

Prometheus 大型企业级监控

<https://prometheus.io>

内容简介

1. 开源监控程序介绍
2. Prometheus简介
3. Prometheus监控流程
4. Prometheus数据存储
5. Prometheus部署与告警
6. Prometheus Metrics类型
7. Prometheus收集数据方式
8. Prometheus 规则文件
9. Prometheus 标签替换与 ServiceMonitor
10. Prometheus 告警计算规则
11. Alertmanager 配置与模板
12. Alertmanager 优化—分组、抑制、静默
13. Alertmanager 告警对接方式
14. thanos 集群方案
15. Prometheus大规模集群解决方案
16. Prometheus自动分片

监控程序

Zabbix

Nagios

Cacti

Prometheus

Open-falcon

Nightingale

ZABBIX

Nagios®



Prometheus



falcon

监控对比

名称	存储	维护成本	管理	社区版本迭代
夜莺	RRD	成本高	界面丰富	低
zabbix	MySQL	成本低	界面丰富	高
Nagios	MySQL	成本低	界面丰富	高
Openfalcon	RRD	成本高	自研管理界面	低
Prometheus	TSDB	集成度高	自研管理界面	高

监控内容-立体化

服务端监控

开源组件/公有云

1. 基础服务

设备资源(cpu/内存/磁盘)
网络设备监控与网卡监控
TCP连接监控
硬件故障监控
ping 监控

开源组件

2. 业务与中间件

进程端口
服务SLA
接口耗时
资源消耗(cpu/memory)

用户质量监控

自研监控/第三方

3. CDN与流媒体

直播推流与拉流
点播拉流
CDN质量

第三方拨测/apm

4. 用户端监控

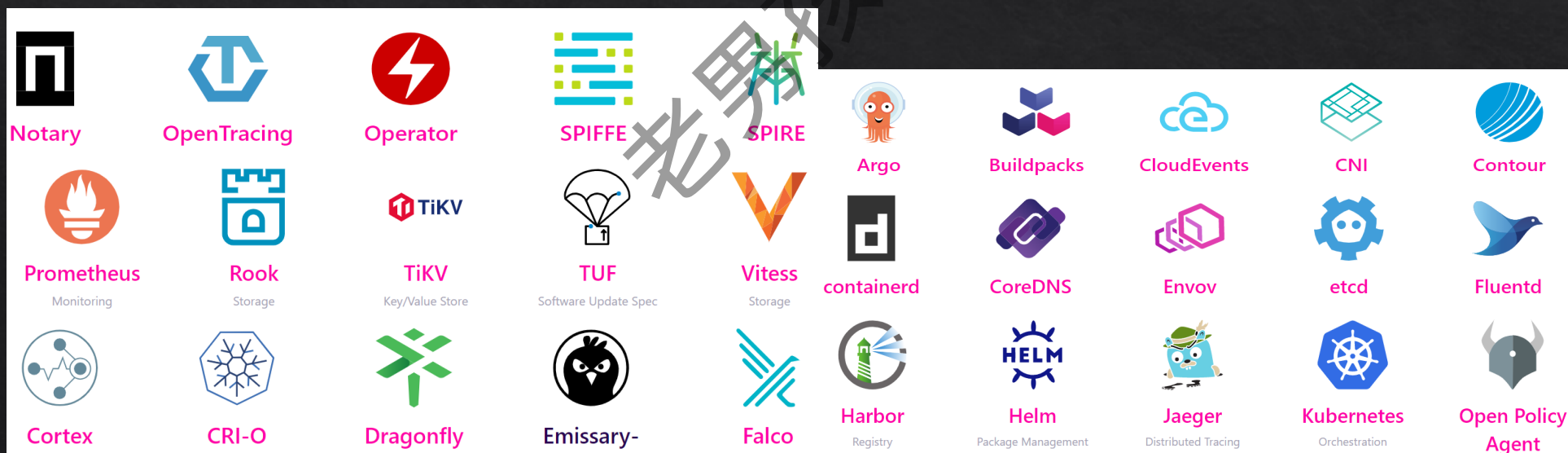
网络质量
DNS劫持
奔溃与卡顿
响应时间
错误率
慢请求

普罗米修斯简介

Prometheus 是由前 Google 工程师从 2012 年开始在 Soundcloud 以开源软件的形式进行研发的系统监控和告警工具。之后许多公司和组织都使用了 Prometheus 作为监控告警工具。

Prometheus 的开发者和用户社区非常活跃,现在是一个独立的开源项目。

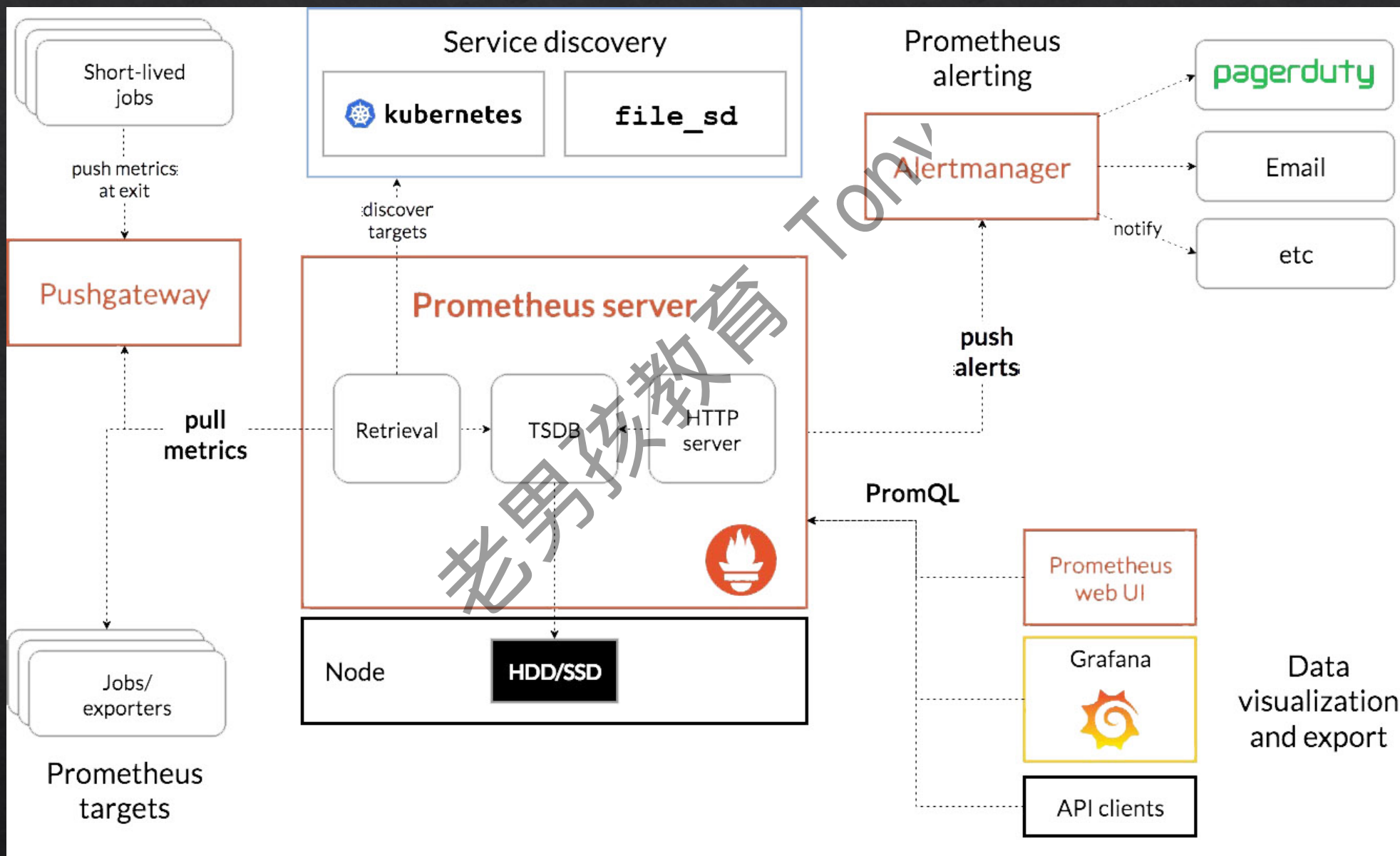
Prometheus 于 2016 年 5 月加入 CNCF (云原生)基金会。



Prometheus特点

1. 部署简单: 部署使用的是go编译的二进制文件, 不存在任何第三方依赖。
2. 强大的查询语言 PromQL: 通过PromQL可以实现对监控数据的查询、聚合。
3. 可扩展性强: 可以单机部署, 也可以使用跨机房集群。
4. 易于集成: 目前官方提供多种语言的客户端sdk, 客户端集成非常方便。
5. 可视化: 可以对接Grafana可视化工具展示监控指标。

Prometheus 架构



Prometheus数据存储方式

时序数据库（Time Series Database，简称 TSDB）是一种高性能、低成本、提供高读写、高压缩比存储、时序数据插值及聚合计算等服务。

TSDB 具备秒级写入百万级时序数据的性能，提供高压缩比低成本存储、预降采样、多维聚合计算、可视化查询结果等功能。解决由设备采集点数量巨大、数据采集频率高造成的存储成本高、写入和查询分析效率低的问题。

TSDB 是一个分布式时序数据库，具备多副本高可用能力。同时在高负载大规模数据量的情况下可以方便地进行弹性扩容，方便用户结合业务流量特点进行动态规划与调整。

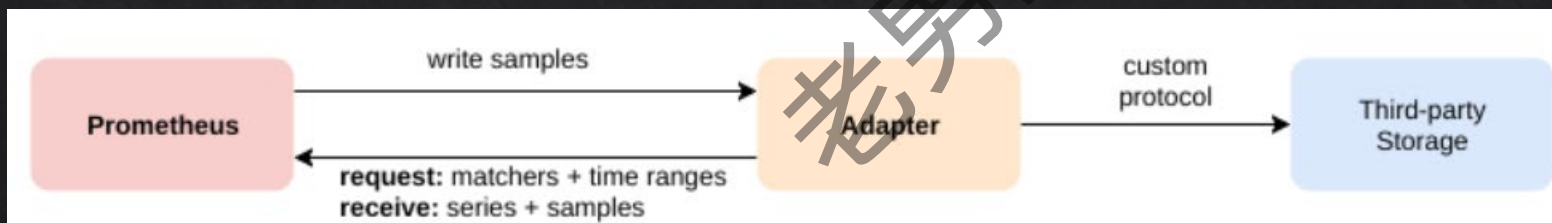
Prometheus数据存储支持类型

1. Local storage (本地存储)

Prometheus's local time series database stores data in a custom, highly efficient format on local storage.

