# Kubernetes监控

## 概述

本文描述了kubernetes集群的监控方案，主要描述了kubernetes集群中的prometheus的部署，prometheus与数据平台的cortex对接，告警以及数据展示相关内容。

## 架构

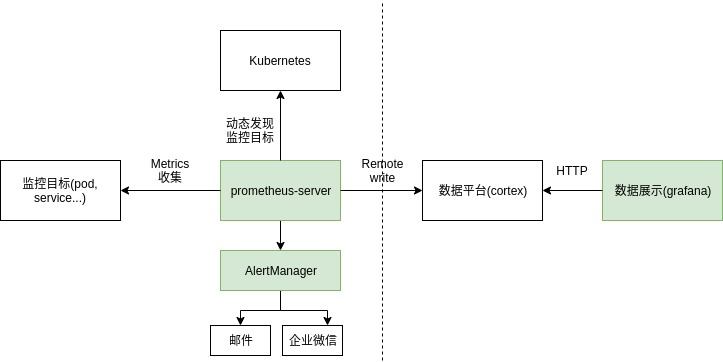


图2-1 整体架构

图中绿色部分属于监控部署范围。

* Kubernetes集群内部部署一套完整独立的基于prometheus的监控系统。
* 集群内部部署告警组件，无法聚合多个Kubernetes集群之间的数据。
* 集群内部的prometheus将监控数据推送给数据平台的cortex，remote\_write的url和认证方式由数据平台提前分配。
* 数据通过grafana展示，grafana的数据源配置为数据平台提供的url，dashboard按需自定义，可以参考或者直接使用grafana官方提供的dashboard。

## 部署配置

### 3.1集群内prometheus-server对接cortex

* 从数据平台获取remote\_write的url和认证信息，在prometheus配置文件添加remote\_write段，并添加从数据平台获取的信息。

serverFiles:

prometheus.yml:

# 将数据通过remote\_write发送到数据平台网关, 目前需要手动选择就近的网关地址

remote\_write:

- url: <数据平台提供的url>

- basic\_auth:

username: <数据平台提供的url>

password: <数据平台提供的url>

* 在global下面配置extenal\_labels段，来制定采集到的metrics的所属site和集群信息。

global:

external\_labels:

[ site: aliyun / tenxun... ]

[ cluster: prod / pre-prod... ]

### 3.2 部署平台使用kubernetes集群

部署平台在kubernetes部署的pod如果要采集metrics，要满足两个条件。

1. pod提供metrics接口。
2. pod的annotation需要加上指定的简直对。

* 必须项

prometheus.io/scrape: "true"

* 可选项

prometheus.io/path: 默认路径是/metrics，可按需更改路径。

prometheus.io/port: 默认不指定，可以按需更改port。

## 监控对象

prometheus针对Kubernetes集群的监控，利用了kubernetes\_sd\_configs服务发现机制，prometheus不仅可以监控集群内的静态对象，如node。还可以动态的发现集群内的资源对象并进行监控。我们主要关注node和pod的指标，除此之外kubernetes\_sd\_configs还支持service, endpoints, ingress这3这角色的指标采集。

* 1. 主机监控

利用node\_exporter来完成数据指标的采集工作，node\_exporter支持常见的监控指标，比如CPU，内测，网络，文件系统，磁盘I/O，连接数等。node\_exporter默认开启的监控指标如下表所示：



* 1. 容器监控

kubelet内置的cadvisor提供容器监控指标。主要提供了下面四类指标：

* CPU：指标以container\_cpu开头，比如container\_cpu\_user\_seconds\_total表示容器用户台CPU时间总和。
* 内测：指标以container\_memory开头，比如container\_memory\_usage\_byters表示内测总占用量。
* 磁盘IO：指标以container\_fs开头，比如container\_fs\_writes\_bytes\_total表示磁盘写总量。
* 网络：指标以container\_network开头，比如container\_network\_recive\_bytes\_total表示接收的字节总数。

