

一小时玩转云原生监控实操

孟凡杰

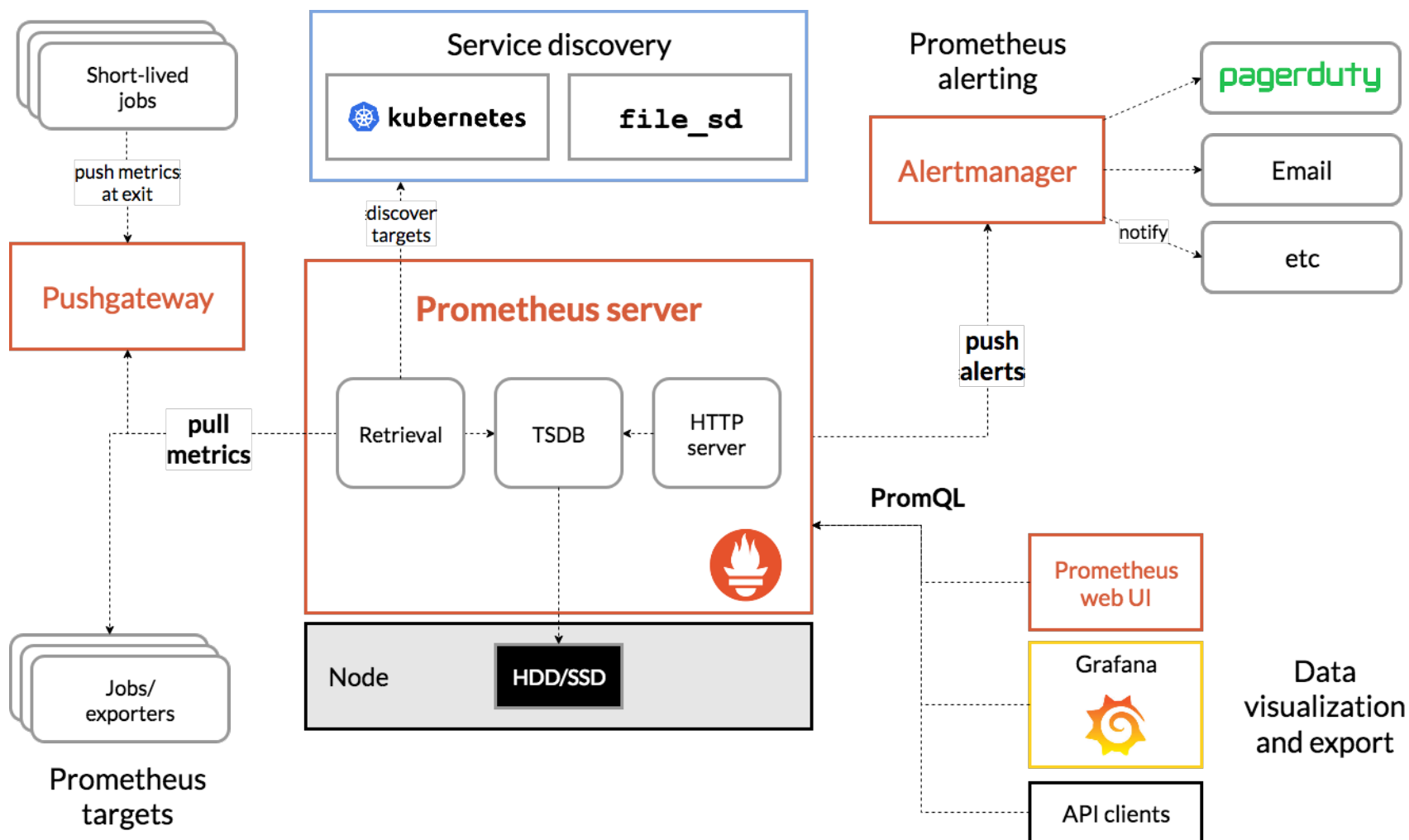
腾讯云容器技术专家



目录

1. Prometheus 服务发现机制
2. 深入理解 Relabel 机制和配置
3. 基于 AlertManager 的告警

重温 Prometheus 架构



Prometheus 典型配置

```
global:
  scrape_interval: 15s # 指标抓取周期, 默认 1 分钟
  evaluation_interval: 15s # 规格评估周期, 默认 1 分钟
  scrape_timeout: 15s # 指标抓取超时时间, 默认一分钟
```

告警配置, alertmanager 地址

```
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        # - alertmanager:9093
```

一次性加载规则, 并按照 'evaluation_interval' 定期评估规则, 判断是否触发告警

```
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"
```

指标抓取目标配置

```
scrape_configs:
  # 任何抓取的指标都会添加新的 `job=<job_name>` 的 label
  - job_name: 'prometheus'
```

metrics_path 默认为 '/metrics'

协议默认为 'http'.

