# 一小时玩转云原生系统监控

孟凡杰

腾讯云容器技术专家

# 目录

# 监控系统

## 为什么监控，监控什么内容？

- 对自己系统的运行状态了如指掌，有问题及时发现，而不让用户先发现我们系统不能使用。
- 我们也需要知道我们的服务运行情况。

## 监控目的

- 长期趋势分析：比如资源用量预测
- 对照分析：比如两个版本的系统运行资源使用情况的差异
- 告警：当系统出现或者即将出现故障时，监控系统需要迅速反应并通知管理员
- 故障分析与定位：通过对不同监控监控以及历史数据的分析，能够找到并解决根源问题。
- 数据可视化：通过可视化仪表盘能够直接获取系统的运行状态、资源使用情况、以及服务运行状态等直观的信息。

# 监控的分类

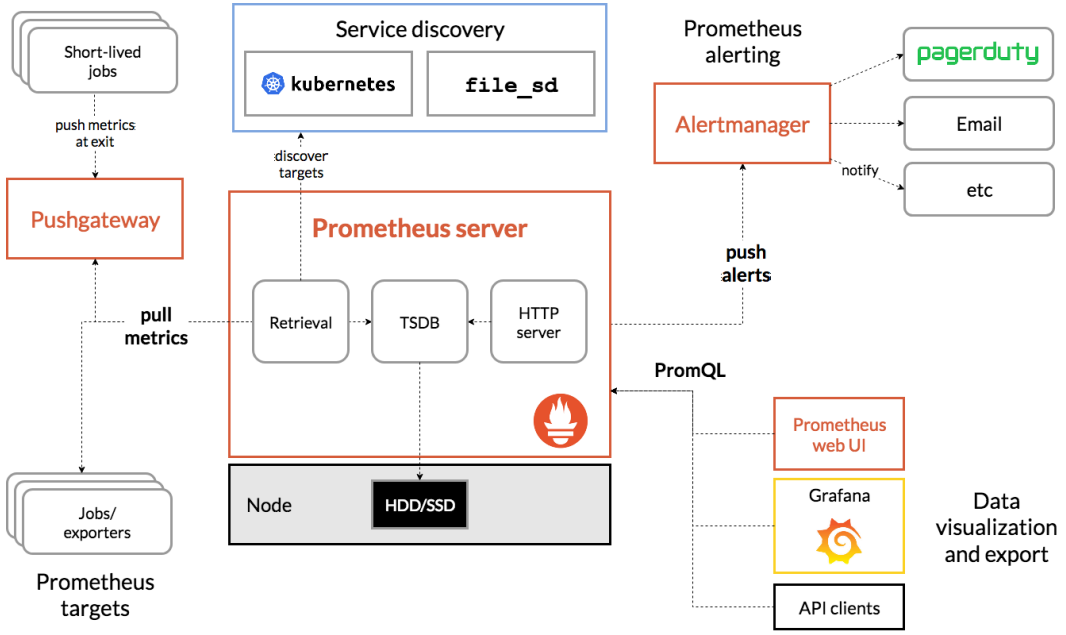
| | 黑盒监控 | 白盒监控 |
| --- | --- | --- |
| 实现机制 | • 从系统外部监控。<br>• 对应用运行逻辑一无所知。<br>• 基于 ping，http request 等检测手段，并等待检测结果。 | • 通过侵入性代码采集系统指标（如 cgroup）和自定义业务指标（如 httpcode，并发 request，用户行为信息等）。 |
| 应用场景 | • 从系统外部观测整体可用性。<br>• 无法获得系统内部的运行情况。<br>• 测试结果通常是布尔型：成功或失败，无法全方位监控。<br>• 可以在系统或者服务在发生故障时能够快速通知相关的人员进行处理。 | • 黑盒监控的补充。<br>• 通过白盒能够了解其内部的实际运行状态，通过对监控指标的观察能够预判可能出现的问题，从而对潜在的不确定因素进行优化。<br>• 监控指标丰富可扩展，可监控 CPU 利用率，也可以监控并发请求数量。 |

# Prometheus 架构

由前 Google 员工，受 Google 内部 Borgman 的启发，2012 年开始的开源项目，2018 年进入毕业状态。

Prometheus：先见之明。

- 以指标名称和键值对唯一标识的基于时间序列的多维数据模型
- 支持多维灵活查询的 PromQL
- 与存储系统解耦
- 基于 HTTP 协议的 Pull 模式进行时间序列指标采集
- 中间网关支持 Push 模式
- 基于静态配置和或服务发现的目标发现机制
- 灵活的图像化展示