(Prometheus的本地时间序列数据库在本地存储上以一种定制的、高效的格式存储数据。)

2. Remote storage integrations (远程存储)



Prometheus 远程存储支持类型

存储服务

AppOptics

Chronix

Cortex

CrateDB

Gnocchi

Graphite

InfluxDB

OpenTSDB

PostgreSQL/TimescaleDB

SignalFx

M3DB

支持模式

write

write

read/write

read/write

write

write

read/write

write

read/write

write

read/write

如果使用Prometheus, 监控步骤?

1. 安装Prometheus服务端

需要考虑的问题? 大规模的监控数据收集

2. 安装报警服务端 Alertmanager

3. 客户端暴露Metrics

4. 服务端收集数据与编写报警规则

5. 对接报警

Prometheus部署方式

1. 安装Prometheus

1.1 二进制安装: `prometheus --config.file=prometheus.yml`

1.2 容器安装: `docker-compose.yml`

1.3 Kubernetes安装: `kube-Prometheus` 或者 `prometheus operator`

1.4 安装报警服务: `alertmanager -c alertmanager.yml`

Prometheus配置文件

全局配置global:

默认抓取周期, 可用单位ms、smhdwy #设置每15s采集数据一次, 默认1分钟

[**scrape_interval**: <duration> | default = 1m]

默认抓取超时

[**scrape_timeout**: <duration> | default = 10s]

估算规则的默认周期 # 每15秒计算一次规则。默认1分钟

[**evaluation_interval**: <duration> | default = 1m]

和外部系统 (例如AlertManager) 通信时为时间序列或者警报 (Alert) 强制添加的
标签列表

external_labels:

[<labelname>: <labelvalue> ...]

规则文件列表

rule_files:

[- <filepath_glob> ...]

抓取配置列表

scrape_configs:

[- <scrape_config> ...]

Alertmanager相关配置

alerting:

alert_relabel_configs:

[- <relabel_config> ...]

alertmanagers:

[- <alertmanager_config> ...]

远程读写特性相关的配置

remote_write:

[- <remote_write> ...]

remote_read:

[- <remote_read> ...]

如果使用Prometheus监控, 需要做什么?

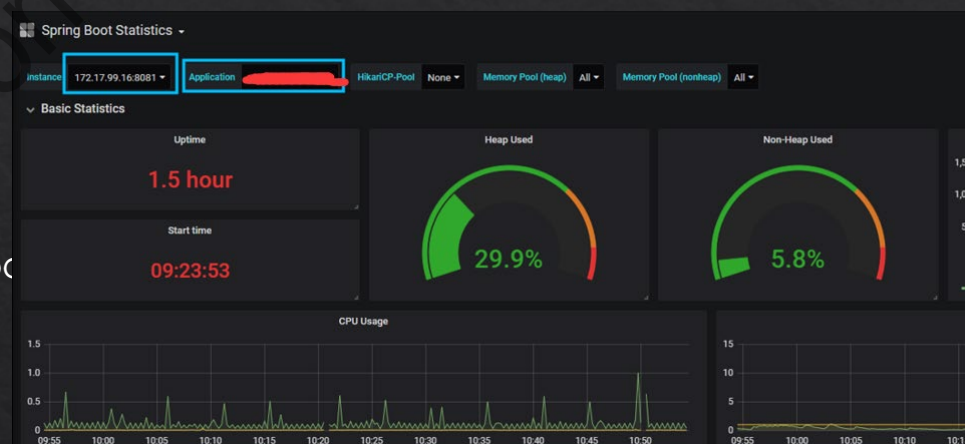
客户端需要暴露监控指标metrics

暴露方式:

App 代码集成

SpringCloud

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-
actuator</artifactId>
</dependency>
<!-- Micrometer core dependency -->
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-core</artifactId>
</dependency>
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-
prometheus</artifactId>
```



另外一种暴露方式metrics

自写 监控客户端启动

RedisexporterProject -config=./redis.cfg

```
promhttp_metric_handler_requests_total{code="503"}
```

```
# HELP redis_connected_clients Redis客户端连接数
```

```
# TYPE redis_connected_clients gauge
```

```
redis_connected_clients 3
```

```
# HELP redis_total_system_mem Redis系统总内存
```

```
# TYPE redis_total_system_mem gauge
```

```
redis_total_system_mem 3.069517824e+09
```

```
# HELP redis_used_memory Redis 使用内存总量
```

```
# TYPE redis_used_memory gauge
```

```
redis_used_memory 6.037504e+06
```

```
# HELP summary_http_request_duration_seconds summary
```

```
# TYPE summary http request duration seconds summary
```

第三方开源暴露方式metrics

使用开源exporter ssl-exporter

```
kubectl apply -f ssl-exporter-deployment.yaml
```

老男孩教育

Prometheus metrics 分类

•Histograms: 直方图

- 度量流数据中value的分布情况,
- Histogram可以计算最大/小值、平均值, 方差, 分位数

- 流量最大值
- 流量最小值
- 流量平均值
- 流量中位值

•Timers: 计时器

- 请求时延
- 磁盘读写时延

•Meters: TPS计数器

- 一种只能自增的计数器

- 平均每秒请求数
- 最近1分钟平均每秒请求数
- 最近15分钟平均每秒请求数

•Counters: 计数器

- 累计型的度量指标, 数值只能单调递增

- 服务请求数
- 任务完成数
- 错误出现次数

•Gauges: 计量器

既可以增加, 又可以减少的度量指标值

- 温度
- 内存使用量

Prometheus metrics 属性值

Gauges:

属性 value (某个属性实时值变化)

Counter:

属性 count (定时时间间隔之内的请求总数)

Meters:

属性 count (定时时间间隔之内的请求总数)

属性 meanRate (平均每秒请求数 (时间间隔内总请求数 / 定时任务时间间隔))

属性 oneMinuteRate (最近1分钟平均每秒请求数)

属性 FiveMinuteRate (最近5分钟平均每秒请求数)

属性 FifteenMinuteRate (最近15分钟平均每秒请求数)

Timer:

属性 count (定时时间间隔之内的请求总数)

属性 meanRate (平均每秒请求数 (时间间隔内总请求数 / 定时任务时间间隔))

属性 oneMinuteRate (最近1分钟平均每秒请求数)

属性 FiveMinuteRate (最近5分钟平均每秒请求数)

属性 FifteenMinuteRate (最近15分钟平均每秒请求数)

属性 min (定时时间间隔统计数据的最小值)

属性 max (定时时间间隔统计数据的最大值)

属性 mean (定时时间间隔内统计数据的平均值)

属性 stddev (定时时间间隔内统计数据的方差)

属性 median (定时时间间隔内统计数据的中位数)

属性 rate_unit (比率记录单位)

属性 duration_unit (时间数值单位)

Histogram:

属性 count (定时时间间隔之内的请求总数)

属性 min (定时时间间隔内请求参数的最小值)

属性 max (定时时间间隔内请求参数的最大值)

属性 mean (定时时间间隔内请求参数的平均值)

属性 stddev (定时时间间隔内请求参数的方差)

属性 median (定时时间间隔内请求参数的中位数)

属性 rate_unit (比率记录单位)

属性 75% <= (75百分位)

属性 95% <= (95百分位)

属性 98% <= (98百分位)

属性 99% <= (99百分位)

参考: https://www.jianshu.com/p/5d3ea3be3122?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation

Prometheus 收集数据方式

1. 编写采集任务

scrape_configs:

The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.

– job_name: 'prometheus' # metrics_path defaults to '/metrics' # scheme defaults to 'http'.

static_configs:

– targets: ['127.0.0.1:9090']

2. serviceMonitor (operator)

apiVersion: monitoring.coreos.com/v1

kind: ServiceMonitor

metadata:

labels:

k8s-app: alertmanager

Prometheus 收集数据方式

3. 自动发现

<scrape_config>	<nerve_sd_config>	<gce_sd_config>	<relabel_config>
<tls_config>	<serverset_sd_config>	<hetzner_sd_config>	<metric_relabel_configs>
oauth2	<triton_sd_config>	<kubernetes_sd_config>	<alert_relabel_configs>
<azure_sd_config>	<eureka_sd_config>	<lightsail_sd_config>	<alertmanager_config>
<consul_sd_config>	<scaleway_sd_config>	<marathon_sd_config>	<remote_write>
<digitalocean_sd_config>	<static_config>	<openstack_sd_config>	<remote_read>
<docker_sd_config>	<file_sd_config>	<ec2_sd_config>	<dns_sd_config>
<dockerswarm_sd_config>			

4. pushgateway (中转)

- job_name: 'pushgateway'
- static_configs:
 - targets: ['192.168.91.21:9091']
- labels:
 - instance: pushgateway

Prometheus- serviceMonitor

第一种方式, 使用ServiceMonitor 直接关联SVC

apiVersion: monitoring.coreos.com/v1

kind: ServiceMonitor

metadata:

labels:

k8s-app: prometheus

name: prometheus

namespace: monitoring

spec:

endpoints:

- interval: 30s

port: web

selector:

matchLabels:


prometheus: k8s

直接选择pod 标签: Prometheus = k8s


Prometheus- serviceMonitor

第二种方式. 使用ServiceMonitor -> SVC -> ENDPOINT