静态目标配置

```
static_configs:
  - targets: ['localhost:9090']
```

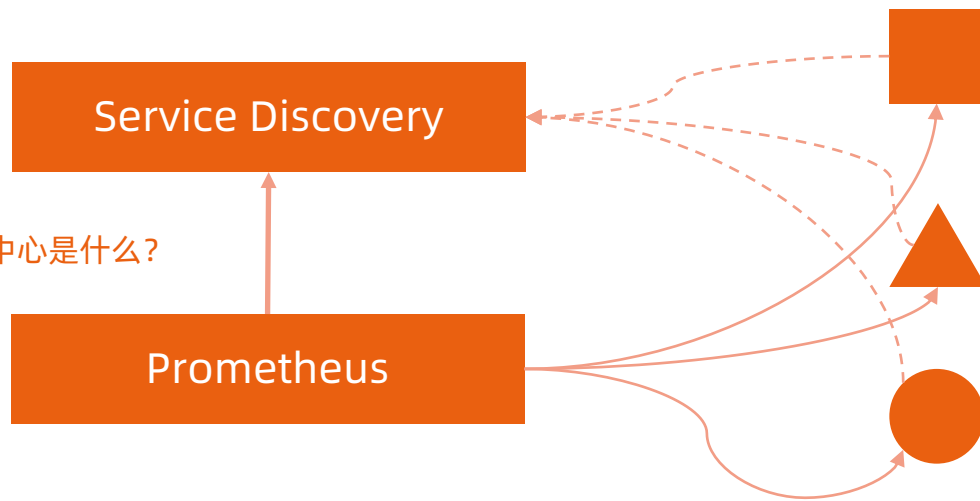
配置加载: `prometheus --config.file=/etc/prometheus/prometheus.yml --storage.tsdb.path=/data/prometheus`

Prometheus 的服务发现

云原生环境下服务发现的挑战

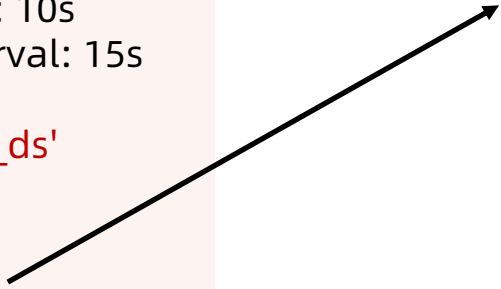
- 云原生应用当前已无状态为主，无状态意味着实例可被任意替换。
 - 即使有状态应用，通常也不会有固定 IP 地址等静态配置，相反 Pod 更新后通常会更换 IP 地址。
 - 自动扩缩容与故障转移使得 Pod IP 不停变换。
- 因此，静态配置变得不再适用，云原生场景下的目标发现与服务发现机制一致。

思考：Kubernetes 体系下的服务注册中心是什么？



基于文件的服务发现

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s
scrape_configs:
- job_name: 'file_ds'
  file_sd_configs:
  - files:
    - targets.json
```



```
targets.json
[
  {
    "targets": [ "localhost:8080" ],
    "labels": {
      "env": "dev",
      "app": "mywebserver"
    }
  },
  {
    "targets": [ "localhost:8081" ],
    "labels": {
      "env": "test",
      "app": "mydb"
    }
  },
  {
    "targets": [ "localhost:8082" ],
    "labels": {
      "env": "prod",
      "app": "myredis"
    }
  }
]
```

服务发现配置

