# Prometheus+Grafana企业监控系统课程文档

|  |  |
| --- | --- |
| 作者 | 李振良 （微信: olizhenliang） |
| 官方网站 | <http://www.ctnrs.com> |
| 课程地址 | <https://ke.qq.com/course/374130> |
| 技术博客 | <http://lizhenliang.blog.51cto.com> |
| 技术公众号 |  |

目录

[1. 聊聊监控 1](#_Toc11056654)

[1.1 为什么要监控 1](#_Toc11056655)

[1.2 怎么来监控 1](#_Toc11056656)

[1.3 准备工作 2](#_Toc11056657)

[1.3 要监控什么 2](#_Toc11056658)

[2. Prometheus概述 3](#_Toc11056659)

[2.1 Prometheus是什么 3](#_Toc11056660)

[2.2 Prometheus组成及架构 3](#_Toc11056661)

[2.3 数据模型 4](#_Toc11056662)

[2.4 指标类型 4](#_Toc11056663)

[2.5 指标和实例 5](#_Toc11056664)

[3. Prometheus部署 5](#_Toc11056665)

[3.1 二进制部署 5](#_Toc11056666)

[3.2 Docker容器部署 6](#_Toc11056667)

[3.3 访问Web 6](#_Toc11056668)

[3.4 配置Prometheus监控本身 6](#_Toc11056669)

[4. 配置文件与核心功能使用 7](#_Toc11056670)

[4.1 全局配置文件 7](#_Toc11056671)

[4.2 scrape\_configs 9](#_Toc11056672)

[4.3 relabel\_configs 11](#_Toc11056673)

[4.4 基于文件的服务发现 11](#_Toc11056674)

[5. 监控案例 12](#_Toc11056675)

[5.1 监控Linux服务器 12](#_Toc11056676)

[5.2 控CPU，内存，硬盘 12](#_Toc11056677)

[5.3 监控服务状态 13](#_Toc11056678)

[5.4 Granfana炫图展示数据 13](#_Toc11056679)

[5.5 监控Docker服务器 14](#_Toc11056680)

[5.6 监控MySQL 15](#_Toc11056681)

[6. 告警 15](#_Toc11056682)

[6.1 部署Alertmanager 16](#_Toc11056683)

[6.2 配置Prometheus与Alertmanager通信 17](#_Toc11056684)

[6.3 在Prometheus中创建告警规则 17](#_Toc11056685)

[6.4 告警状态 18](#_Toc11056686)

[6.5 告警分配 19](#_Toc11056687)

[6.6 告警收敛（分组、抑制、静默） 20](#_Toc11056688)

[6.7 Prometheus一条告警怎么触发的？ 21](#_Toc11056689)

[6.8 编写告警规则案例 22](#_Toc11056690)

[7. 监控Kubernetes资源与应用 24](#_Toc11056691)

[7.1 监控方案 24](#_Toc11056692)

[7.2 监控指标 26](#_Toc11056693)

[7.3 实现思路 27](#_Toc11056694)

[7.4 在K8S中部署Prometheus 27](#_Toc11056695)

[7.5 在K8S中部署Grafana与可视化 27](#_Toc11056696)

[7.8 监控K8S集群Node与Pod 27](#_Toc11056697)

[7.7 监控K8S资源对象 28](#_Toc11056698)

[7.8 在K8S中部署Alertmanager 29](#_Toc11056699)

# 1. 聊聊监控

## 1.1 为什么要监控

监视应用程序的当前状态是预测问题和发现生产环境中瓶颈的最有效方法之一。

监控是整个产品周期中最重要的一环，及时预警减少故障影响免扩大，而且能根据历史数据追溯问题。

* 对系统不间断实时监控
* 实时反馈系统当前状态
* 保证业务持续性运行

## 1.2 怎么来监控

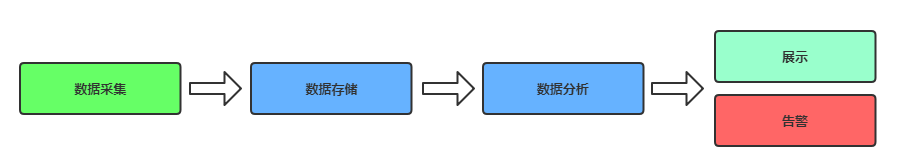
* **监控工具**
  + free
  + vmstat
  + df
  + top
  + ss
  + iftop
  + ...