```
apiVersion: v1
kind: Endpoints
metadata:
  name: kube-etcd
  namespace: kube-system
  labels:
    k8s-app: kube-etcd
subsets:
- addresses:
  - ip: 192.168.91.143
  ports:
  - name: http-metrics
    port: 2390
    protocol: TCP
```



```
apiVersion: v1
kind: Service
metadata:
  name: kube-etcd
  namespace: kube-system
  labels:
    k8s-app: kube-etcd
spec:
  type: ClusterIP
  clusterIP: None
  ports:
  - name: http-metrics
    port: 2390
    protocol: TCP
    targetPort: 2390
```



```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: kube-etcd
  namespace: monitoring
  labels:
    k8s-app: kube-etcd
spec:
  jobLabel: k8s-app
  endpoints:
  - port: port
    interval: 30s
    port: http-metrics
  selector:
    matchLabels:
      k8s-app: kube-etcd
  namespaceSelector:
    matchNames:
    - kube-system
```


Prometheus-relabel_configs(监控数据标签替换)

```
cat prometheus-additional.yaml
```

```
- job_name: ssl-exporter
  metrics_path: /probe
  static_configs:
    - targets:
      - kubernetes.default.svc:443
      - repo.hostscc.com:443
      - www.baidu.com:443
  relabel_configs:
    - source_labels: [__address__]
      target_label: __param_target
    - source_labels: [__param_target]
      target_label: instance
    - target_label: __address__
      replacement: ssl-exporter.monitoring:9219
```

source_labels: 源标签, 没有经过relabel处理之前的标签名;

target_label: 目标标签, 通过action动作处理之后的新的标签名;

regex: 正则表达式, 用于匹配源标签值使用的;

replacement: replacement指定的替换后的标签(target_label)对应的数值;

scheme: https、http等代表获取指标数据时使用的协议类型

Prometheus 监控—relabel_configs(动作)

action 动作 含义

replace 根据regex来去匹配source_labels标签上的值，并将匹配到的值写入target_label中，其实就是替换；

keep 只是收集匹配到regex的源标签source_labels，而会丢弃没有匹配到的所有标签，用于选择保留哪些标签；

drop 丢弃匹配到regex的源标签，而会收集没有匹配到的所有标签，用于排除，与keep相反；

labeldrop 使用regex表达式匹配标签，符合规则的标签将从target实例中移除；

labelkeep 使用regex表达式匹配标签，仅收集符合规则的target，不符合匹配规则的不收集；

labelmap 根据regex的定义去匹配Target实例所有标签的名称，并且以匹配到的内容为新的标签名称，其值作为新标签的值；

```
- job_name: 'kubernetes-node'
  kubernetes_sd_configs:
  - role: node
  relabel_configs:
  - source_labels: [__address__]
    regex: '(.*)[:10250]'
    replacement: '${1}:9100'
    target_label: __address__
    action: replace
  - action: labelmap
    regex: __meta_kubernetes_node_label_(.+)
```


Prometheus 规则文件-Alert

groups:

- name: example

rules:

对于任何无法访问> 5分钟的实例的警报。

- alert: InstanceDown

expr: up == 0

for: 5m

labels:

severity: page

annotations:

summary: "Instance {{ \$labels.instance }} down"

description: "{{ \$labels.instance }} of job {{ \$labels.job }} has been down for more than 5 minutes."

Prometheus 规则文件-预聚合 (Recording Rule)

预聚合 (Recording Rule) 可以让我们对一些常用的指标或者计算相对复杂的指标进行提前计算，然后将这些数据存储到新的数据指标中，查询这些计算好的数据将比查询原始的数据更快更便捷。这对于 Dashboard 场景非常适用，可以解决用户配置以及查询慢的问题。

- record: instance_path:requests:rate5m
expr: rate(requests_total{job="myjob"}[5m])
- record: path:requests:rate5m
expr: sum without (instance)(instance_path:requests:rate5m{job="myjob"})

Prometheus 告警状态

Prometheus 以 `scrape_interval`（默认为1m）规则周期，从监控目标上收集信息。其中 `scrape_interval` 可以基于全局或基于单个metric定义；然后将监控信息持久存储在其本地存储上。

Prometheus 以 `evaluation_interval`（默认为1m）另一个独立的规则周期，对告警规则做定期计算。其中 `evaluation_interval` 只有全局值；然后更新告警状态。

其中包含三种告警状态：

`inactive`：没有触发阈值

`pending`：已触发阈值但未满足告警持续时间

`firing`：已触发阈值且满足告警持续时间

Prometheus 告警状态转换

```
- name: "Redis连接数告警"
  rules:
    - alert: "Redis连接数告警"
      expr: (sum (redis_connected_clients{job="redis_exporter"})by(instance))>1
      for: 30s
      labels:
        severity: critical
      annotations:
        description: '客户端{{ $labels.instance }}实例连接数大于1'
        summary: "Redis连接数告警"
```

收集到Redis连接数告警（数据指标），告警状态为inactive（能够收集到数据的状态）

收集到Redis连接数告警>1，且持续时间小于30s，告警状态为pending

收集到Redis连接数告警>1，且持续时间大于30s，告警状态为firing

注意：配置中的for语法就是用来设置告警持续时间的；如果配置中不设置for或者设置为0，那么pending状态会被直接跳过。

Prometheus告警计算方法

怎样计算告警阈值持续时间？

1. Prometheus以5s (scrape_interval) 一个采集周期采集状态；
2. 根据采集到状态按照1m (evaluation_interval) 一个计算周期，计算表达式；
3. 表达式为真，告警状态切换到pending；
4. 下个计算周期，表达式仍为真，且符合for持续30s，告警状态变更为active，并将告警从Prometheus发送给Alertmanager；
5. 下个计算周期，表达式仍为真，且符合for持续30s，持续告警给Alertmanager；
6. 直到某个计算周期，表达式为假，告警状态变更为inactive，发送一个resolve给Alertmanager，说明此告警已解决。

Prometheus 告警通知

```
- name: "Redis连接数告警"
  rules:
    - alert: "Redis连接数告警"
      expr: (sum (redis_connected_clients{job="redis_exporter"})by(instance))>1
      for: 30s
      labels:
        severity: critical
      annotations:
        description: '客户端{{ $labels.instance }}实例连接数大于1'
        summary: "Redis连接数告警"
```



告警规则触发告警

Alertmanager配置文件

alertmanager.yaml

config:

global:

resolve_timeout: 5m

smtp_smarthost: 'smtp.hostscc.com:25'

smtp_from: 'alert@hostscc.com'

smtp_auth_username: 'alert@hostscc.com'

smtp_auth_password: '4haFWYcfhhzH'

smtp_hello: 'hostscc.com'

smtp_require_tls: false

route:

group_by: ['job', 'severity']

group_wait: 5s

group_interval: 5m

repeat_interval: 12h

receiver: default

receivers:

- name: 'default'

email_configs:

- to: 'ex@163.com'

send_resolved: true

templates:

- '/etc/alertmanager/config/*.tmpl'

#报警恢复5分钟之后发送恢复通知

#发件人的smtp服务器地址

#发件人的邮箱地址

#发件人用户名

#发件人密码

#发件人hello域名

#是否需要启用加密发送

#告警信息分组

#告警等待时间

#新告警产生时,等待多久增加到已发送的告警内容中并发送。

#重复发送告警时间间隔

#接收组

#邮件接收

#接收人地址

#告警恢复之后,是否发送邮件通知

#告警模板

Alertmanager 告警模板

微信模板

wechat.html.tmpl

```
{{ define "wechat.default.message" }}
{{ range $i, $alert := .Alerts }}
告警状态: {{ .Status }}
告警级别: {{ $alert.Labels.severity }}
告警类型: {{ $alert.Labels.alertname }}
告警应用: {{ $alert.Annotations.summary }}
告警主机: {{ $alert.Labels.instance }}
告警详情: {{ $alert.Annotations.description }}
触发阈值: {{ $alert.Annotations.value }}
告警时间: {{ ($alert.StartsAt.Add 28800e9).Format "2006-01-02
15:04:05" }}{{ end }}{{ end }}
```

Alertmanager告警优化手段

如果服务端的告警较多的时候，有效的告警信息会被其他的告警掩盖，所以对Alertmanager做告警优化(收敛).

手段

分组：group

通过告警邮件的分组合并，减少告警数量

抑制：inhibitor

消除冗余的告警

静默：silencer

阻止发送可预期的告警

Alertmanager 告警优化—分组

```
{alertname="server_cpu_high" id="redis-01"}
```

```
{alertname="server_uptime" id="redis-02"}
```

```
{alertname="server_connection" id="redis-02"}
```

```
{alertname="server_memory" id="redis-02"}
```

将告警信息根据id分为两个组

```
{alertname="server_cpu_high" id="redis-01"}
```

```
{alertname="server_uptime" id="redis-02"}
```

```
{alertname="server_connection" id="redis-02"}
```

```
{alertname="server_memory" id="redis-02"}
```

group_by: ["id"]

```
- name: "Redis连接数告警"
  rules:
    - alert: "Redis连接数告警"
      expr: (sum for: 30s
      labels:
        severity: critical
```


Alertmanager 告警优化—抑制

```
groups:
- name: goroutines_monitoring
  rules:
  - alert: TooMuchGoroutines
    expr: go_goroutines{job="prometheus"} > 20
    for: 5m
    labels:
      severity: warning
    annotations:
      summary: "too much goroutines of job prometheus."
      description: "testing "
```