```
global:
  scrape_interval: 15s
  scrape_timeout: 10s
  evaluation_interval: 15s
  scrape_configs:
    - job_name: kubernetes-apisservers
      metrics_path: /metrics
      scheme: https
      kubernetes_sd_configs:
        - api_server: https://192.168.2.1:6443/ # 将 Kubernetes API Server 作为服务注册中心
          role: endpoints
          bearer_token_file: /prometheus/k8s_token # 访问 Kubernetes 的 Token
          tls_config:
            insecure_skip_verify: true # 跳过证书安全检查
      bearer_token_file: /prometheus/k8s_token
      tls_config:
        insecure_skip_verify: true
```

relabel_configs: # Relabel 配置

```
- source_labels: [__meta_kubernetes_namespace, __meta_kubernetes_service_name,
__meta_kubernetes_endpoint_port_name]
  separator: ;
  regex: default;kubernetes;https
  replacement: $1
  action: keep
- separator: ;
  regex: (.*?)
  target_label: __address__
  replacement: 192.168.2.1:6443
  action: replace
```

Relabel

为什么需要 Relabel ?

- 按照不同的环境 dev, test, prod 聚合监控数据;
- 对于研发团队而言, 我可能只关心 dev 环境的监控数据;
- 对于超大集群, 我可能只关心部分关键指标的监控, 忽略非换件指标, 从而降低监控系统负担;
- 多租户场景下, 多个团队的 Prometheus Server 可能只需要采集自己的业务数据。

面对以上这些场景下的需求时, 我们实际上是希望 Prometheus Server 能够按照某些规则 (比如标签) 从服务发现注册中心返回的 Target 实例中有选择性的采集某些 Exporter 实例的监控数据。

Prometheus 的 Relabeling 机制

在 Prometheus 所有的 Target 实例中，都包含一些默认的 Metadata 标签信息。可以通过 Prometheus UI 的 Targets 页面中查看这些实例的 Metadata 标签的内容，Metadata 标签以下划线开始，通常对指标使用用户不可见。

`__address__`: 当前 Target 实例的访问地址 `<host>:<port>`

`__scheme__`: 采集目标服务访问地址的 HTTP Scheme, HTTP 或者 HTTPS

`__metrics_path__`: 采集目标服务访问地址的访问路径

`__param_<name>`: 采集任务目标服务的中包含的请求参数

从真实的 Prometheus 控制台中我们看到类似如下的指标：

```
http_response_time{env="test",instance="localhost:8081",app="mywebserver"}
```

这里的 `instance` 是从 `__address__` 转换来的，这就是 Prometheus 的 Relabel 机制。

再看 Relabel Config

Relabel 用来重写 Target 的标签。

每个 Target 可以配置多个 Relabel 动作，按照配置文件顺序应用。

内置的标签都可以用于 Relabel，在 Relabel 时未保留，内置标签将被删除。

action: 执行的 Relabeling 动作，可选值包括 replace、keep、drop、hashmod、labelmap、labeldrop 或者 labelkeep，默认值为 replace。

separator: 分隔符，一个字符串，用于在连接源标签 source_labels 时分隔它们，默认为 ;。

source_labels: 源标签，使用配置的分隔符串联的标签名称列表，并与提供的正则表达式进行匹配。

target_label: 目标标签，当使用 replace 或者 hashmod 动作时，应该被覆盖的标签名。

regex: 正则表达式，用于匹配串联的源标签，默认为 (.*)，匹配任何源标签。

modulus: 模数，串联的源标签哈希值的模，主要用于 Prometheus 水平分片。

replacement: replacement 字符串，写在目标标签上，用于替换 relabeling 动作，它可以参考由 regex 捕获的正则表达式捕获组。

Relabel 示例

设置一个固定的标签值

```
action: replace
replacement: production
target_label: env
```

保留或丢弃对象

```
scrape_configs:
- ...
- job_name: "mywebserver"
  relabel_configs:
    - action: keep
      source_labels:
        - __address__
      regex: mywebserver01.*
```

替换抓取任务端口