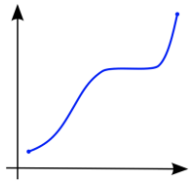


# Prometheus 数据模型

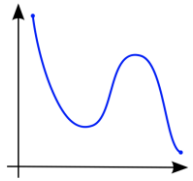
- 指标名与标签（Metrics Name 和 Labels）
  - 每个业务指标由指标名称和键值对标签唯一标识
- 采样点（Samples）
  - 每次指标收集到的采样数据由两部分组成
    - Timestamp
    - Value
- 指标表示方式
  - OpenTSDB的标示方式
    - <metric name>{<label name>=<label value>, ...}
  - 示例
    - api_http_requests_total{method="POST", handler="/messages"}

# Prometheus 中的指标类型

- Counter（计数器器）
  - Counter 类型代表一种样本数据单调递增的指标，即只增不减，除非监控系统发生了重置。

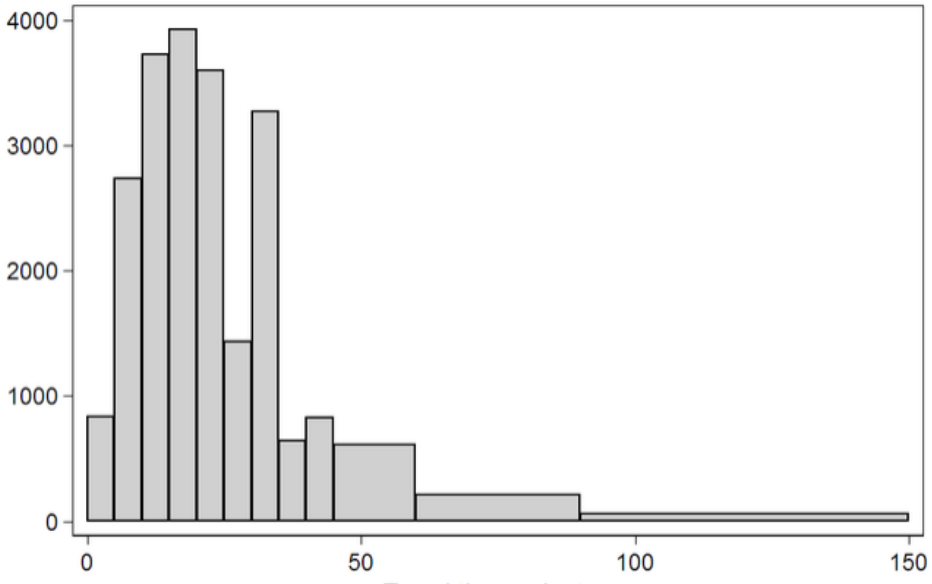


- Gauge（仪表盘）
  - Guage 类型代表一种样本数据可以任意变化的指标，即可增可减。



- Summary（摘要）
  - 与 Histogram 类型类似，用于表示一段时间内的数据采样结果（通常是请求持续时间或响应大小等），但它直接存储了分位数（通过客户端计算，然后展示出来），而不是通过区间来计算。
  - 它们都包含了 <basename>_sum 和 <basename>_count 指标。
  - Histogram 需要通过 <basename>_bucket 来计算分位数，而 Summary 则直接存储了分位数的值。

- Histogram（直方图）
  - Histogram 在一段时间范围内对数据进行采样（通常是请求持续时间或响应大小等），并将其计入可配置的存储桶（bucket）中，后续可通过指定区间筛选样本，也可以统计样本总数，最后一般将数据展示为直方图。
  - 样本的值分布在 bucket 中的数量，命名为 <basename>_bucket{le=“<上边界>"}。
  - 所有样本值的大小总和，命名为 <basename>_sum
  - 样本总数，命名为 <basename>_count。值和 <basename>_bucket{le=“+Inf"} 相同。