```
- name: goroutines_monitoring
  rules:
  - alert: TooMuchGoroutines
    expr: go_goroutines{job="prometheus"} > 30
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "too much goroutines of job prometheus."
      description: "testing"
```

```
inhibit_rules:
- source_match:
    alternname: 'TooMuchGoroutines'
    severity: 'critical'
  target_match:
    severity: 'warning'
  # Apply inhibition if the alertname is the same.
  equal: ['alertname', 'instance']
```

当已经发送的告警通知匹配到target_match和target_match_re规则，当有新的告警规则如果满足source_match或者定义的匹配规则，并且已发送的告警与新产生的告警中equal定义的标签完全相同，则启动抑制机制，新的告警不会发送。

此例中如果同时配置warning与critical，那么只会报警critical

Alertmanager 告警优化—静默

临时任务导致系统资源超过正常值，比如：跑批任务

<https://lvjianzhao.gitee.io/lvjianzhao/2020/10/08/Alertmanager%E8%AE%BE%E7%BD%AE%E5%91%8A%E8%AD%A6%E9%9D%99%E9%BB%98/>

老男孩教育

Alertmanager 告警对接

1. mail
2. slack
3. victorops
4. pagerduty
5. wechat (企业微信)
6. webhook

老男孩教育 Tony

Thanos

<https://thanos.io/>

1. thanos 是什么?

Open source, highly available Prometheus setup with long term storage capabilities.

2. thanos 能解决什么问题?

2.1 数据接收

2.2 数据存储

2.3 规则报警

Thanos 组件

<https://thanos.io/>

边车组件 (Sidecar)：连接 Prometheus，并把 Prometheus 暴露给查询网关 (Querier/Query)，以供实时查询，并且可以上传 Prometheus 数据给云存储，以供长期保存

查询网关 (Querier/Query)：实现了 Prometheus API，与汇集底层组件 (如边车组件 Sidecar，或是存储网关 Store Gateway) 的数据

存储网关 (Store Gateway)：将云存储中的数据内容暴露出来

压缩器 (Compactor)：将云存储中的数据进行压缩和下采样

接收器 (Receiver)：从 Prometheus 的 remote-write WAL (Prometheus 远程预写式日志) 获取数据，暴露出去或者上传到云存储

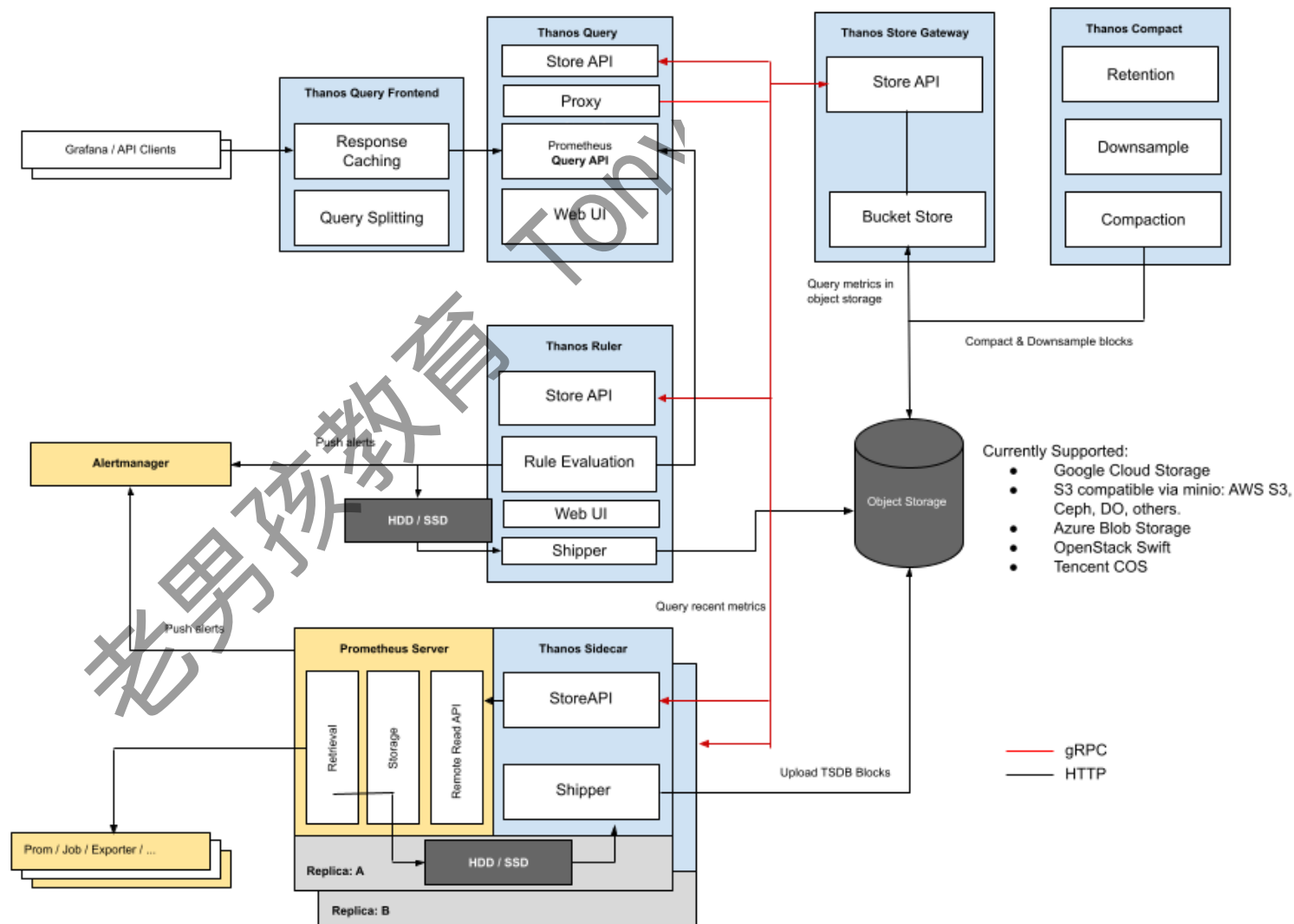
规则组件 (Ruler)：针对监控数据进行评估和报警

Bucket：主要用于展示对象存储中历史数据的存储情况，查看每个指标源中数据块的压缩级别，解析度，存储时段和时间长度等信息。

Thanos 架构

<https://thanos.io/>

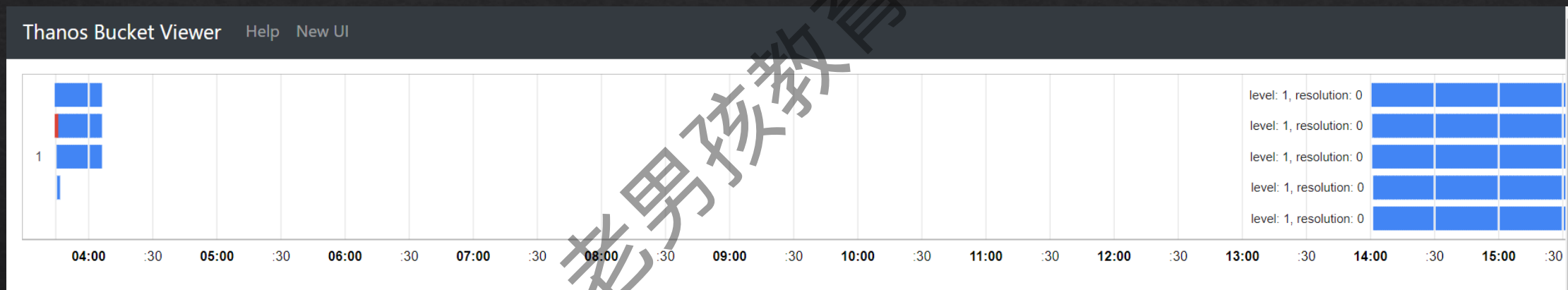
Bucket
Store
Compact
Ruler
receive
Query



Thanos 组件-Bucket

Bucket

用于展示对象存储中历史数据的存储情况,查看每个指标源中数据块的压缩级别,解析度,存储时段和时间长度等信息



Thanos 组件—Store

Store 为可选组件,支持对监控数据长期存储(历史数据).通过配置选项——`objstore.config`,获取到bucket地址,从而获取到监控数据

Kubernetes 中的pvc对象