```
action: replace
source_labels: [__address__]
regex: ([^:]+)(?:\d+)?
replacement: "$1:80"
target_label: __address__
```

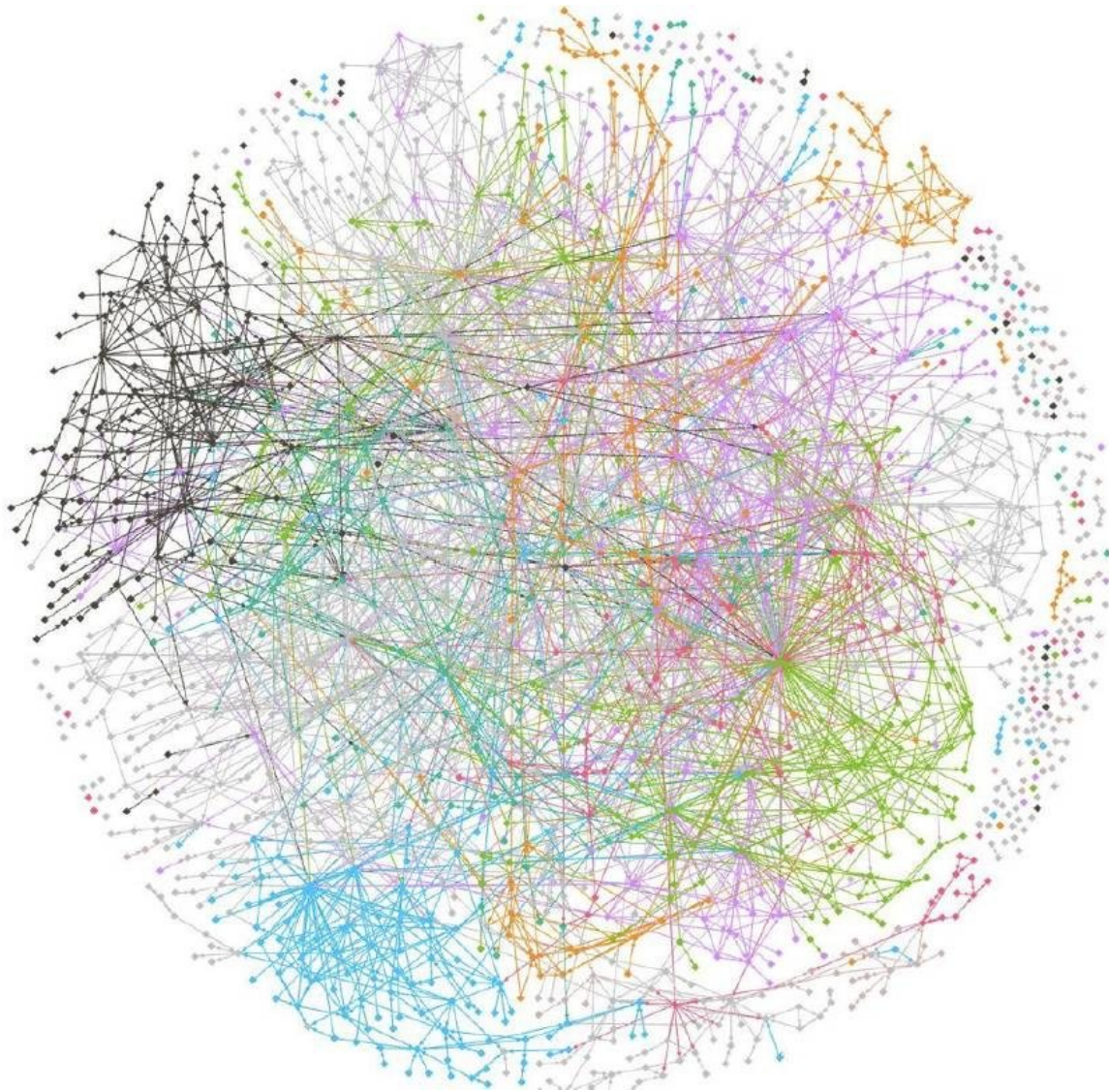
只抓取具有注解的目标

```
action: keep
source_labels:
  [__meta_kubernetes_service_annotation_prometheus_scraped]
regex: true
```

mywebserver 这个 job 中 __address__ 标签值为 mywebserver01.* 的指标会被 Keep，其他会被 Drop。

云原生告警

看图说话：为什么需要告警系统？



告警困境

困境 1：没有告警

系统装箱过度，导致节点 CPU 100%，业务已崩而调度器还在不停调度 Pod。

困境 2：海量告警 = 没有告警

一天 200 个告警电话，电话接到没电，处理一个告警的时候同时又收到 10 个告警。

困境 3：可自愈的告警该不该告警

周末背着电脑在外玩，接到告警电话立马在路边处理，登录到系统发现已经自动恢复了。关上电脑又出现。



Prometheus 典型配置

