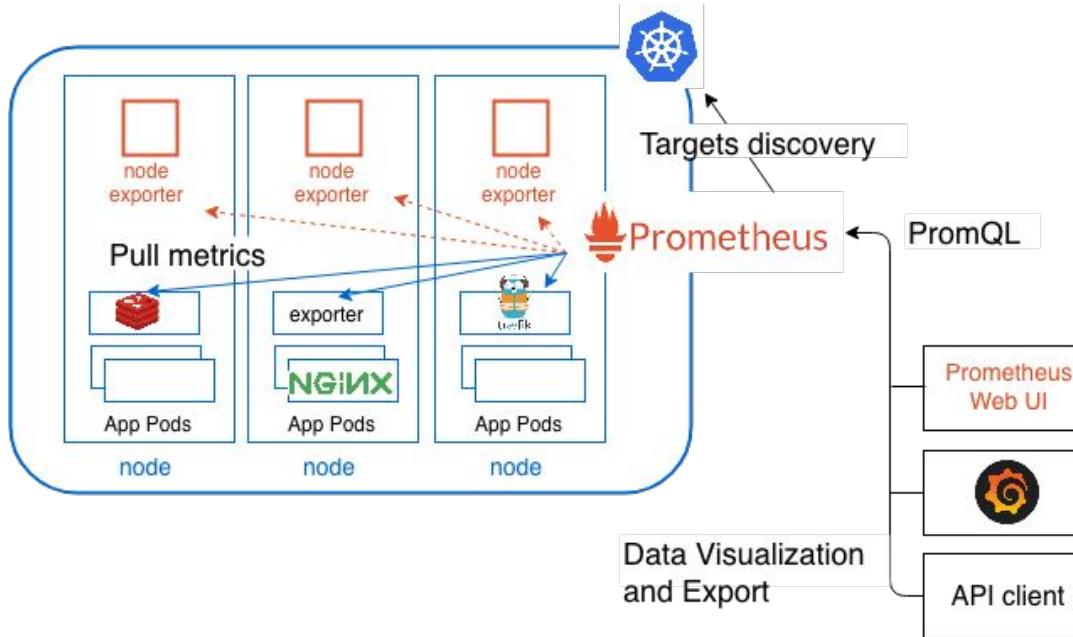


## 2. Monitoring systems and applications

Metrics scraping, querying, graphing, instrumenting prometheus-native apps

## 2. Monitoring systems and applications

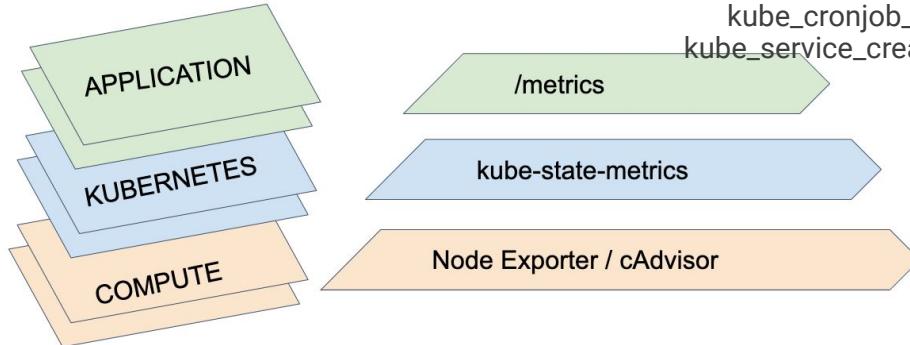
Metrics, scraping, querying, graphing, prometheus-native apps



# 2.1 Monitoring systems

## Available exporters

- node-exporter (Prometheus)
- kube-state-metrics (Kubernetes)
- cAdvisor (Google)
- Docker daemon (Docker)



node-exporter metrics

arp, boottime, cpu, diskstats, entropy, exec, filesystem, ipvs, loadavg, meminfo, netclass, netstat, nfs, sockstat, stat, **textfile**, time, timex, vmstat, zfs

cadvisor metrics

container\_accelerator\_memory\_used\_bytes, container\_cpu\_load\_average\_10s, container\_fs\_reads\_total, container\_last\_seen, container\_network\_receive\_errors\_total ...