```
[root@master-2 ~]# kubectl get pvc -n thanos | grep store
data-thanos-store-0      Bound      pvc-b98f75a2-acfa-4165-9faa-f6be3b2530e5
data-thanos-store-1      Bound      pvc-d29d83dc-a140-49bf-8b2a-...
```

支持存储类型:

Google Cloud Storage

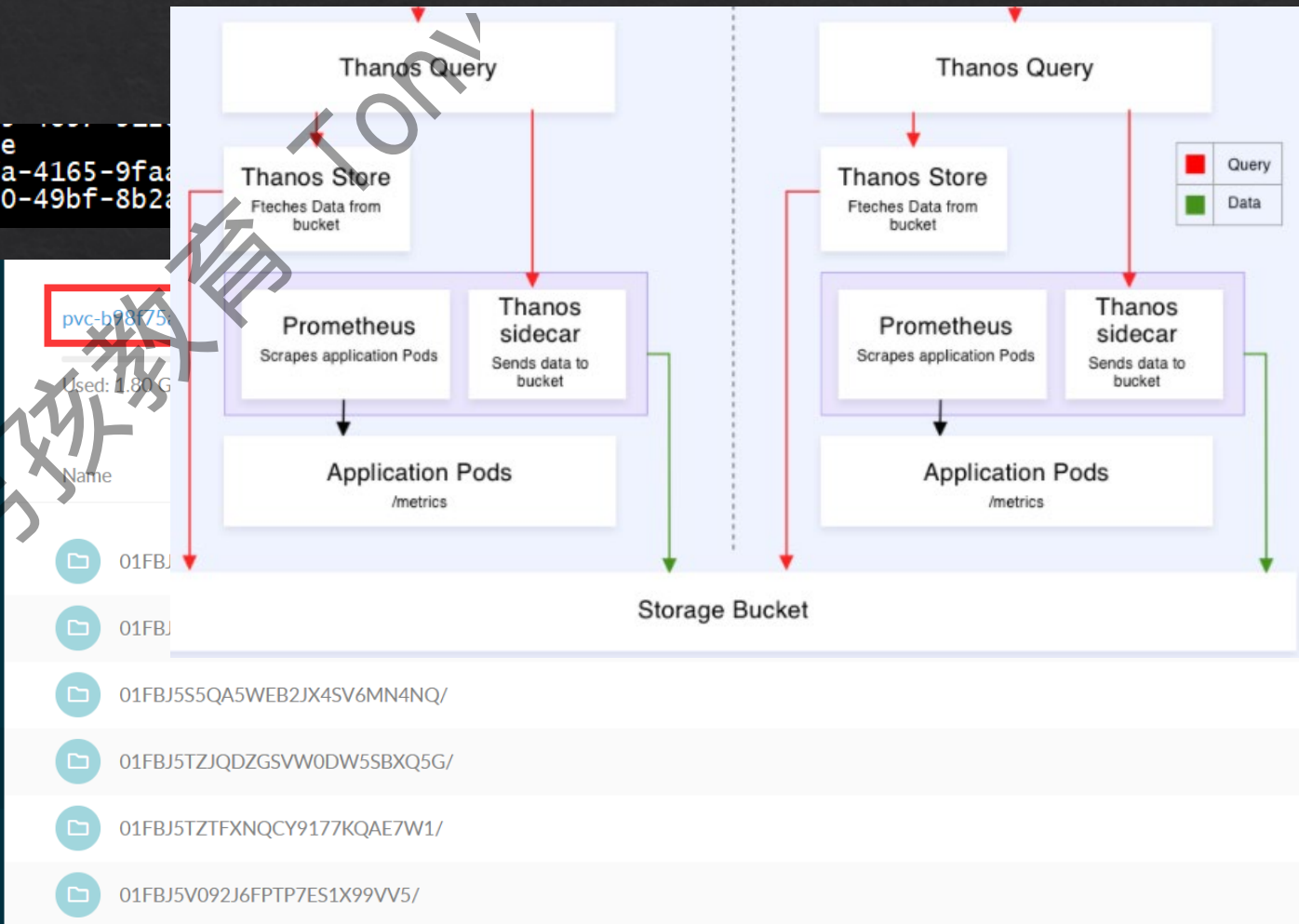
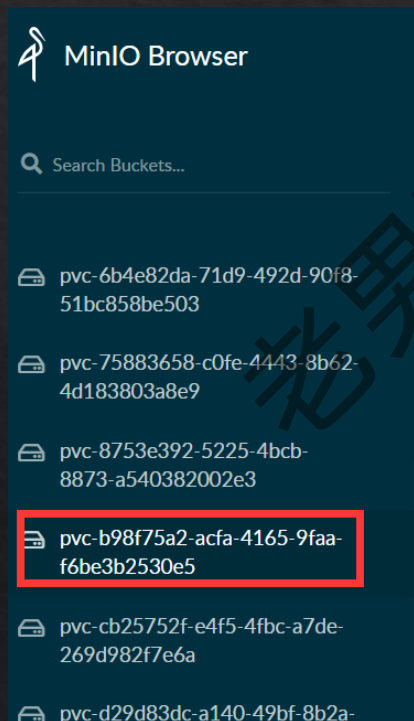
AWS/S3

Azure Storage Account

OpenStack Swift

Tencent COS

AliYun OSS



Thanos 组件-Compact

用来对存储里的数据进行压缩与降采样(数据块中的数据合并),可以提升查询大时间范围监控数据的性能

<https://github.com/thanos-io/thanos/blob/main/docs/components/compact.md>

```
[root@master-1 thanos]# kubectl get pvc -n thanos | grep compact
data-thanos-compact-0          Bound          pvc-18b355d0-6b6e-4554-ba37-7335ba2c8c91    10Gi          RWO          csi-s3          14h
```

The screenshot shows the MinIO Browser interface. On the left, a sidebar lists several buckets. The bucket `pvc-18b355d0-6b6e-4554-ba37-7335ba2c8c91` is selected and highlighted in orange. The main area displays the details for this bucket, including its name, a progress bar, and the text "Used: 1.80 GB". Below this, a list of files is shown, with a single file named `.metadata.json` visible.

Thanos 组件—Rule

可选组件

在prometheus中记录rule的内容有两类:

第一类: 预聚合 (Recording Rule)

常用的指标或者计算相对复杂的指标进行提前计算,然后将这些数据存储到新的数据指标中,解决查询数据慢的问题.

第二类: 告警规则

在分片集群中,每个 Prometheus 只是采集一部分 endpoint 的数据,所以需要 Thanos Ruler 统一设置告警

Thanos 组件—Rule 告警流程

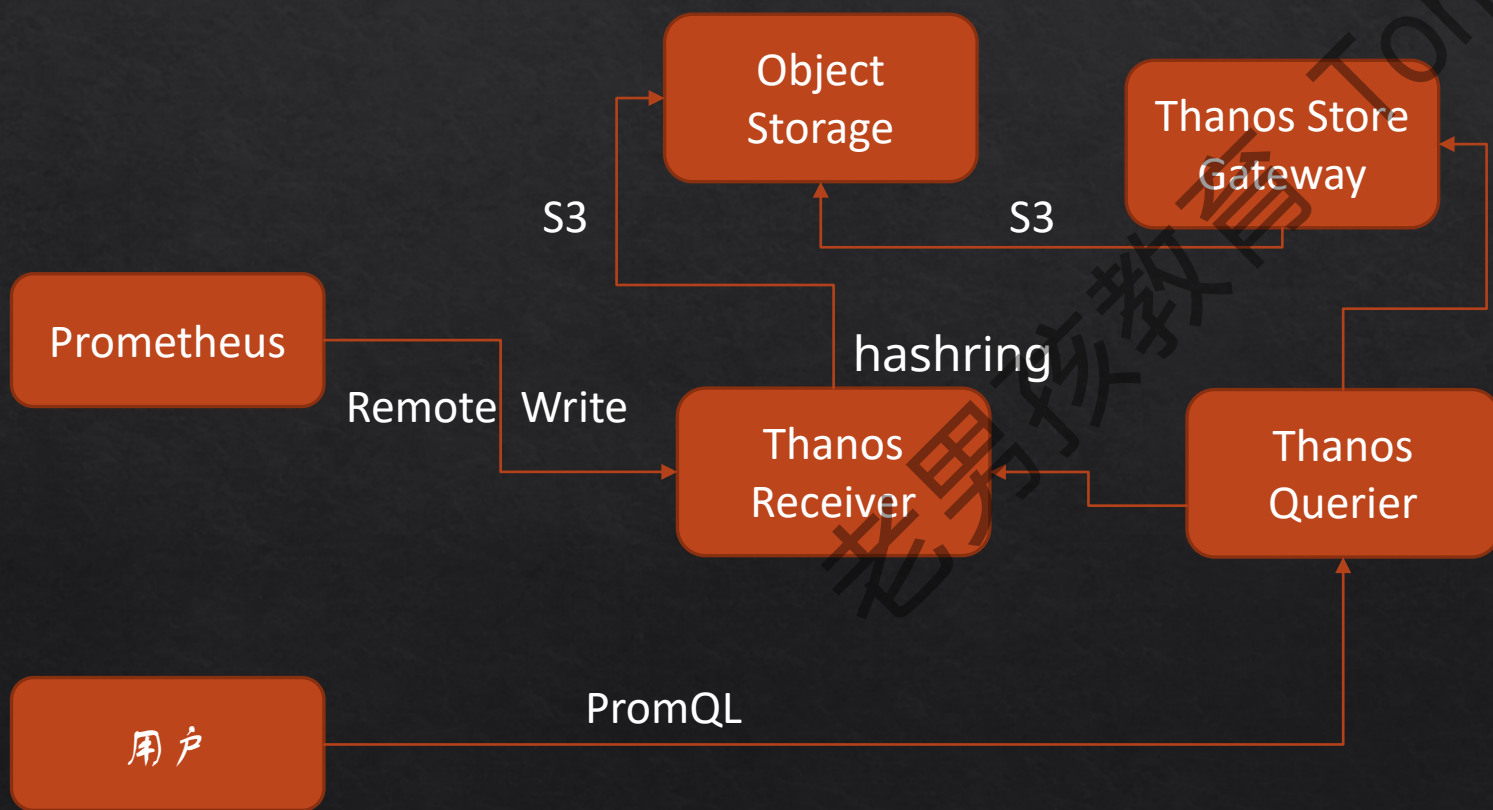
Rule 告警流程: 根据rule-file 配置的规则文件, 去 query 参数接口查询数据, 如果匹配到规则, 触发报警, 则全调用alertmanager服务