# 监控数据的采集

## Push or Pull

| | Push | Pull |
|---|---|---|
| 代表应用 | InfluxDB | Prometheus |
| 发起者 | 被监控方发起 | 监控系统发起（短时 Job 如何监控？） |
| 网络需求 | 目标地址固定，容易绕过防火墙 | 需要连接所有被监控方，通常需要与被监控房部署在一起 |
| 并发 | 由被监控系统上报数据，容易对监控系统造成较大并发压力，导致监控系统网络拥塞或者系统过载 | 可由监控系统轮训并顺序拉取，无并发问题 |
| 故障感知 | 若监控系统假死，被监控方无法感知，可能会继续推送数据导致雪崩 | 若系统超负荷，数据采集会变慢，雪崩几率较小 |
| 目标发现 | 被监控目标主动上报，无需进行目标发现 | 需要主动发现被监控目标 |

# 在 Kubernetes 集群中的监控系统



每个节点的 kubelet 会收集当前节点 host 上所有信息，包括 CPU、内存、磁盘等。Prometheus 会 pull 这些信息，给每个节点打上标签来区分不同的节点。

```
# HELP container_cpu_system_seconds_total Cumulative system cpu time consumed in seconds.
# TYPE container_cpu_system_seconds_total counter
container_cpu_system_seconds_total{id="/"} 735292
container_cpu_system_seconds_total{id="/system.slice"} 710067.82
container_cpu_system_seconds_total{id="/system.slice/atd.service"} 0.04
container_cpu_system_seconds_total{id="/system.slice/auditd.service"} 652.29
container_cpu_system_seconds_total{id="/system.slice/cloud-config.service"} 0
container_cpu_system_seconds_total{id="/system.slice/cloud-final.service"} 0
container_cpu_system_seconds_total{id="/system.slice/cloud-init-local.service"} 0
container_cpu_system_seconds_total{id="/system.slice/cloud-init.service"} 0
container_cpu_system_seconds_total{id="/system.slice/crond.service"} 4267.6
container_cpu_system_seconds_total{id="/system.slice/dbus.service"} 3.44
container_cpu_system_seconds_total{id="/system.slice/dm-event.service"} 428.39
container_cpu_system_seconds_total{id="/system.slice/docker.service"} 55096.24
container_cpu_system_seconds_total{id="/system.slice/dracut-shutdown.service"} 0
container_cpu_system_seconds_total{id="/system.slice/fedora-autorelabel-mark.service"} 0
container_cpu_system_seconds_total{id="/system.slice/fedora-import-state.service"} 0
container_cpu_system_seconds_total{id="/system.slice/fedora-readonly.service"} 0
container_cpu_system_seconds_total{id="/system.slice/gssproxy.service"} 27.07
container_cpu_system_seconds_total{id="/system.slice/iscsi-shutdown.service"} 0
```

# 在 Kubernetes 中汇报指标



应用 Pod 需要声明上报指标端口和地址

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    prometheus.io/port: http-metrics
    prometheus.io/scrape: "true"
name: loki-0
  namespace: default
spec:
 ports:
    - containerPort: 3100
      name: http-metrics
      protocol: TCP
```

应用启动时，需要注册 metrics

```
http.Handle("/metrics", promhttp.Handler())
http.ListenAndServe(sever.MetricsBindAddress, nil)
```

注册指标

```
func RegisterMetrics() {
   registerMetricOnce.Do(func() {
      prometheus.MustRegister(APIServerRequests)
      prometheus.MustRegister(WorkQueueSize)
   })
}
```

代码中输出指标