kube-state-metrics metrics

.  
kube\_cronjob\_status\_active, kube\_namespace\_annotations, kube\_service\_labels, kube\_service\_created, kube\_replicaset\_status\_replicas, kube\_node\_status\_condition

docker daemon metrics

engine\_daemon\_container\_action\_seconds\_bucket, process\_cpu\_seconds\_total

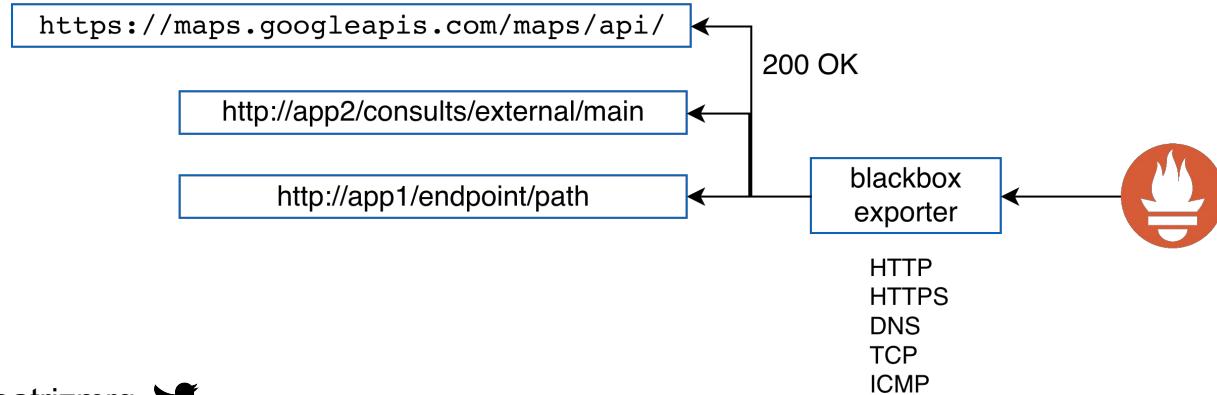
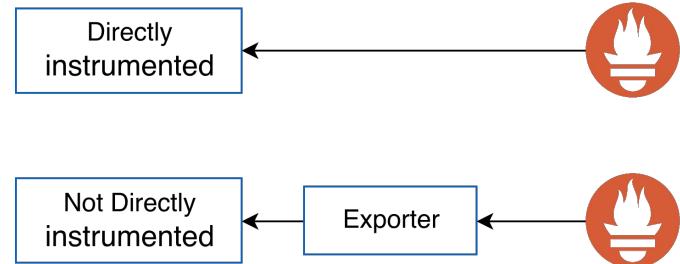
**commit**

MAD · NOV 22-23 · 2019

## 2.2 Monitoring applications

### Application metrics

- Blackbox exporter: monitoring microservices endpoints
- Official exporters
- prometheus-native applications



## 2.3 prometheus-native applications

### Instrumenting applications

Before you can **monitor your services**, you need to add instrumentation to their code.

1. Implement the Prometheus metric types
2. Choose a Prometheus **client library** that matches the language of your application.
3. **Define internal metrics**
4. **Expose** them via an HTTP endpoint on your application instance.

Available metrics types:

- Counter
- Gauge
- Histogram
- Summary

Available client libraries:

- Go
- Java or Scala
- Python
- Ruby
- Unofficial, third-party: bash, PHP, node.js, .....



## 2.3.1 prometheus-native applications

### Book store app

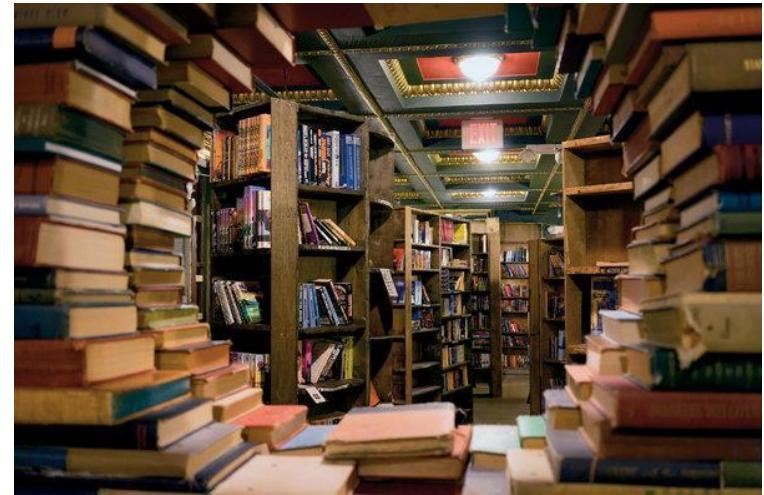
Possible actions taken by users:

- Buy a book

Every book categorized by:

- Genre: Terror, Romance, Science Fiction
- Pages

- I would like to know, **which genres** are buy the most by my clients.
- **Which editions** are preferred, pocket editions, portable..
- If a specific genre is most buy to read in a comfortable light edition.
- Number of **visits**
- **Time** spend before buying