```
global:
  scrape_interval: 15s # 指标抓取周期, 默认1分钟
  evaluation_interval: 2m # 规格评估周期, 默认1分钟
  scrape_timeout: 15s # 指标抓取超时时间, 默认一分钟
```

告警配置, alertmanager 地址

```
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        # - alertmanager:9093
```

一次性加载规则, 并按照 'evaluation_interval' 定期评估规则, 判断是否触发告警

```
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"
```

指标抓取目标配置

```
scrape_configs:
  # 任何抓取的指标都会添加新的 `job=<job_name>` 的 label
  - job_name: 'prometheus'
```

```
  # metrics_path 默认为 '/metrics'
  # 协议默认为 'http'.
```

静态目标配置

```
static_configs:
  - targets: ['localhost:9090']
```

配置加载: `prometheus --config.file=/etc/prometheus/prometheus.yml --storage.tsdb.path=/data/prometheus`

告警规则 (Alerting Rule)

Rules

example	592.8us
Rule	Evaluation Time
<pre>alert: HighErrorRate expr: job:request_latency_seconds:mean5m{job="myjob"} > 0.5 for: 10m labels: severity: page annotations: summary: High request latency</pre>	237.4us
<pre>alert: DeadMansSwitch expr: vector(1) labels: severity: none annotations: description: This is a DeadMansSwitch meant to ensure that the ent is functional. summary: Alerting DeadMansSwitch</pre>	

Alerts

Show annotations

DeadMansSwitch (1 active)