```
metrics.AddAPIServerRequest(controllerName,
constants.CoreAPIGroup, constants.SecretResource, constants.Get,
cn.Namespace)
```

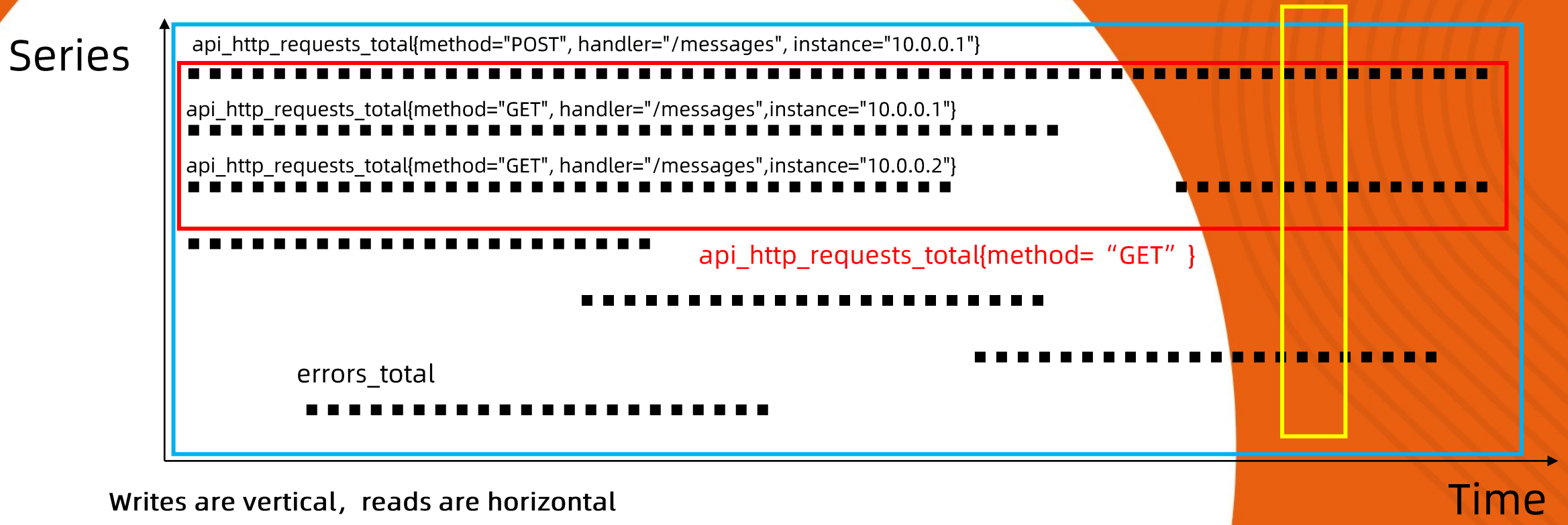
# 在 Kubernetes 集群中的监控系统



Kubernetes 的控制平面组件，包括各种 controller 都原生的暴露 Prometheus 格式的 metrics。