适合临时性分析性能及排查故障。少量的服务器。

目前业内有很多不错的开源产品可供选择，利用开源产品很容易能做到这一点。

* **监控系统**
  + Zabbix
  + Open-Falcon
  + Prometheus

|  |  |  |  |
| --- | --- | --- | --- |
| **监控方案** | **告警** | **特点** | **适用** |
| Zabbix | Y | 大量定制工作 | 大部分的互联网公司 |
| open-falcon | Y | 功能模块分解比较细，显得更复杂 | 系统和应用监控 |
| Prometheus+Grafana | Y | 扩展性好 | 容器，应用，主机全方面监控 |



市场上主流的开源监控系统基本都是这个流程：

* 数据采集：对监控数据采集
* 数据存储：将监控数据持久化存储，用于历时查询
* 数据分析：数据按需处理，例如阈值对比、聚合
* 数据展示：Web页面展示
* 监控告警：电话，邮件，微信，短信

## 1.3 准备工作

* 熟悉被监控对象
* 整理监控指标
* 告警阈值定义
* 故障处理流程

## 1.3 要监控什么

可以从其他监控产品公司了解到要监控啥。

<https://www.jiankongbao.com> 一个技巧。

|  |  |
| --- | --- |
| 硬件监控 | 1. 通过远程控制卡：Dell的IDRAC 2. IPMI（硬件管理接口）监控物理设备。 3. 网络设备：路由器、交换机   温度，硬件故障等。 |
| 系统监控 | CPU，内存，硬盘利用率，硬件I/O，网卡流量，TCP状态，进程数 |
| 应用监控 | Nginx、Tomcat、PHP、MySQL、Redis等，业务涉及的服务都要监控起来 |
| 日志监控 | 系统日志、服务日志、访问日志、错误日志，这个现成的开源的ELK解决方案，会在下一章讲解 |
| 安全监控 | 1）可以利用Nginx+Lua实现WAF功能，并存储到ES，通过Kibana可视化展示不同的攻击类型。  2）用户登录数，passwd文件变化，其他关键文件改动 |
| API监控 | 收集API接口操作方法（GET、POST等）请求，分析负载、可用性、正确性、响应时间 |
| 业务监控 | 例如电商网站，每分钟产生多少订单、注册多少用户、多少活跃用户、推广活动效果（产生多少用户、多少利润） |
| 流量分析 | 根据流量获取用户相关信息，例如用户地理位置、某页面访问状况、页面停留时间等。监控各地区访问业务网络情况，优化用户体验和提升收益 |

# 2. Prometheus概述

## 2.1 Prometheus是什么

Prometheus（普罗米修斯）是一个最初在SoundCloud上构建的监控系统。自2012年成为社区开源项目，拥有非常活跃的开发人员和用户社区。为强调开源及独立维护，Prometheus于2016年加入云原生云计算基金会（CNCF），成为继Kubernetes之后的第二个托管项目。

https://prometheus.io

https://github.com/prometheus

**作为新一代的监控框架，Prometheus 具有以下特点：**

* 多维数据模型：由度量名称和键值对标识的时间序列数据
* PromSQL：一种灵活的查询语言，可以利用多维数据完成复杂的查询
* 不依赖分布式存储，单个服务器节点可直接工作
* 基于HTTP的pull方式采集时间序列数据
* 推送时间序列数据通过PushGateway组件支持
* 通过服务发现或静态配置发现目标
* 多种图形模式及仪表盘支持