```
alert: DeadMansSwitch
expr: vector(1)
labels:
  severity: none
annotations:
  description: This is a DeadMansSwitch meant to ensure that the entire Alerting pipeline
    is functional.
  summary: Alerting DeadMansSwitch
```

Labels	State	Active Since	Value
alertname="DeadMansSwitch" severity="none"	FIRING	2018-07-24 18:38:25.582918892 +0000 UTC	1

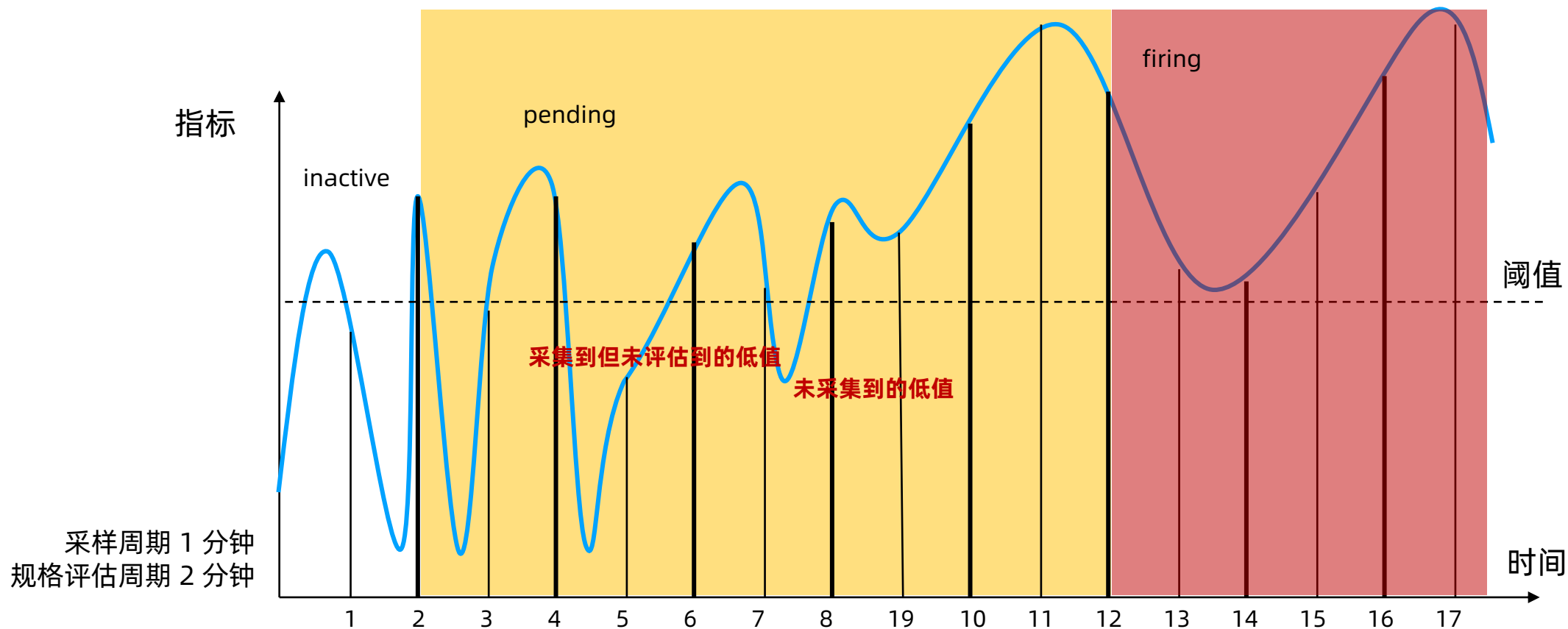
HighErrorRate (0 active)

理解标准告警持续时间

for: 10m 表示指标超过阈值持续时间（即 `expr=true`）需要超过 10 分钟以后才触发告警，其核心目的是确保过滤掉抖动引发的瞬时异常。

但依然会有潜在问题：

- 采样周期的存在，所有指标是离散的，地域阈值的指标可能未被采集到。
- 采集周期和告警规则评估周期是不同的，采集到的正常值可能未被评估到。



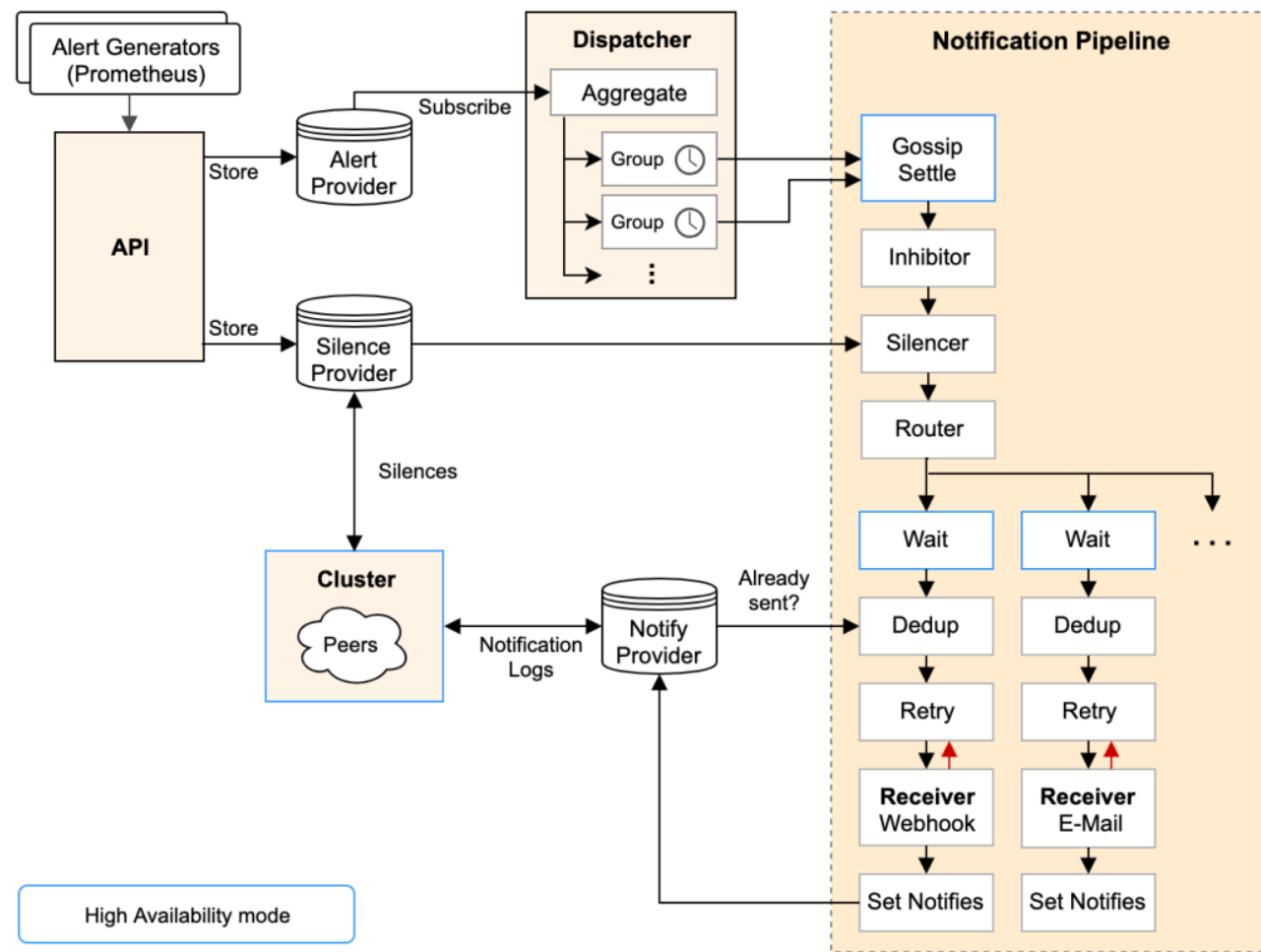
录制规则（Recording Rule）

- 复杂的 **PromQL** 开销很大，基于 **Recording Rule** 可以将开销较大的复杂规则进行预处理并将其结果记录成一条新的时间序列。
- 查询处理好的结果数据比每次查询重新计算效率更高。
- 基于 **Recording Rule** 的结果查询可以频次更高。
- Recording Rule 通常用于 **Grafana** 看板的优化，比如某个 Dashboard 的查询较为复杂，则可以通过 Recording Rule 将查询结果记录下来，供 **Dashboard** 查询。