```
var (
	// TODO(a-robinson): Add unit tests for the handling of these metrics once
	// the upstream library supports it.
	requestCounter = prometheus.NewCounterVec(
		prometheus.CounterOpts{
			Name: "apiserver_request_count",
			Help: "Counter of apiserver requests broken out for each verb, API resource, client, and HTTP response contentType and code.",
		},
		[]string{"verb", "resource", "client", "contentType", "code"},
	)
	requestLatencies = prometheus.NewHistogramVec(
		prometheus.HistogramOpts{
			Name: "apiserver_request_latencies",
			Help: "Response latency distribution in microseconds for each verb, resource and client.",
			// Use buckets ranging from 125 ms to 8 seconds.
			Buckets: prometheus.ExponentialBuckets(125000, 2.0, 7),
		},
		[]string{"verb", "resource"},
	)
	requestLatenciesSummary = prometheus.NewSummaryVec(
		prometheus.SummaryOpts{
			Name: "apiserver_request_latencies_summary",
			Help: "Response latency summary in microseconds for each verb and resource.",
			// Make the sliding window of 1h.
			MaxAge: time.Hour,
		},
		[]string{"verb", "resource"},
	)
)
```

# 数据的存储（TSDB）

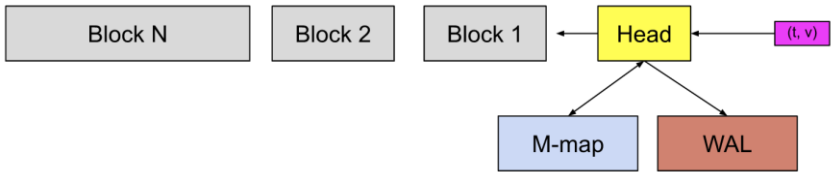


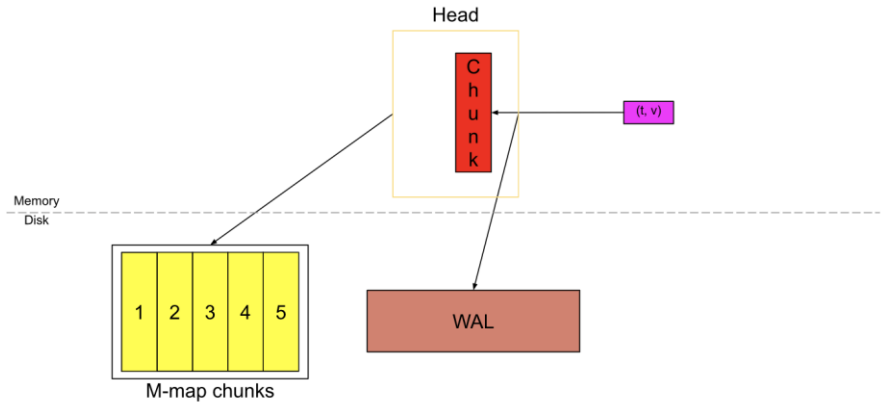

Writes are vertical，reads are horizontal

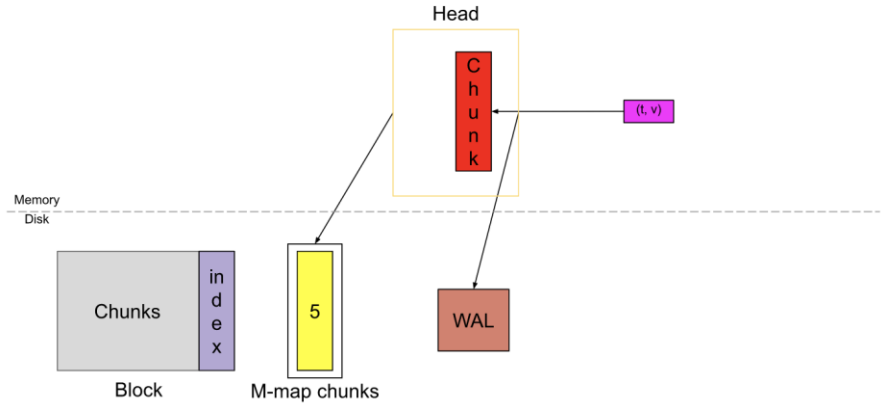
- 纵向写：每一次数据采集，会汇报周期内所有指标数据。
- 横向读：指标读取通常是按特定序列读取，单条时间序列 = Metrics Name + 唯一的 labels。

# 数据的存储机制

数据存储由内存部分和持久化部分组成




- 每次的采样数据
- 被保存至内存空间（mmap 的 chunks）
- 同时写入 WAL 日志
- 当一个 chunk 中保存了 120 个采样数据或时间跨度超过 2 小时
- 该 chunk 会标记为满，新 chunk 会被创建，老chunk 会被通过 mmap 映射至硬盘
- 内存中只需保存数据地址信息

- 当活跃 chunks 记录的数据超过一定阈值时（比如超出 3 小时），到达一个 checkpoint
- 旧 chunk 会被持久化至硬盘
- 持久化数据会被从 WAL 日志中剔除

# 数据索引

- 每一个 block 中，每个时间序列都有一个唯一 ID。
- 索引模块维护了一个键值 Label 到 ID 的映射关系。
- 通过 K 路轮训完成时间序列的高效查询。