## 2.3.1 prometheus-native applications

What metrics look like

For every metric

```
# HELP books_sold Number of books sold
# TYPE books_sold counter
books_sold{genre="terror"} 199.0
books_sold{genre="romance"} 70.0
# HELP uptime uptime
# TYPE uptime gauge
uptime 4.2769899E7
# HELP systemload_average systemload_average
# TYPE systemload_average gauge
systemload_average 0.55
# HELP heap_committed heap_committed
# TYPE heap_committed gauge
heap_committed 2234880.0
```

Description  
Metric type

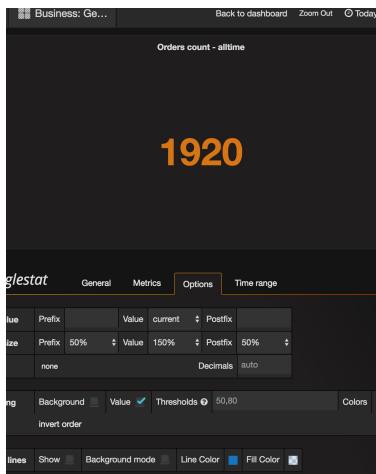
Values are  
float64

## 2.3.2 prometheus-native applications

Metrics types: Counter and Gauge

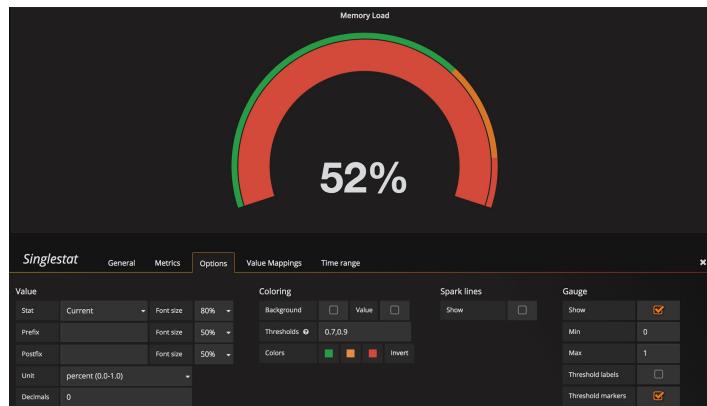
Counter:

- Increment or reset
- Number of visits or books sold



Gauge:

- Type of counter
- Increment or decrement
- Common use: represent memory or CPU



## 2.3.2 prometheus-native applications

Metrics types: Counter

```
# HELP books_sold Number of books sold
# TYPE books_sold counter
books_sold{genre="terror"} 2
books_sold{genre="romance"} 1
```

1



```
booksSold = promauto NewCounterVec(prometheus.CounterOpts{
    Name: "books_sold",
    Help: "The total number of books sold",
},
[]string{"genre"})
```

2

```
r.POST("/buy/:genre", func(c *gin.Context) {
    genre := c.Param("genre")
    booksSold.WithLabelValues(genre).Inc()
    c.JSON(200, gin.H{
        "message": "Thank you for purchasing a book",
    })
})
```

3

## 2.3.2 prometheus-native applications

Metrics types: Gauge

```
# HELP online_users Users that are online
# TYPE online_users gauge
online_users 3
```

1

```
r.GET("/", func(c *gin.Context) {
    onlineUsers Inc()
    c.JSON(200, gin.H{
        "message": "Look at all the books we have!",
    })
})
```

3

```
onlineUsers = promauto.NewGauge(prometheus.GaugeOpts{
    Name: "online_users",
    Help: "Users that are online",
})
```

2

```
r.POST("/buy/:genre", func(c *gin.Context) {
    onlineUsers Dec()
    c.JSON(200, gin.H{
        "message": "Thank you for purchasing a book",
    })
})
```