Prometheus适用于以机器为中心的监控以及高度动态面向服务架构的监控。

## 2.2 Prometheus组成及架构

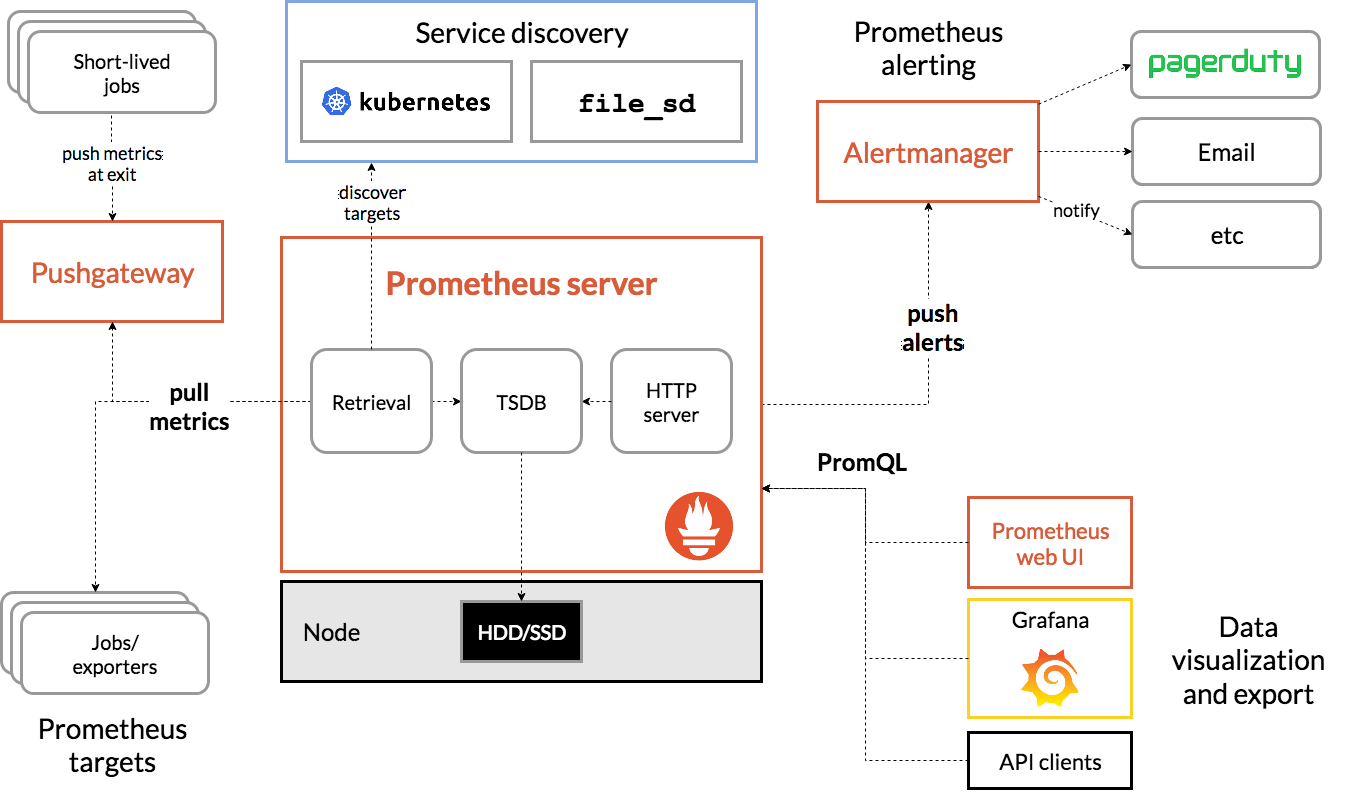
Prometheus 由多个组件组成，但是其中许多组件是可选的：

* **Prometheus Server：**用于收集指标和存储时间序列数据，并提供查询接口
* **client Library：**客户端库（例如Go，Python，Java等），为需要监控的服务产生相应的/metrics并暴露给Prometheus Server。目前已经有很多的软件原生就支持Prometheus，提供/metrics，可以直接使用。对于像操作系统已经不提供/metrics，可以使用exporter，或者自己开发exporter来提供/metrics服务。
* **push gateway：**主要用于临时性的 jobs。由于这类 jobs 存在时间较短，可能在 Prometheus 来 pull 之前就消失了。对此Jobs定时将指标push到pushgateway，再由Prometheus Server从Pushgateway上pull。

这种方式主要用于服务层面的 metrics

* **exporter：**用于暴露已有的第三方服务的 metrics 给 Prometheus。
* **alertmanager：**从 Prometheus server 端接收到 alerts 后，会进行去除重复数据，分组，并路由到对收的接受方式，发出报警。常见的接收方式有：电子邮件，pagerduty，OpsGenie, webhook 等。
* **Web UI：**Prometheus内置一个简单的Web控制台，可以查询指标，查看配置信息或者Service Discovery等，实际工作中，查看指标或者创建仪表盘通常使用Grafana，Prometheus作为Grafana的数据源；

大多数 Prometheus 组件都是用 Go 编写的，因此很容易构建和部署为静态的二进制文件。



* Prometheus Server：收集指标和存储时间序列数据，并提供查询接口
* ClientLibrary：客户端库
* Push Gateway：短期存储指标数据。主要用于临时性的任务
* Exporters：采集已有的第三方服务监控指标并暴露metrics
* Alertmanager：告警
* Web UI：简单的Web控制台

## 数据模型

Prometheus将所有数据存储为时间序列；具有相同度量名称以及标签属于同一个指标。

每个时间序列都由度量标准名称和一组键值对（也成为标签）唯一标识。

**时间序列格式：**

<metric name>{<label name>=<label value>, ...}

示例：api\_http\_requests\_total{method="POST", handler="/messages"}

## 指标类型

* Counter：递增的计数器
* Gauge：可以任意变化的数值
* Histogram：对一段时间范围内数据进行采样，并对所有数值求和与统计数量
* Summary：与Histogram类似