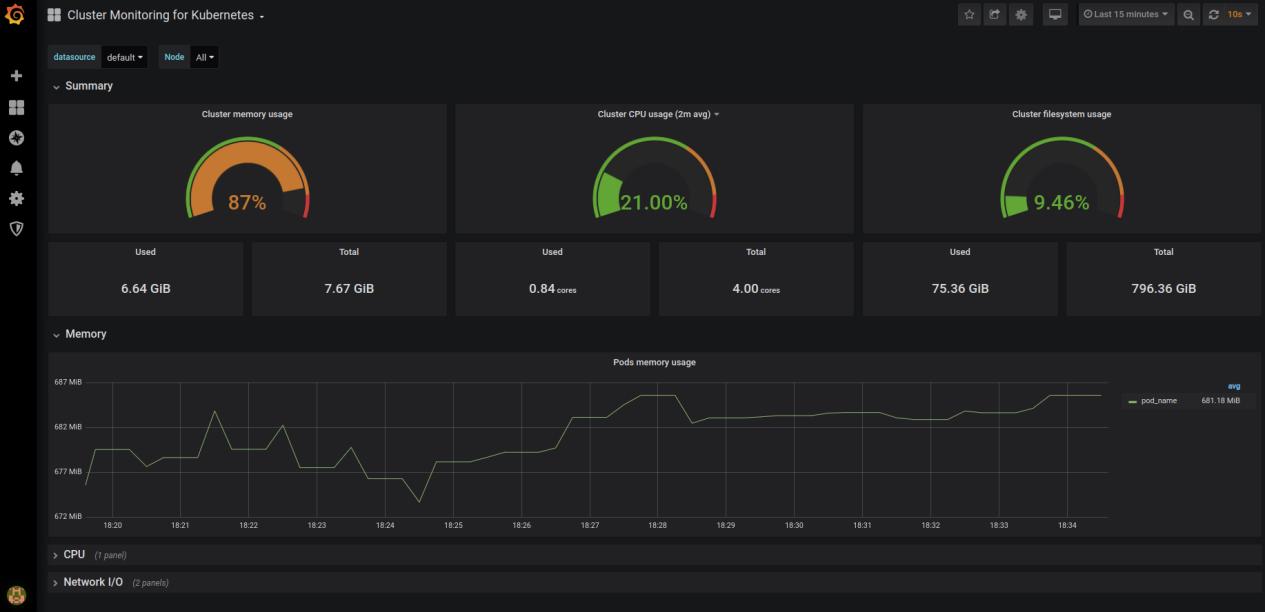
Kubernetes的资源对象的指标都可以在容器的基础上做聚合来得到。

## 数据展示

我们利用grafana来做数据展示，grafana利用helm部署在kubernetes集群内部，grafana的数据源来是数据平台提供的url。

可以根据自己的需求新建dashboard，也可以从grafana的官网下载kubernetes相关的dashboard，根据需求改造dashboard。

* 集群状况展示



* Node状况展示



* Pod状况展示



## 告警

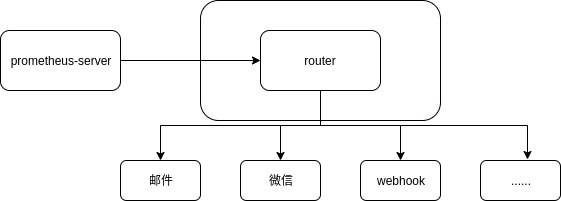


图6-1 alertmanager工作流程

### 6.1告警规则

prometheus-server的配置文件里添加需要告警的规则。最关键的是是用PromQL来定义告警规则。

我们这里添加了3条测试的报警规则，分别是节点的文件系统、节点内存和CPU的使用量，如果大于了80%的话就触发label为team=node的receiver。label会在alertmanager里被使用到，receiver在alertmanager里定义。

groups:

- name: test-rule

rules:

- alert: NodeFilesystemUsage

expr: (node\_filesystem\_size{device="rootfs"} - node\_filesystem\_free{device="rootfs"}) / node\_filesystem\_size{device="rootfs"} \* 100 > 80

for: 2m

labels:

team: node

annotations:

summary: "{{$labels.instance}}: High Filesystem usage detected"

description: "{{$labels.instance}}: Filesystem usage is above 80% (current value is: {{ $value }}"

- alert: NodeMemoryUsage

expr: (node\_memory\_MemTotal - (node\_memory\_MemFree+node\_memory\_Buffers+node\_memory\_Cached )) / node\_memory\_MemTotal \* 100 > 80

for: 2m

labels:

team: node

annotations:

summary: "{{$labels.instance}}: High Memory usage detected"

description: "{{$labels.instance}}: Memory usage is above 80% (current value is: {{ $value }}"

- alert: NodeCPUUsage

expr: (100 - (avg by (instance) (irate(node\_cpu{job="kubernetes-node-exporter",mode="idle"}[5m])) \* 100)) > 80

for: 2m

labels:

team: node

annotations:

summary: "{{$labels.instance}}: High CPU usage detected"

description: "{{$labels.instance}}: CPU usage is above 80% (current value is: {{ $value }}"