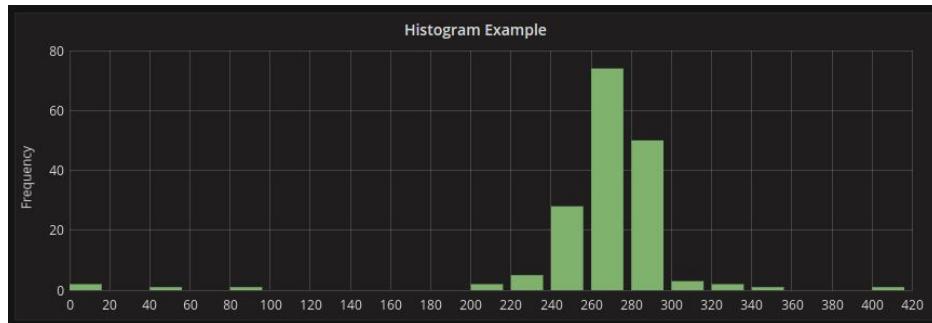
4

## 2.3.3 prometheus-native applications

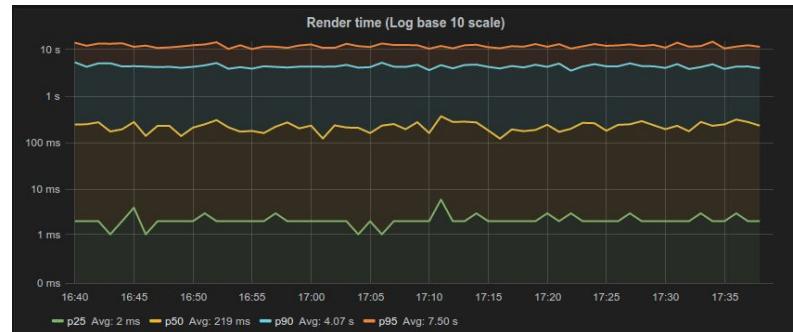
Metrics types: Histogram and Summary

Histogram:

- Values grouped in buckets
- **Cumulative**
- Number of books with number of pages ranged from 50 to 100 pages.



@beatrizmrg



Summary:

- **Average and Percentiles** information
- Get extremes behaviours
- Time spent in the application previous to buy anything

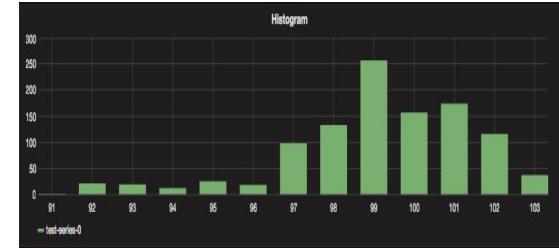
## 2.3.3 prometheus-native applications

### Metrics types: Histogram

```
# HELP books_sold_pages Books sold with number of pages
# TYPE books_sold_pages histogram
books_sold_pages_bucket{genre="terror",le="20"} 5
books_sold_pages_bucket{genre="terror",le="100"} 21
books_sold_pages_bucket{genre="terror",le="150"} 29
books_sold_pages_bucket{genre="terror",le="200"} 32
books_sold_pages_bucket{genre="terror",le="+Inf"} 32
books_sold_pages_sum{genre="terror"} 2624
books_sold_pages_count{genre="romance"} 32
```

```
bSoldPages = promauto.NewHistogramVec(
    prometheus.HistogramOpts{
        Name:      "books_sold_pages",
        Help:      "Books sold with number of pages",
        Buckets:  []float64{20.0, 100.0, 150.0, 200.0},
    },
    []string{"genre"})
```

1



```
r.POST("/buy/:genre", func(c *gin.Context) {
    genre := c.Param("genre")

    bSoldPages.WithLabelValues(genre).Observe(c.and.Float64
        () * 200)

    c.JSON(200, gin.H{
        "message": "Thank you for purchasing a book",
    })
})
```

3

## 2.3.3 prometheus-native applications

### Metrics types: Summary

```
# HELP time_spent Time spent to buy a book
# TYPE time_spent summary
time_spent{genre="romance",quantile="0.5"} 36.08
time_spent{genre="romance",quantile="0.9"} 68.68
time_spent{genre="romance",quantile="0.99"} 94.05
time_spent_sum{genre="romance"} 912.74
time_spent_count{genre="romance"} 22
```

```
timeSpent = promauto.NewSummaryVec(prometheus.SummaryOpts{
    Name:      "time_spent",
    Help:      "Time spent to buy a book",
    Objectives: map[float64]float64{0.5: 0.05, 0.9: 0.01, 0.99: 0.001},
    },
    []string{"genre"})
```