## 2.5 指标和实例

实例：可以抓取的目标称为实例（Instances）

作业：具有相同目标的实例集合称为作业（Job）

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090']

- job\_name: 'node'

static\_configs:

- targets: ['192.168.1.10:9090']

# 3. Prometheus部署

<https://blog.gmem.cc/prometheus-study-note>

## 3.1 二进制部署

为您的平台[下载最新版本](https://prometheus.io/download)的Prometheus，然后解压缩并运行它：

<https://prometheus.io/download/>

<https://prometheus.io/docs/prometheus/latest/getting_started/>

tar xvfz prometheus-\*.tar.gz

cd prometheus-\*

mv prometheus-\* /usr/local/prometheus

**1. 注册为服务**

[Unit]

Description=https://prometheus.io

[Service]

Restart=on-failure

ExecStart=/usr/local/prometheus/prometheus --config.file=/usr/local/prometheus/prometheus.yml

[Install]

WantedBy=multi-user.target

**2. 启动Prometheus**

./prometheus --config.file=prometheus.yml

## 3.2 Docker容器部署

<https://prometheus.io/docs/prometheus/latest/installation/>

prometheus.yml通过运行以下命令将您从主机绑定：

docker run -p 9090:9090 -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml \

prom/prometheus

或者为配置使用额外的卷：

docker run -p 9090:9090 -v /prometheus-data \

prom/prometheus --config.file=/prometheus-data/prometheus.yml

## 3.3 访问Web

http://localhost:9090访问自己的状态页面

可以通过访问localhost:9090验证Prometheus自身的指标：localhost:9090/metrics

## 3.4 配置Prometheus监控本身

Prometheus从目标机上通过http方式拉取采样点数据, 它也可以拉取自身服务数据并监控自身的健康状况

当然Prometheus服务拉取自身服务采样数据，并没有多大的用处，但是它是一个好的DEMO。保存下面的Prometheus配置，并命名为：prometheus.yml:

global:

scrape\_interval: 15s # 默认情况下，每15s拉取一次目标采样点数据。

# 我们可以附加一些指定标签到采样点度量标签列表中, 用于和第三方系统进行通信, 包括：federation, remote storage, Alertmanager

external\_labels:

monitor: 'codelab-monitor'

# 下面就是拉取自身服务采样点数据配置

scrape\_configs:

# job名称会增加到拉取到的所有采样点上，同时还有一个instance目标服务的host：port标签也会增加到采样点上

- job\_name: 'prometheus'

# 覆盖global的采样点，拉取时间间隔5s

scrape\_interval: 5s

static\_configs:

- targets: ['localhost:9090']

# 4. 配置文件与核心功能使用

配置文件的格式是YAML，使用--config.file指定配置文件的位置。本节列出重要的配置项。

## 4.1 全局配置文件

有关配置选项的完整，请参阅：<https://prometheus.io/docs/prometheus/latest/configuration/configuration/>

Prometheus以scrape\_interval规则周期性从监控目标上收集数据，然后将数据存储到本地存储上。scrape\_interval可以设定全局也可以设定单个metrics。

Prometheus以evaluation\_interval规则周期性对告警规则做计算，然后更新告警状态。evaluation\_interval只有设定在全局。

* global：全局配置
* alerting：告警配置
* rule\_files：告警规则
* scrape\_configs：配置数据源，称为target，每个target用job\_name命名。又分为静态配置和服务发现

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情（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> ... ]

## 4.2 scrape\_configs

根据配置的任务（job）以http/s周期性的收刮（scrape/pull）

指定目标（target）上的指标（metric）。目标（target）

可以以静态方式或者自动发现方式指定。Prometheus将收刮（scrape）的指标（metric）保存在本地或者远程存储上。

使用scrape\_configs定义采集目标

配置一系列的目标，以及如何抓取它们的参数。一般情况下，每个scrape\_config对应单个Job。

目标可以在scrape\_config中静态的配置，也可以使用某种服务发现机制动态发现。

# 任务名称，自动作为抓取到的指标的一个标签

job\_name: <job\_name>

# 抓取周期

[ scrape\_interval: <duration> | default = <global\_config.scrape\_interval> ]

# 每次抓取的超时

[ scrape\_timeout: <duration> | default = <global\_config.scrape\_timeout> ]