```
- args:
- rule
- --log.level=info
- --log.format=logfmt
- --grpc-address=0.0.0.0:10901
- --http-address=0.0.0.0:10902
- --objstore.config=$(OBJSTORE_CONFIG)
- --data-dir=/var/thanos/rule
- --label=rule_replica="$(NAME)"
- --alert.label-drop=rule_replica
- --query=dnssrv+_http._tcp.thanos-query.thanos.svc
- --alertmanagers.url=alertmanager:9093
- --rule-file=/etc/thanos/rules/*.yaml
- |-
```

Thanos Alerts Rules Help New UI				
Rules				
count			6.722s ago	1.673ms
Rule	State	Error	Last Evaluation	Evaluation Time

Thanos 组件—Receive

Receive 是一种支持 Prometheus 远程写入监控数据的组件。是 thanos 对 Prometheus 支持数据写入的一种方式。另外一种方式是 sidecar
thanos-receiver.thanos:19291/api/v1/receive

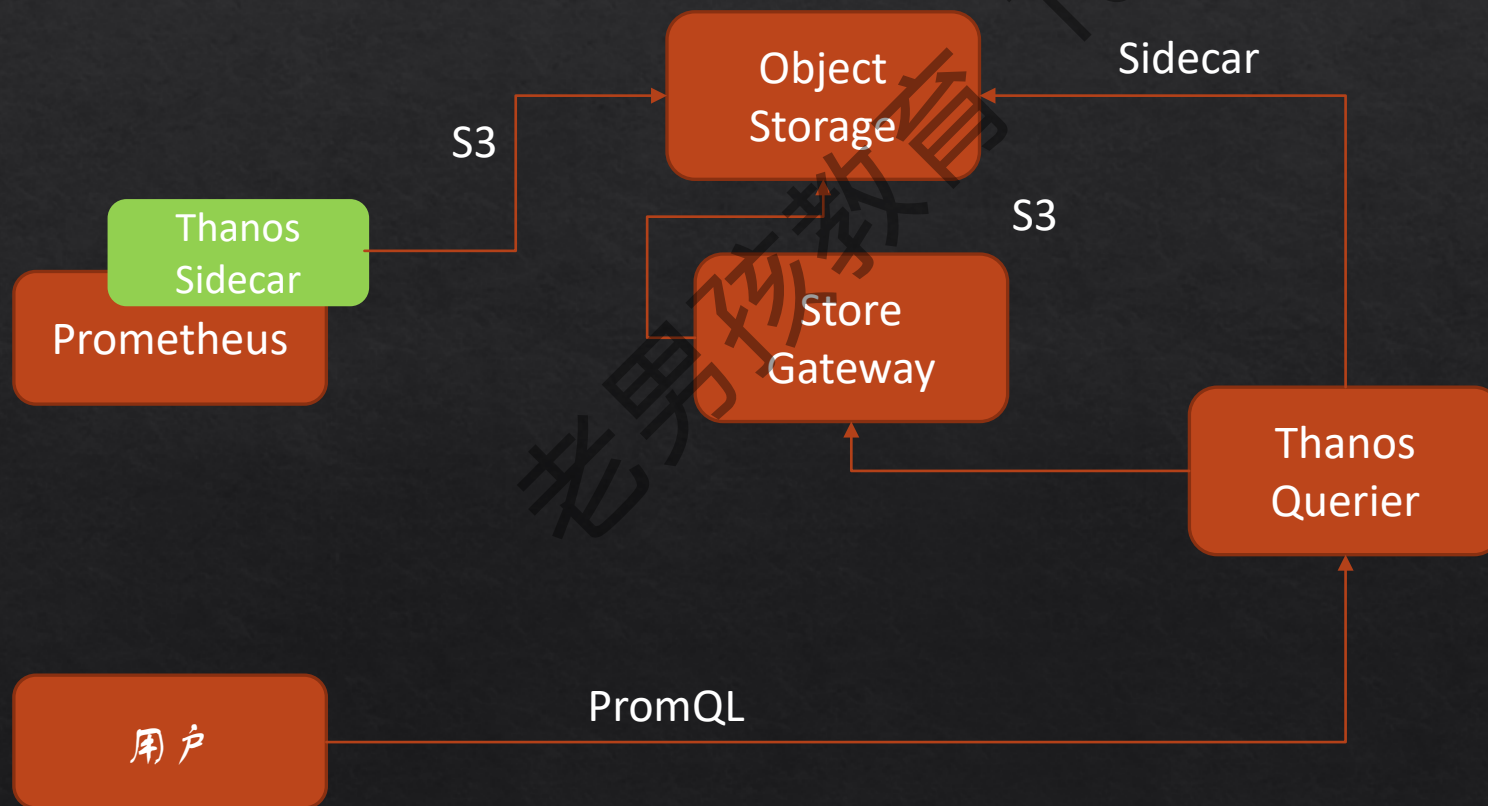


```
args:
- query
- --grpc-address=0.0.0.0:10901
- --http-address=0.0.0.0:9090
- --log.level=debug
- --log.format=logfmt
- --query.replica-label=prometheus_replica
- --query.replica-label=rule_replica
- --store=dnssrv+_grpc._tcp.thanos-receive.thanos
- --store=dnssrv+_grpc._tcp.thanos-store.thanos
- --store=dnssrv+_grpc._tcp.thanos-receive-default-
- --store=dnssrv+_grpc._tcp.thanos-store-0.thanos
- --store=dnssrv+_grpc._tcp.thanos-store-1.thanos
- --store=dnssrv+_grpc._tcp.thanos-store-2.thanos
- --store=dnssrv+_grpc._tcp.prometheus.thanos
- --store=dnssrv+_grpc._tcp.thanos-rule.thanos
- --query.timeout=5m
- --query.lookback-delta=15m
```


Prometheus 另外一种写入方式:Sidecar

配置项

```
- name: thanos
  image: quay.io/thanos/thanos:v0.18.0
  imagePullPolicy: IfNotPresent
  args:
    - sidecar
    - --tsdb.path=/prometheus
    - --prometheus.url=http://localhost:8080
    - --objstore.config=$(OBJSTORE_CONFIG)
    - --reloader.config-file=/etc/prometheus/
```



Thanos组件-Query

提供多个prometheus的数据源的合并查询

Thanos - Query

GraphStoresStatus▼HelpClassic UI

☐ Enable query history

☒ Use local time

☐ Enable Store Filtering

☒ Enable autocomplete

Thanos - Query

Q

container_cpu_cfs_periods_total

Execute

Receive

☒ Use Deduplication

☐ Use Partial Response

Endpoint

172.17.19.11:109

Table

Graph

Load time: 11ms

Resolution: 14s

Result series: 0

1h

+

End time

Res. (s)

Only raw data

Rule

show less

Empty query result

Endpoint	Status	Announced LabelSets	Min Time	Max Time	Last Successful Health Check
10.0.0.51:10901	UP	<div>rule_replica="thanos-rule-0"</div>	2021-07-26 12:43:16	∞	-267.000ms ago

Sidecar

show less

Endpoint	Status	Announced LabelSets	Min Time	Max Time	Last Successful Health Check	Last Message
10.0.0.100:10901	DOWN	<div>cluster="custom2"</div>	2021-07-27 06:00:00	∞	2m 59s ago	fetching store info from 10.0.0.100:10901: rpc error: code = DeadlineExceeded de

Thanos组件-部署方式

第一种方式: 直接启动