1

```
r.POST("/buy/:genre", func(c *gin.Context) {
    genre := c.Param("genre")

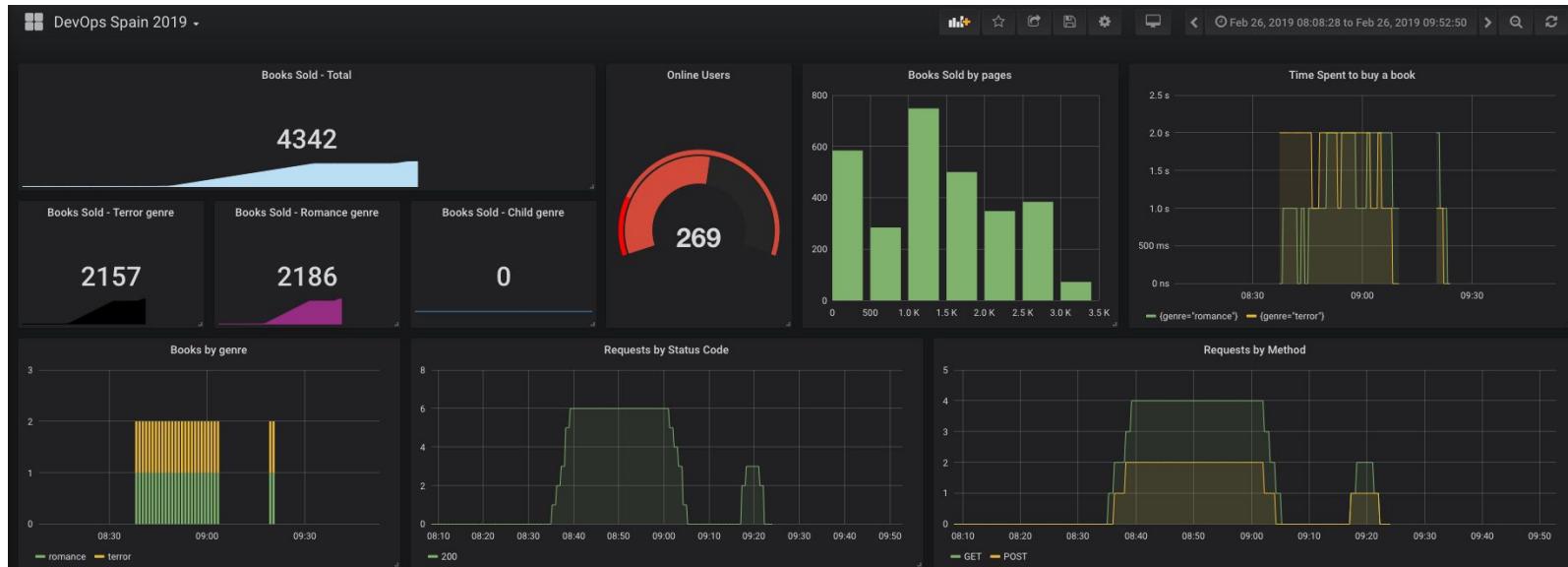
    timeSpent.WithLabelValues(genre).Observe(rand.Float64() *100
)
    c.JSON(200, gin.H{
        "message": "Thank you for purchasing a book",
    })
})
```