# 从目标抓取指标的URL路径

[ metrics\_path: <path> | default = /metrics ]

# 当添加标签发现指标已经有同名标签时，是否保留原有标签不覆盖

[ honor\_labels: <boolean> | default = false ]

# 抓取协议

[ scheme: <scheme> | default = http ]

# HTTP请求参数

params:

  [ <string>: [<string>, ...] ]

# 身份验证信息

basic\_auth:

  [ username: <string> ]

  [ password: <secret> ]

  [ password\_file: <string> ]

# Authorization请求头取值

[ bearer\_token: <secret> ]

# 从文件读取Authorization请求头

[ bearer\_token\_file: /path/to/bearer/token/file ]

# TLS配置

tls\_config:

  [ <tls\_config> ]

# 代理配置

[ proxy\_url: <string> ]

# DNS服务发现配置

dns\_sd\_configs:

  [ - <dns\_sd\_config> ... ]

# 文件服务发现配置

file\_sd\_configs:

  [ - <file\_sd\_config> ... ]

# K8S服务发现配置

kubernetes\_sd\_configs:

  [ - <kubernetes\_sd\_config> ... ]

# 此Job的静态配置的目标列表

static\_configs:

  [ - <static\_config> ... ]

# 目标重打标签配置

relabel\_configs:

  [ - <relabel\_config> ... ]

# 指标重打标签配置

metric\_relabel\_configs:

  [ - <relabel\_config> ... ]

# 每次抓取允许的最大样本数量，如果在指标重打标签后，样本数量仍然超过限制，则整个抓取认为失败

# 0表示不限制

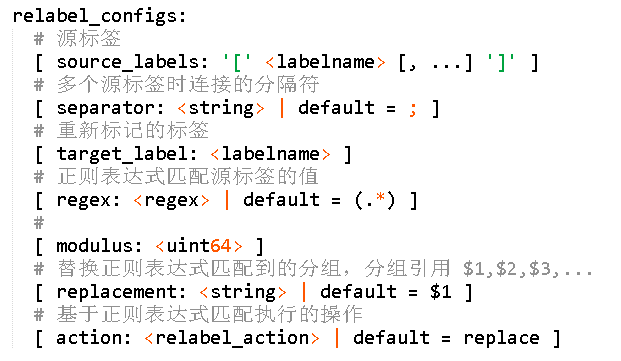
[ sample\_limit: <int> | default = 0

## 4.3 relabel\_configs

relabel\_configs ：允许在采集之前对任何目标及其标签进行修改

**重新标签的意义？**

* 重命名标签名
* 删除标签
* 过滤目标



**action：重新标签动作**

* replace：默认，通过regex匹配source\_label的值，使用replacement来引用表达式匹配的分组
* keep：删除regex与连接不匹配的目标 source\_labels
* drop：删除regex与连接匹配的目标 source\_labels
* labeldrop：删除regex匹配的标签
* labelkeep：删除regex不匹配的标签
* hashmod：设置target\_label为modulus连接的哈希值source\_labels
* labelmap：匹配regex所有标签名称。然后复制匹配标签的值进行分组，replacement分组引用（${1},${2},…）替代

## 4.4 基于文件的服务发现

**支持服务发现的来源：**

* azure\_sd\_configs
* consul\_sd\_configs
* dns\_sd\_configs
* ec2\_sd\_configs
* openstack\_sd\_configs
* file\_sd\_configs
* gce\_sd\_configs
* kubernetes\_sd\_configs
* marathon\_sd\_configs
* nerve\_sd\_configs
* serverset\_sd\_configs
* triton\_sd\_configs

Prometheus也提供了服务发现功能，可以从consul，dns，kubernetes，file等等多种来源发现新的目标。

其中最简单的是从文件发现服务。

<https://github.com/prometheus/prometheus/tree/master/discovery>

服务发现支持： endpoints，ingress，kubernetes，node，pod，service

# 5. 监控案例

## 5.1 监控Linux服务器

**node\_exporter：用于\*NIX系统监控，使用Go语言编写的收集器。**

使用文档：https://prometheus.io/docs/guides/node-exporter/

GitHub：https://github.com/prometheus/node\_exporter

exporter列表：https://prometheus.io/docs/instrumenting/exporters/

## 5.2 控CPU，内存，硬盘

收集到 node\_exporter 的数据后，我们可以使用 PromQL 进行一些业务查询和监控，下面是一些比较常见的查询。

注意：以下查询均以单个节点作为例子，如果大家想查看所有节点，将 instance="xxx" 去掉即可。

CPU使用率：

100 - (avg(irate(node\_cpu\_seconds\_total{mode="idle"}[5m])) by (instance) \* 100)