```
thanos store data-dir "/local/state/data/dir" \  
--objstore.config-file "bucket.yml"
```

第二种方式: 容器启动

启动参数: <https://github.com/thanos-io/thanos/blob/main/docs/components/store.md>

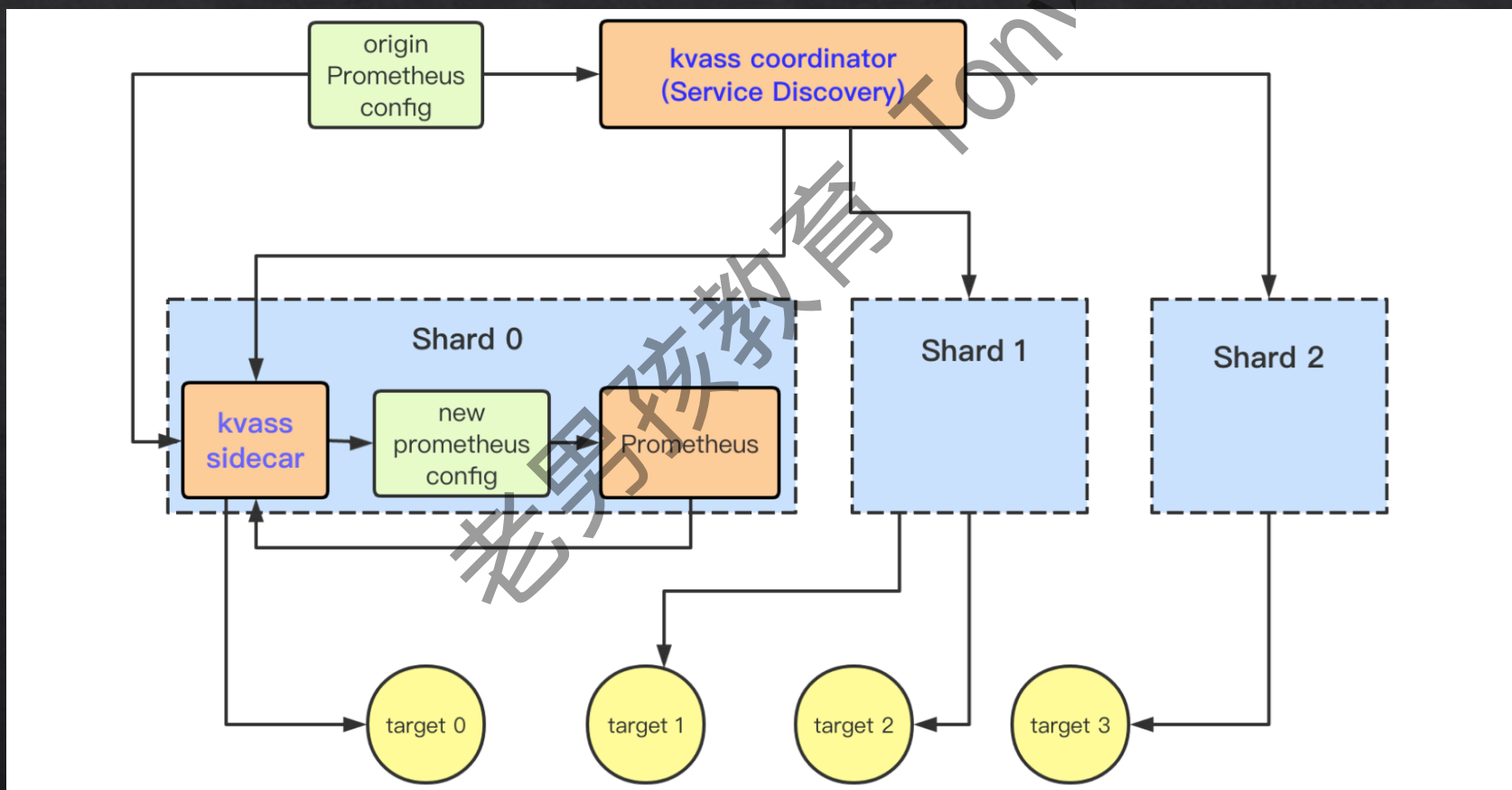
```
containers:  
- args:  
  - query  
  - --grpc-address=0.0.0.0:10901  
  - --http-address=0.0.0.0:9090  
  - --log.level=debug  
  - --log.format=logfmt  
  - --query.replica-label=prometheus_replica  
  - --query.replica-label=rule_replica  
  - --store=dnssrv+_grpc._tcp.thanos-receive.thanos  
  - --store=dnssrv+_grpc._tcp.thanos-store.thanos  
  - --store=dnssrv+_grpc._tcp.thanos-receive-default.thanos  
  - --store=dnssrv+_grpc._tcp.thanos-receive-region-1.thanos  
  - --store=dnssrv+_grpc._tcp.thanos-store-0.thanos  
  - --store=dnssrv+_grpc._tcp.thanos-store-1.thanos  
  - --store=dnssrv+_grpc._tcp.thanos-store-2.thanos  
  - --store=dnssrv+_grpc._tcp.prometheus.thanos  
  - --store=dnssrv+_grpc._tcp.thanos-rule.thanos  
  - --query.timeout=5m  
  - --query.lookback-delta=15m
```


Kvass 是一个 Prometheus 横向扩容解决方案，他有以下特点.

- 轻量，安装方便
- 支持数千万 series 规模 (数千 k8s 节点)
- 无需修改 Prometheus 配置文件，无需加入 hash_mod
- target 动态调度
- 根据 target 实际数据规模来进行分片复杂均衡，而不是用 hash_mod
- 支持多副本

Kvass 架构

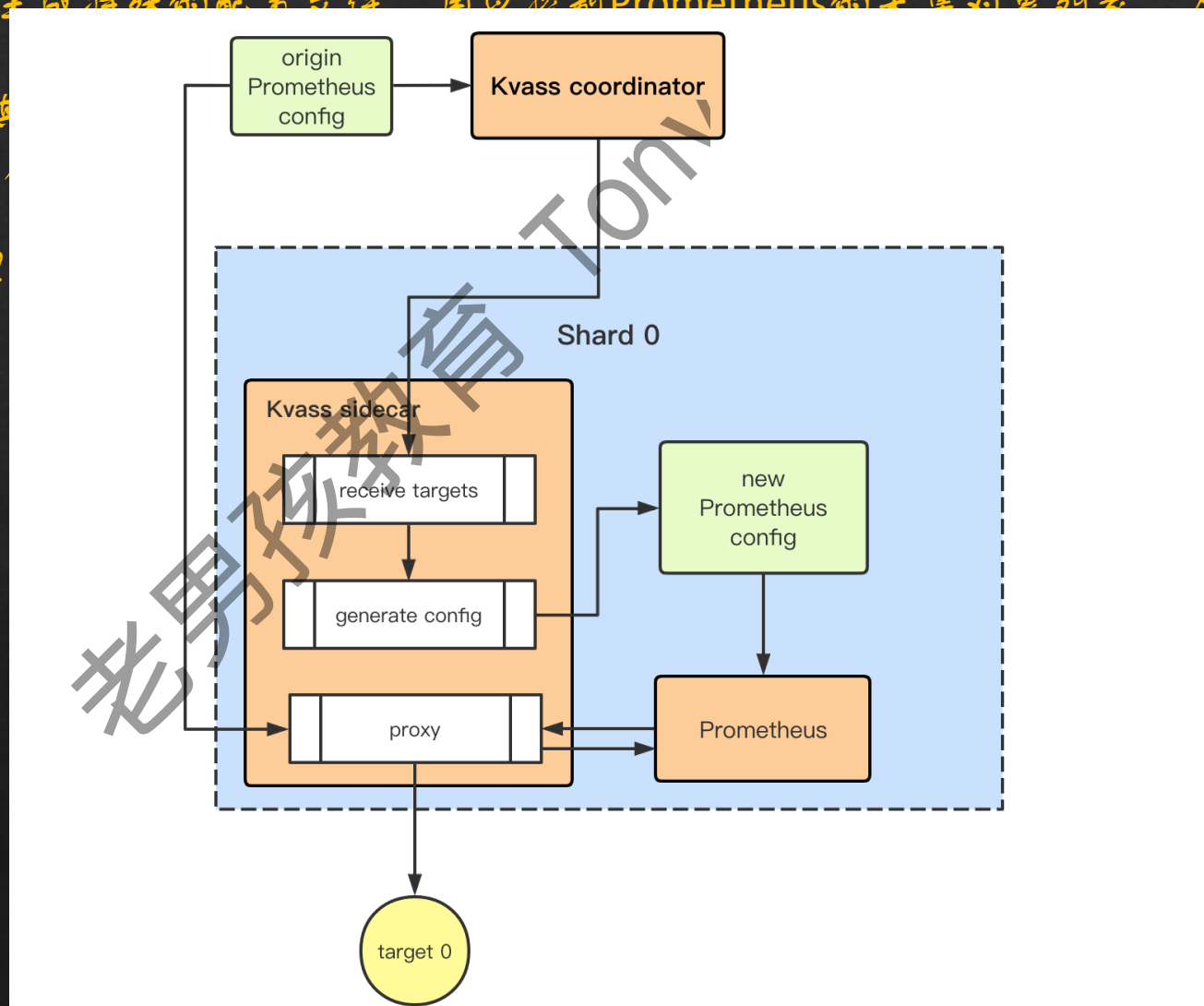
Kvass由2个组件组成：coordinator和sidecar.



Kvass Sidecar 架构

Sidecar负责为其边上的Prometheus生成特殊的配置文件，用以控制Prometheus的采集对象列表。启动参数参考 [code](#)

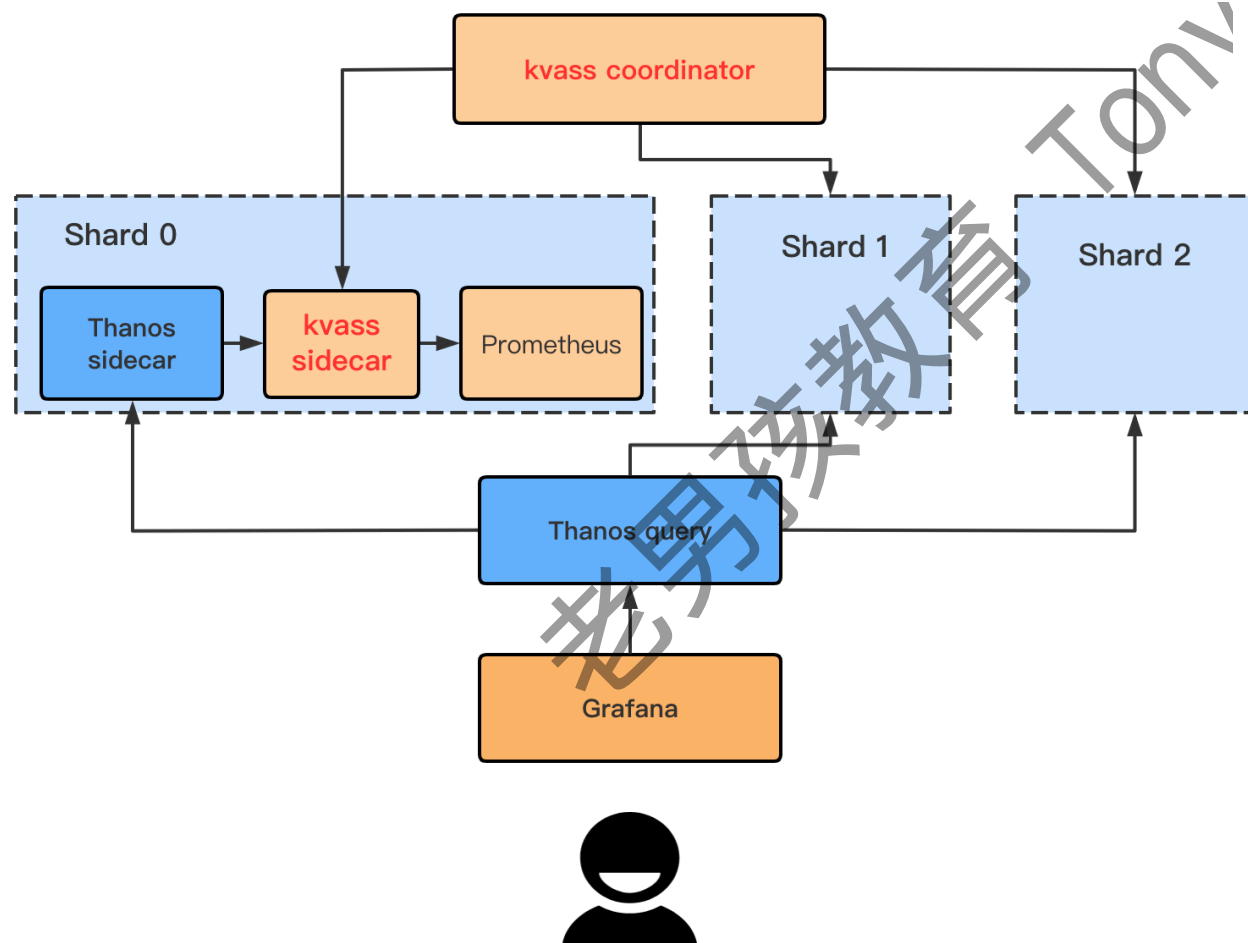
- Sidecar 从Coordinator获得target及其
- Sidecar只使用 "static_configs" 服务，删除所有 "relabel_configs"