2

3

# 2.3 prometheus-native applications

## Book Store insights

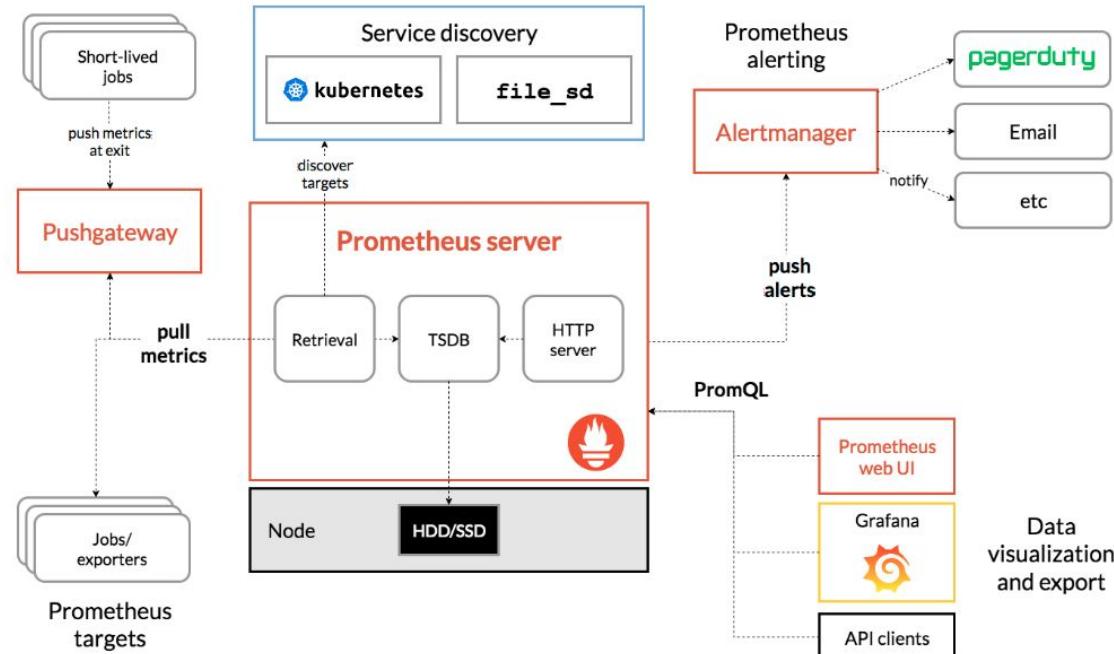


# 3. Alerting and Integrations

Decoupling, grouping, silencing, alerting

# 3. Alerting and Integrations

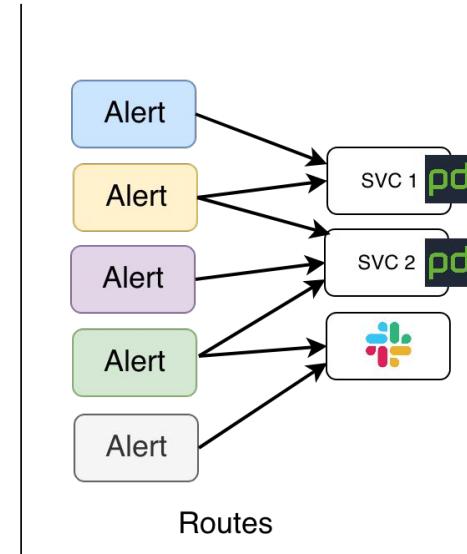
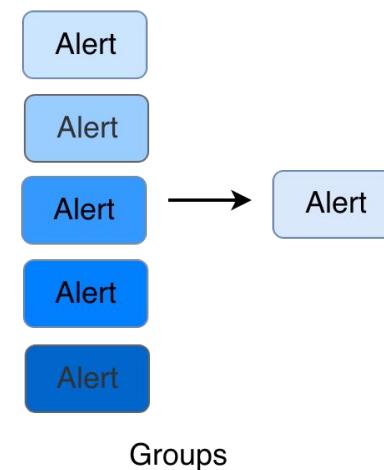
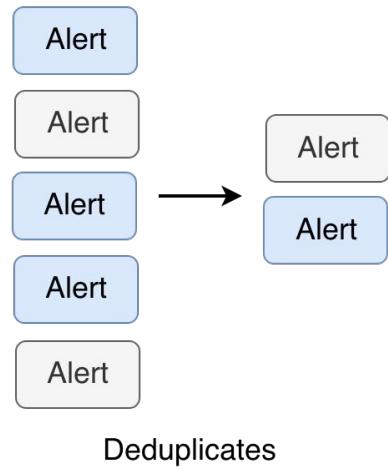
Grouping, silencing, alerting



# 3.1 Alertmanager

## Alerts notification management

- Deduplicates
- Groups
- Routes
- Inhibits
- Silence



# 4. Scaling and ecosystem

Federation, Thanos, k8s operator, Signifai

# 4. Scaling and Ecosystem

Federation, Thanos, K8s operator, Signifai

Getting the most out of Prometheus in real life case scenarios.

- New necessities are born: **Scaling** Prometheus
- A whole ecosystem appears to:
  - **Enable** an easy deployment, management and operation (Helm, Operator, ...)
  - **Complement** functionalities and support new approaches to cover specific necessities (Thanos, M3,Cortex ..)
  - Provide new paths to **exploit** the observability and instrumentations of our applications (Signifai)