内存使用率：

100 - (node\_memory\_MemFree\_bytes+node\_memory\_Cached\_bytes+node\_memory\_Buffers\_bytes) / node\_memory\_MemTotal\_bytes \* 100

磁盘使用率：

100 - (node\_filesystem\_free\_bytes{mountpoint="/",fstype=~"ext4|xfs"} / node\_filesystem\_size\_bytes{mountpoint="/",fstype=~"ext4|xfs"} \* 100)

## 5.3 监控服务状态

使用systemd收集器：

--collector.systemd.unit-whitelist=".+" 从systemd中循环正则匹配单元  
--collector.systemd.unit-whitelist="(docker|sshd|nginx).service" 白名单，收集目标

/usr/bin/node\_exporter --collector.systemd --collector.systemd.unit-whitelist=(docker|sshd|nginx).service

node\_systemd\_unit\_state{name=“docker.service”} 只查询docker服务

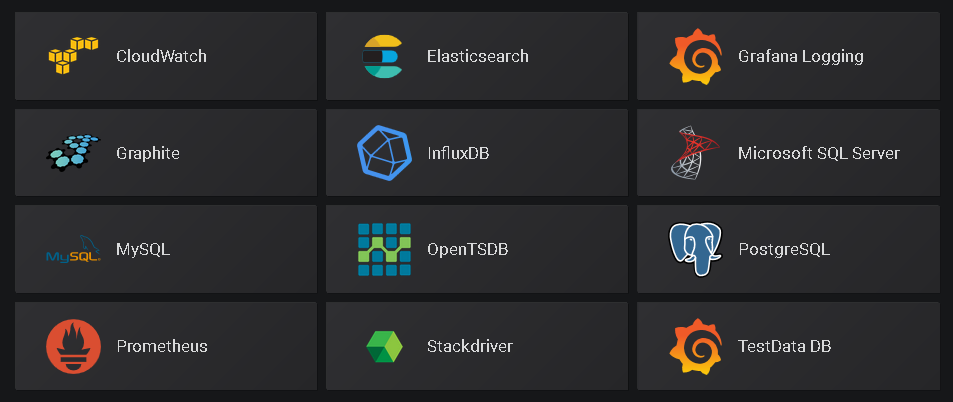
node\_systemd\_unit\_state{name=“docker.service”,state=“active”} 返回活动状态，值是1

node\_systemd\_unit\_state{name=“docker.service”} == 1 当前服务状态

## 5.4 Granfana炫图展示数据

Grafana是一个开源的度量分析和可视化系统。

https://grafana.com/grafana/download

[Grafana](http://grafana.org/)支持查询普罗米修斯。自Grafana 2.5.0（2015-10-28）以来，包含了Prometheus的Grafana数据源。  


**从Grafana.com导入预先构建的仪表板**

Grafana.com维护着[一组共享仪表板](https://grafana.com/dashboards) ，可以下载并与Grafana的独立实例一起使用。使

<https://grafana.com/dashboards/9276>



## 5.5 监控Docker服务器

cAdvisor（Container Advisor）用于收集正在运行的容器资源使用和性能信息。

https://github.com/google/cadvisor

https://grafana.com/dashboards/193

运行单个cAdvisor来监控整个Docker主机

docker run \

--volume=/:/rootfs:ro \

--volume=/var/run:/var/run:ro \

--volume=/sys:/sys:ro \

--volume=/var/lib/docker/:/var/lib/docker:ro \

--volume=/dev/disk/:/dev/disk:ro \

--publish=8080:8080 \

--detach=true \

--name=cadvisor \

google/cadvisor:latest

cAdvisor支持将统计信息导出到各种存储插件。有关更多详细信息和示例，请参阅[文档](https://github.com/google/cadvisor/blob/master/docs/storage/README.md)

cAdvisor在其端口公开Web UI：http://<hostname>:<port>/

使用Prometheus监控cAdvisor

cAdvisor将容器统计信息公开为[Prometheus](https://prometheus.io/)指标。默认情况下，这些指标在/metrics HTTP端点下提供。可以通过设置-prometheus\_endpoint命令行标志来自定义此端点。