```
groups:  
- name: example  
  rules:  
  - record: job:http_inprogress_requests:sum  
    expr: sum(http_inprogress_requests) by (job)
```

AlertManager 架构

- Prometheus 发送的警报到 Alertmanager。
- 警报会被存储到 AlertProvider，内置实现是一个 Memory Map。
- Dispatcher 是一个单独的 goroutine，它会不断到 AlertProvider 拉新的警报，并且根据 YAML 配置的 Routing Tree 将警报路由到一个分组中。
- 分组会定时进行 flush（间隔为配置参数中的 group_interval），flush 后这组警报会走一个 Notification Pipeline 链式处理。
- Notification Pipeline 为这组警报确定发送目标，并执行抑制逻辑（如集群故障告警触发时可抑制该集群某个组件的告警规则，防止大面积告警），静默逻辑，去重逻辑，发送与重试逻辑，实现警报的最终投递。



路由规则配置

根路由规则，参数被子规则继承

route:

receiver: 'default-receiver' ←
group_wait: 30s
group_interval: 5m
repeat_interval: 4h ←
group_by: [cluster, alertname]

匹配不到子路由规则的告警会被根路由规则接受，并被转发至 default-receiver

routes:

带着 service=mysql 或 service=cassandra 标签的告警会被转发至 database-pager

All alerts with service=mysql or service=cassandra
are dispatched to the database pager.

- receiver: 'database-pager'
group_wait: 10s
matchers:
- service=~"mysql|cassandra"

带着 team=frontend 标签的告警会被转发至 frontend-pager

- receiver: 'frontend-pager'
group_by: [product, environment]
matchers:
- team="frontend"

receivers:

- name: default-receiver
email_configs:
- to: <mail to address>
send_resolved: true

- group_interval : 控制了这个分组最快多久通知一次
- repeat_interval: 重复触发告警的重复触发间隔
- group_by: 决定了警报怎么分组，每个 group 只会定时产生一次通知

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
- ✓ 简历辅导并直推知名企业

15 周

系统集训

15 大

内容模块

6 大

实践案例

105 天

助教答疑

2 次

企业内推