MAD · NOV 22-23 · 2019

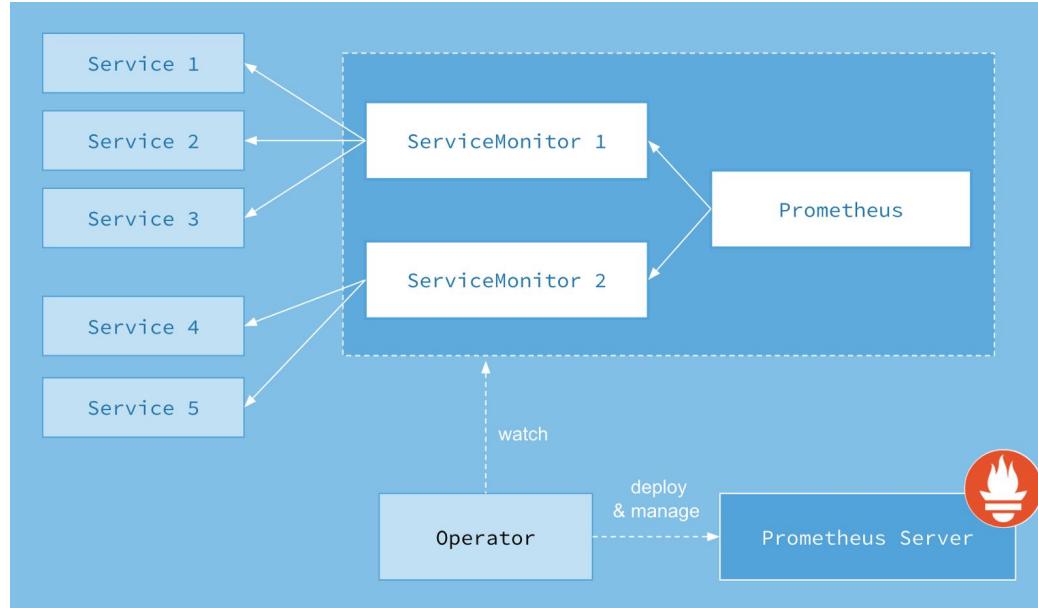
# 4.1 Deploying Prometheus

## K8s prometheus-operator



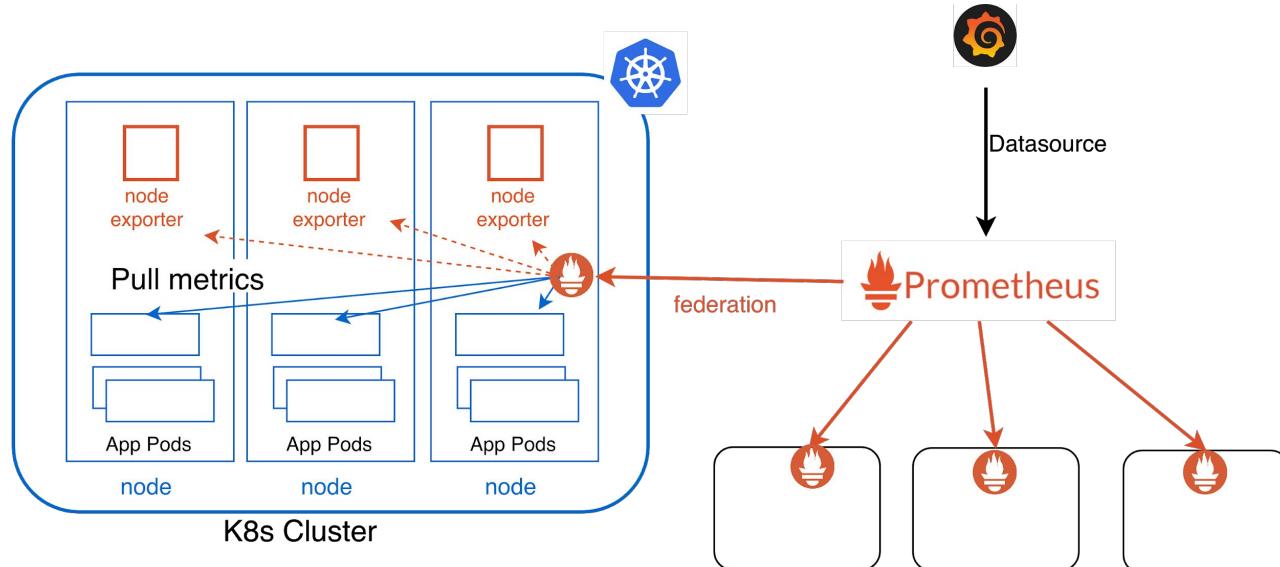
Custom resources:

- **Prometheus**
- ServiceMonitor
- Alertmanager
- PrometheusRule



## 4.2 Scaling Prometheus

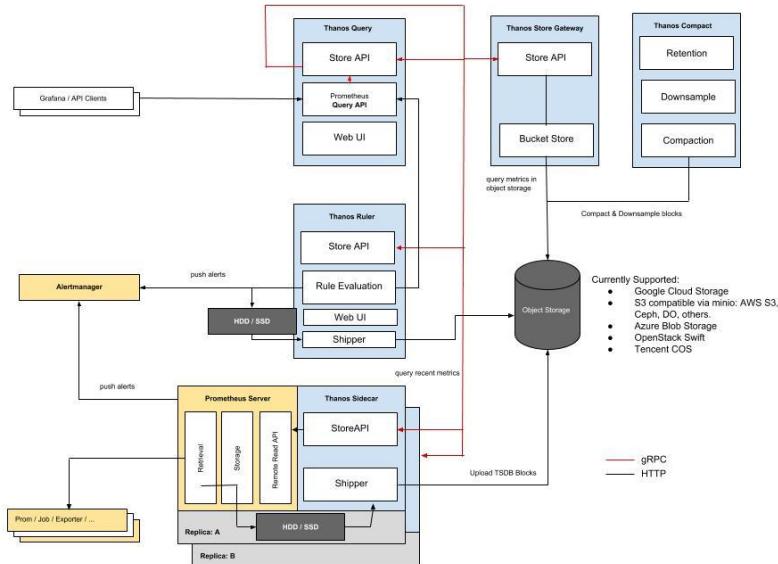
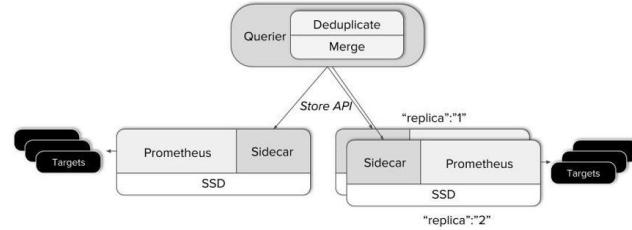
### Federation