```
{
        __name__="request_total",
        pod="mynginx-0",
        status=200,
        method="GET"
}
```

| 过滤标签 | 序列 ID | | | | | |
|---|---|---|---|---|---|---|
| status=200 | 1 | 2 | 3 | 88 | 143 | 1000 |
| method=GET | 3 | 7 | 99 | 138 | 188 | 888 |

# 数据的压缩

## Time 压缩

| 1640970061 | 1640970161 | 1640970261 | 1640970361 | 1640970461 | 1640970560 | 1640970661 | 1640970761 |
|---|---|---|---|---|---|---|---|
| 1640970061 | +100 | +100 | +100 | +100 | +99 | +101 | +100 |
| 1640970061 | +100 | +0 | +0 | +0 | -1 | +1 | +0 |

## Series 压缩

| 十进制值 | 十六进制表示 | 与前值的于或运算 |
|---|---|---|
| 12 | 0x4028000000000000 | 0x0010000000000000 |
| 24 | 0×4038000000000000 | 0x0016000000000000 |
| 15 | 0x402e000000000000 | 0x0006000000000000 |

# 存储规划

一个采样点

- 8 字节时间戳 +8 字节值，总计 16 字节
- 压缩后平均 1.37 字节/采样点，12 倍空间节省!

500 万个活跃时间序列

- 30 秒采样间隔
- 1 个月历史保留

合计需要 166000 个采样数据/秒

总计需要存储 432Billion 采样数据

- 每个采样点 8 字节时间戳 +8 字节值，总计需要 8T 存储
- 应用压缩算法后只需 0.8TB 存储空间

# 数据的查询与消费 PromQL

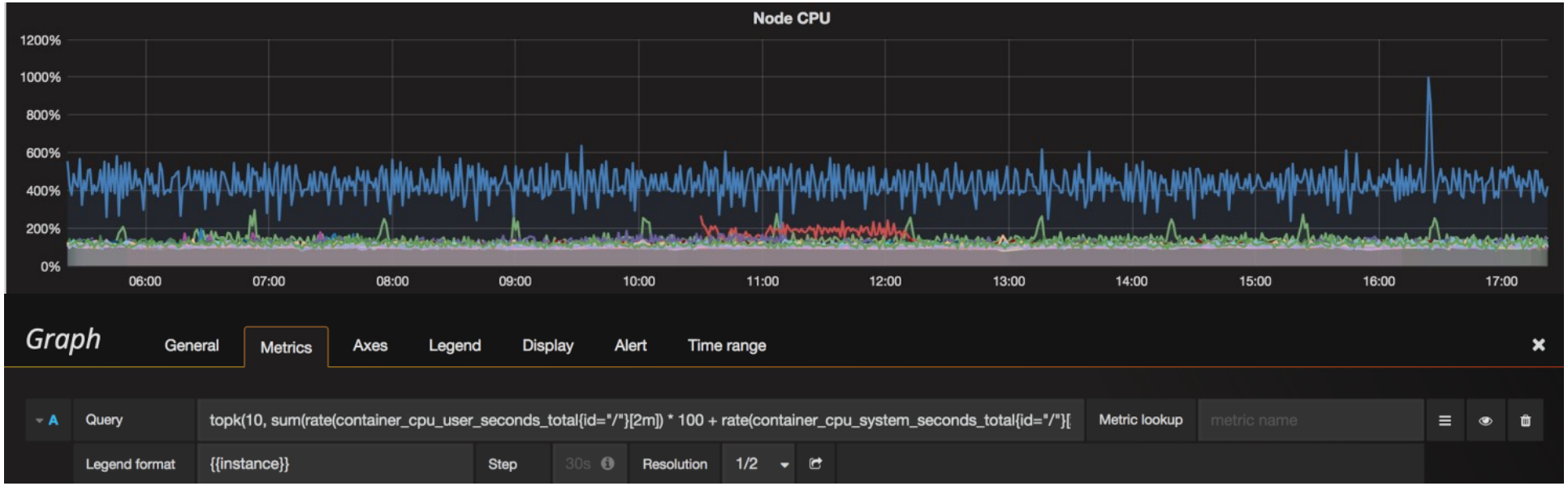


- histogram_quantile(0.95, sum(rate(httpserver_execution_latency_seconds_bucket[5m])) by (le))
- Histogram 是直方图， httpserver_execution_latency_seconds_bucke 是直方图指标，是将 httpserver 处理请求的时间放入不同的桶内，其表达的是落在不同时长区间的响应次数。
- by (le)，是将采集的数据按桶的上边界分组。
- rate(httpserver_execution_latency_seconds_bucket[5m])，计算的是五分钟内的变化率。
- Sum()，是将所有指标的变化率总计。
- 0.95，是取 95 分位。

综上：上述表达式计算的是 httpserver 处理请求时，95% 的请求在五分钟内，在不同响应时间区间的处理的数量的变化情况。

# 在 Kubernetes 集群中的监控系统

## Grafana Dashboard



# 开启告警

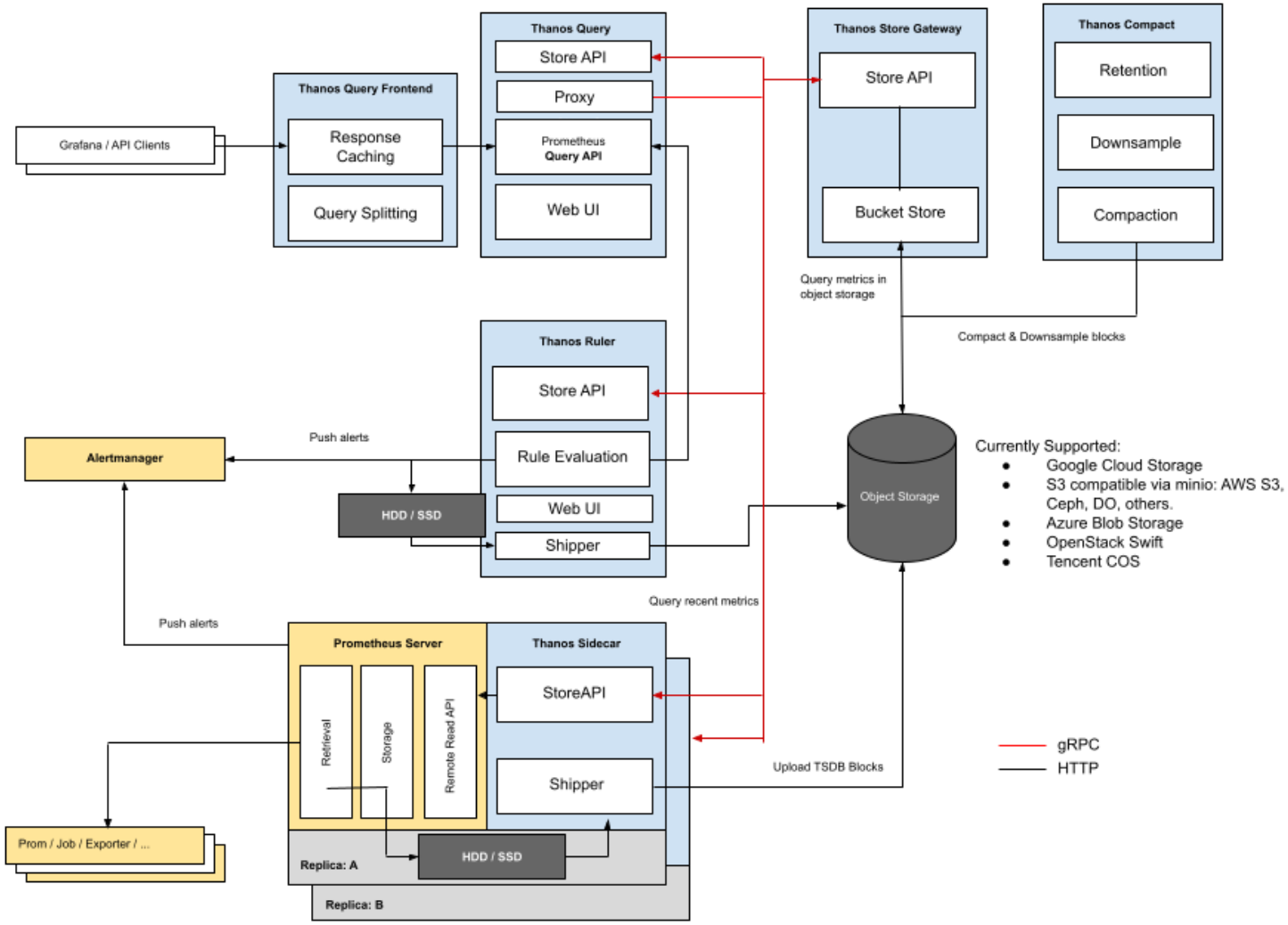
修改 Prometheus 配置文件 prometheus.yml，添加以下配置：