### 6.2告警通知

prometheus根据配置的规则产生告警并推算给alertmanager，alertmanager根据配置将告警推送给目标。

#### 6.2.1邮件通知例子

global:  
 resolve\_timeout: 5m  
 smtp\_smarthost: 'smtp.163.com:465'  
 smtp\_from: 'abc@163.com'  
 smtp\_auth\_username: 'abc@163.com'  
 smtp\_auth\_password: '<password>'  
 smtp\_require\_tls: false   
  
# 所有报警信息进入后的根路由，用来设置报警的分发策略  
route:  
 group\_by: ['chentest'] # 这里的标签列表是接收到报警信息后的重新分组标签  
 group\_wait: 10s # 第一次等待多久时间发送一组警报的通知  
 group\_interval: 10s # 在发送新警报前的等待时间  
 repeat\_interval: 1m # 如果一个报警信息已经发送成功了，等待'repeat\_interval'时间来重新发送他们  
 receiver: 'xxx-email' # 发送警报的接收者的名称，与receivers name的名称相同  
  
receivers:  
- name: 'xxx-email'  
 email\_configs: # 邮箱配置  
 - send\_resolved: true # 告警解决是否通知，默认是不通知  
 to: 'abc@xxx.com'

#### 6.2.2微信通知例子

global:

resolve\_timeout: 2m

wechat\_api\_url: 'https://qyapi.weixin.qq.com/cgi-bin/'

wechat\_api\_secret: 'xxx'

wechat\_api\_corp\_id: 'xxx'

route:

group\_by: ['alertname']

group\_wait: 10s

group\_interval: 10s

repeat\_interval: 1h

receiver: 'wechat'receivers:

- name: 'wechat'

wechat\_configs:

- send\_resolved: true

to\_party: '1'

agent\_id: '1000002'

#### 6.2.3webhook

使用webhook可以灵活的配置多个receiver，但是需要额外提供服务。

global:

resolve\_timeout: 5m

route:

receiver: webhook

group\_wait: 30s

group\_interval: 5m

repeat\_interval: 4h

group\_by: [alertname]

routes:

- receiver: webhook

group\_wait: 10s

match:

team: node

receivers:

- name: webhook

webhook\_configs:

- url: 'http://xxx/hooks/wechat/'

send\_resolved: true

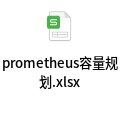
- url: 'http://xxx/hooks/mail/'

send\_resolved: true

## 容量规划

按照默认抓取metrics的间隔(15s)和保留时间(15天)，每个node上跑100个pod，可以估算prometheus需要的内存和磁盘容量。100个节点大致需要的内容为20G，硬盘容量100G。

具体的计算公式可以参考下面的excel文件。



## 附录

### Exporter

业务pod要被监控的前提是需要提供metrics接口，metrics接口代码可以和业务融合在一起，也可以利用prometheus提供的官方依赖库定义独立的exporter。

自定义的exporter可以参考<https://gitlab.oneitfarm.com/hl/myexport>

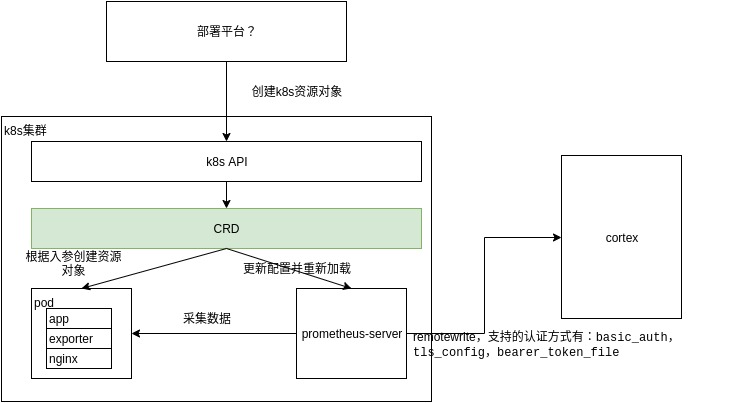
实现。

Peometheus官方以及第三方都提供了各自丰富的exporter，对于常用的中间件，官方和第三方exporter可以满足大部分需求。

独立的exporter在kubernetes以side-car的方式和业务容器部署在一个pod里。

### 拉取metrics认证

大多数exporter没有没有提供认证支持，可以借助nginx作为反向代理服务器来实现exporter的basicauth认证，在kubernetes里通过给业务pod额外注入一个nginx容器来实现。绿色CRD部分需要单独开发。



注\*：对于metrics拉取的认证需求，也可以换一种思路通过namespace的网络隔离来实现。