# 4.3 Thanos

Highly available Prometheus setup with long term storage capabilities

- Global view
- **Distributed queries**
- Unlimited retention



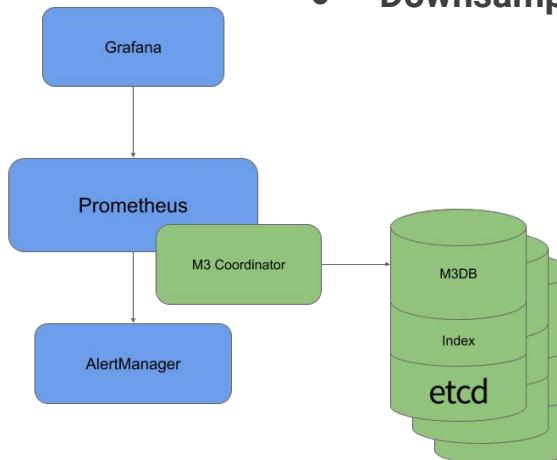
# 4.4 M3DB

Born to facilitate the growth of Uber's global operations



```
- name: Retain all disk_used metrics
  filter: name:disk_used* device:sd*
  policies:
    - resolution: 10s
      retention: 2d
    - resolution: 1m
      retention: 30d
    - resolution: 1h
      retention: 5y
```

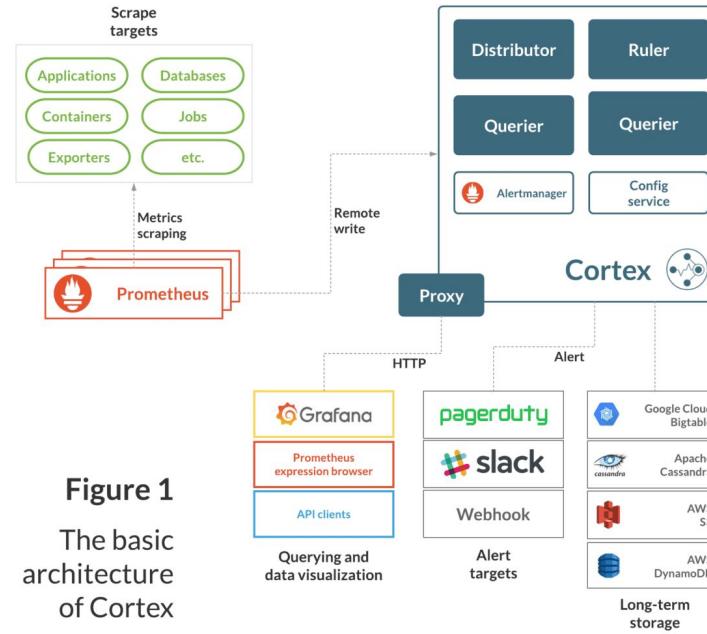
- Reliably houses large-scale metrics over long retention time windows
- **Downsampling of metrics**



# 4.5 Cortex

## Multitenant, as-a-Service

- Multi-tenant, horizontally scalable open source **Prometheus-as-a-Service**
- CNCF project since September 2018
- Provides a complete out-of-the-box solution for even most demanding monitoring and observability use cases.
- Hosted Cortex: Weave Cloud and Grafana Cloud



**Figure 1**  
The basic  
architecture  
of Cortex



# 4.6 Signifai

## AIOps with Prometheus

signifai 

- Acquired by NewRelic
- AI and machine learning **powered correlation engine** for DevOps and Site Reliability Engineering.
- End-to-end systems analysis
- Get answers not alerts
- Automatic correlation, aggregation and prioritization of alerts.
- Be proactive not reactive



# Conclusions:

- cloud-native monitoring: observability
- RED metrics, don't fly blind
- Cross team implication
- Instrumentation: not as painful after all

## **Cloud native monitoring with Prometheus**