```
rule_files:
  - /etc/prometheus/rules/*.rules
```

在目录 /etc/prometheus/rules/ 下创建告警文件 hoststats-alert.rules 内容如下：

```
groups:
- name: hostStatsAlert
  rules:
  - alert: hostCpuUsageAlert
    expr: sum(avg without (cpu)(irate(node_cpu{mode!='idle'}[5m]))) by (instance) > 0.85
    for: 1m
    labels:
      severity: High
    annotations:
      summary: "Instance {{ $labels.instance }} CPU usgae high"
      description: "{{ $labels.instance }} CPU usage above 85% (current value: {{ $value }})"
```

# 以 Thanos 应对规模化挑战



极客时间 | 训练营

孟凡杰
腾讯云容器技术专家
前 eBay 资深架构师

极客时间 | 训练营

# 云原生训练营

向技术要红利，4 个月，挑战 50 万年薪

- 选择比努力更重要，云原生是新赛道
- 一线大厂都在加急招聘云原生工程师
- 懂 Kubernetes 的工程师可以弯道超车
- 简历直推面试官，学会薪资至少涨 50%

**15** 周 系统集训

**15** 大 内容模块

**6** 大 项目实战

**105** 天 助教答疑

**2** 次 企业内推

优惠价 ¥6499 ~~官网价 ¥6999~~

开课 7 天内可退款　支持花呗分期

添加学习助理咨询