要使用Prometheus监控cAdvisor，只需在Prometheus中配置一个或多个作业，这些作业会在该指标端点处刮取相关的cAdvisor流程。

## 5.6 监控MySQL

mysql\_exporter：用于收集MySQL性能信息。

https://github.com/prometheus/mysqld\_exporter

https://grafana.com/dashboards/7362

登录mysql为exporter创建账号：

mysql> CREATE USER 'exporter'@'localhost' IDENTIFIED BY 'XXXXXXXX' WITH MAX\_USER\_CONNECTIONS 3;

mysql> GRANT PROCESS, REPLICATION CLIENT, SELECT ON \*.\* TO 'exporter'@'localhost';

# cat .my.cnf

[client]

user=exporter

password=exporter123

# ./mysqld\_exporter --config.my-cnf=.my.cnf

- job\_name: mysql

static\_configs:

- targets:

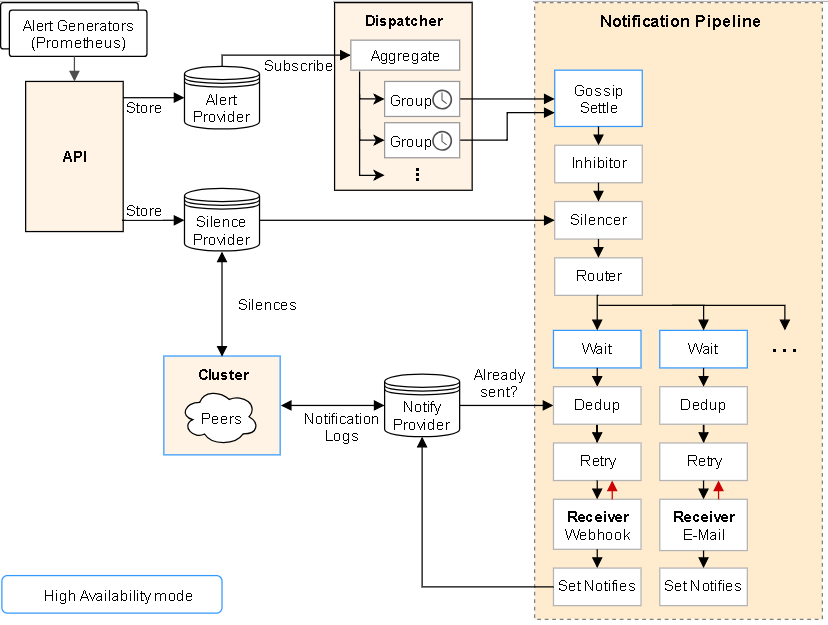
- 192.168.31.66:9104

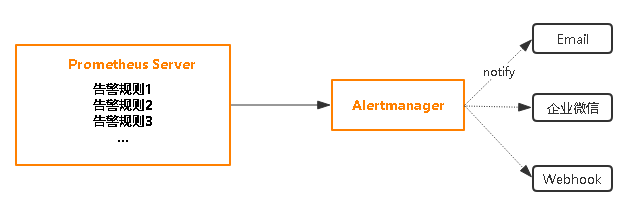
# 告警

告警无疑是监控中非常重要的环节，虽然监控数据可视化了，也非常容易观察到运行状态。但我们很难做到时刻盯着监控，所以程序来帮巡检并自动告警，这个程序是幕后英雄，保障业务稳定性就靠它了。

前面讲到过Prometheus的报警功能主要是利用Alertmanager这个组件。当Alertmanager接收到 Prometheus 端发送过来的 Alerts 时，Alertmanager 会对 Alerts 进行去重复，分组，按标签内容发送不同报警组，包括：邮件，微信，webhook。

使用prometheus进行告警分为两部分：Prometheus Server中的告警规则会向Alertmanager发送。然后，Alertmanager管理这些告警，包括进行重复数据删除，分组和路由，以及告警的静默和抑制。





## 6.1 部署Alertmanager

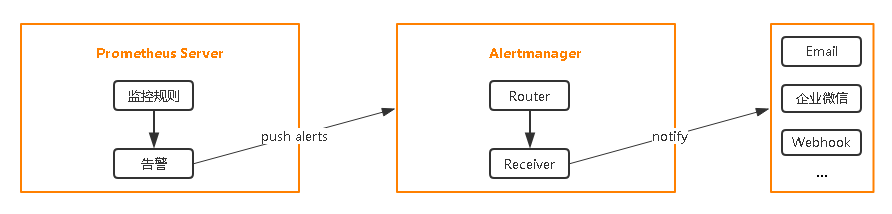
在Prometheus平台中，警报由独立的组件Alertmanager处理。通常情况下，我们首先告诉Prometheus Alertmanager所在的位置，然后在Prometheus配置中创建警报规则，最后配置Alertmanager来处理警报并发送给接收者（邮件，webhook、slack等）。