- 所有Prometheus抓取请求会被代理



的target, 并删

Kvass 与 Prometheus 集成架构

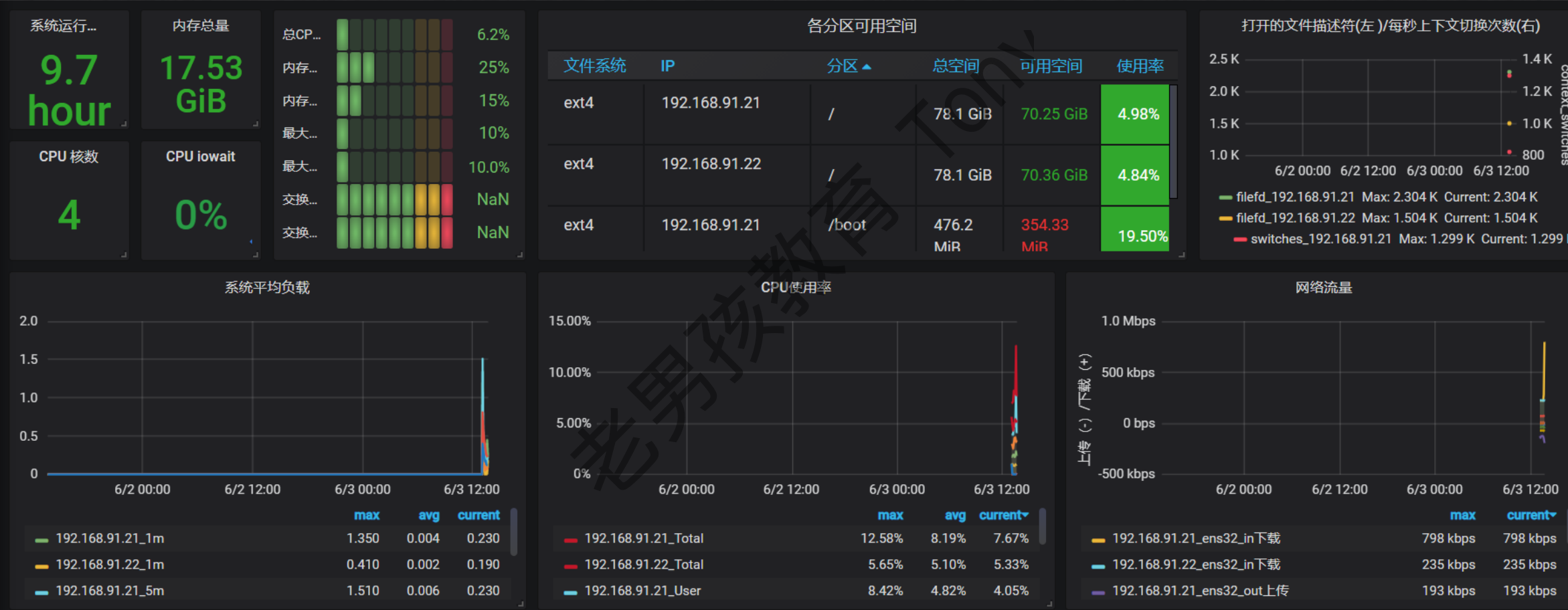
kvass + thanos + prometheus



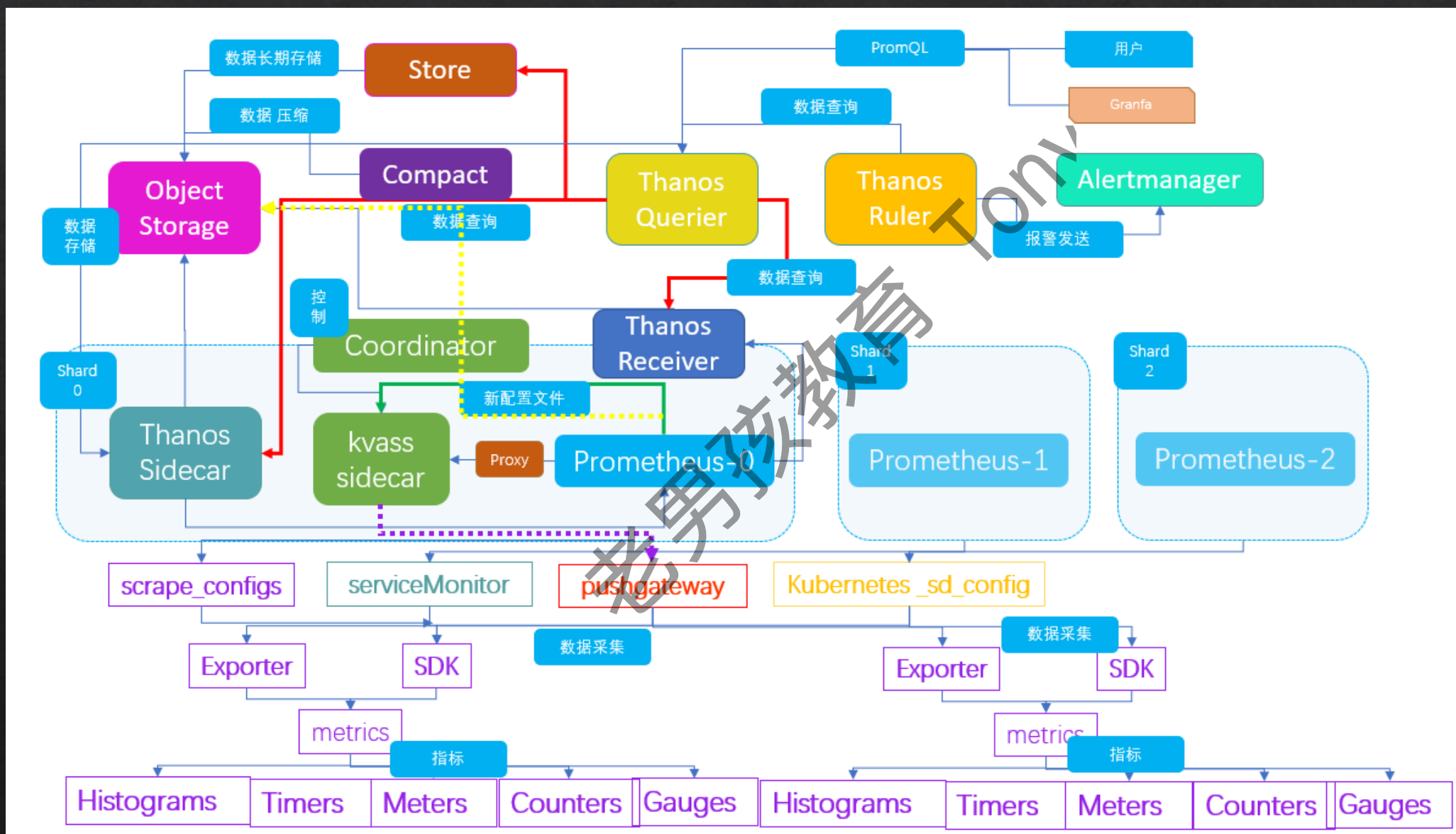
Prometheus 自动分片效果

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
default	nfs-client-provisioner-cb57d6464-cp9zz	1/1	Running	2	12h
default	nginx-pod	1/1	Running	3	43h
kube-system	coredns-867bfd96bd-9kg1g	1/1	Running	1	12h
kubernetes-dashboard	dashboard-metrics-scraper-7b59f7d4df-1gcdx	1/1	Running	1	12h
kubernetes-dashboard	kubernetes-dashboard-79658c479c-s2cfh	1/1	Running	3	12h
thanos	csi-attacher-s3-0	1/1	Running	2	24h
thanos	csi-provisioner-s3-0	2/2	Running	4	24h
thanos	csi-s3-dcq6b	2/2	Running	4	24h
thanos	csi-s3-lxdc7	2/2	Running	4	24h
thanos	kvass-coordinator-5b59d76bcc-m2164	2/2	Running	2	12h
thanos	metrics-774949d94d-2ndnt	1/1	Running	1	12h
thanos	metrics-774949d94d-475tn	1/1	Running	2	17h
thanos	metrics-774949d94d-mznp8	1/1	Running	2	17h
thanos	metrics-774949d94d-q54jv	1/1	Running	1	12h
thanos	metrics-774949d94d-w9lps	1/1	Running	1	12h
thanos	metrics-774949d94d-xt7kz	1/1	Running	1	12h
thanos	minio-665449897f-bsxmp	1/1	Running	2	26h
thanos	prometheus-0	3/3	Running	3	10h
thanos	prometheus-1	3/3	Running	3	10h
thanos	prometheus-2	3/3	Running	3	10h
thanos	prometheus-3	3/3	Running	0	3m17s
thanos	prometheus-4	3/3	Running	1	3m17s
thanos	thanos-bucket-647689c6bc-brbk9	1/1	Running	2	16h
thanos	thanos-compact-0	1/1	Running	9	10h
thanos	thanos-query-79d5fc7758-fccf2	1/1	Running	1	11h

Kubernetes 中监控系统资源效果



Prometheus 自动分片架构



Thanks