地址1：https://prometheus.io/download/

地址2：<https://github.com/prometheus/alertmanager/releases>

设置告警和通知的主要步骤如下：

1. 部署Alertmanager
2. 配置Prometheus与Alertmanager通信
3. 在Prometheus中创建告警规则



## 6.2 配置Prometheus与Alertmanager通信

# vi prometheus.yml

alerting:

alertmanagers:

- static\_configs:

- targets:

- 127.0.0.1:9093

## 6.3 在Prometheus中创建告警规则

**示例：**

# vi prometheus.yml

rule\_files:

- "rules/\*.yml"

# vi rules/node.yml

groups:

- name: example # 报警规则组名称

rules:

# 任何实例5分钟内无法访问发出告警

- alert: InstanceDown

expr: up == 0

for: 5m #持续时间 ， 表示持续一分钟获取不到信息，则触发报警

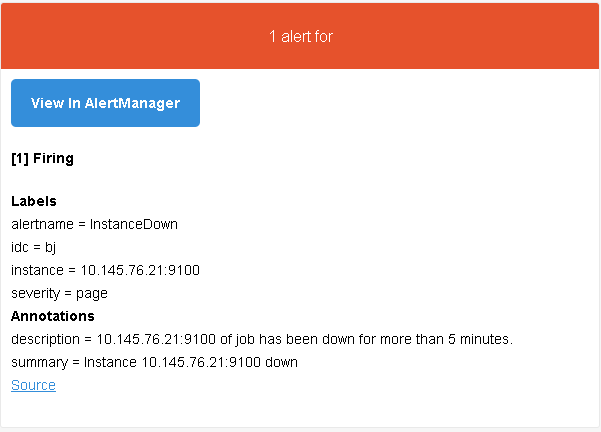
labels:

severity: page # 自定义标签

annotations:

summary: "Instance {{ $labels.instance }} down" # 自定义摘要

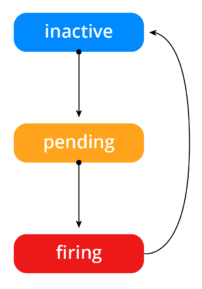
description: "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5 minutes." # 自定义具体描述



<https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/>

## 6.4 告警状态

一旦这些警报存储在Alertmanager，它们可能处于以下任何状态：



* Inactive：这里什么都没有发生。
* Pending：已触发阈值，但未满足告警持续时间（即rule中的for字段）
* Firing：已触发阈值且满足告警持续时间。警报发送到Notification Pipeline，经过处理，发送给接受者。

这样目的是多次判断失败才发告警，减少邮件。

## 6.5 告警分配

route属性用来设置报警的分发策略，它是一个树状结构，按照深度优先从左向右的顺序进行匹配。

route:

receiver: 'default-receiver'

group\_wait: 30s

group\_interval: 5m

repeat\_interval: 4h

group\_by: [cluster, alertname]

# 所有不匹配以下子路由的告警都将保留在根节点，并发送到“default-receiver”

routes:

# 所有service=mysql或者service=cassandra的告警分配到数据库接收端

- receiver: 'database-pager'

group\_wait: 10s

match\_re:

service: mysql|cassandra

# 所有带有team=frontend标签的告警都与此子路由匹配

# 它们是按产品和环境分组的，而不是集群

- receiver: 'frontend-pager'

group\_by: [product, environment]

match:

team: frontend

group\_by：报警分组依据

group\_wait：为一个组发送通知的初始等待时间，默认30s

group\_interval：在发送新告警前的等待时间。通常5m或以上

repeat\_interval：发送重复告警的周期。如果已经发送了通知，再次发送之前需要等待多长时间。通常3小时或以上

**主要处理流程：**

1. 接收到Alert，根据labels判断属于哪些Route（可存在多个Route，一个Route有多个Group，一个Group有多个Alert）。
2. 将Alert分配到Group中，没有则新建Group。
3. 新的Group等待group\_wait指定的时间（等待时可能收到同一Group的Alert），根据resolve\_timeout判断Alert是否解决，然后发送通知。
4. 已有的Group等待group\_interval指定的时间，判断Alert是否解决，当上次发送通知到现在的间隔大于repeat\_interval或者Group有更新时会发送通知。

## 告警收敛（分组、抑制、静默）

告警面临最大问题，是警报太多，相当于狼来了的形式。收件人很容易麻木，不再继续理会。关键的告警常常被淹没。在一问题中，alertmanger在一定程度上得到很好解决。

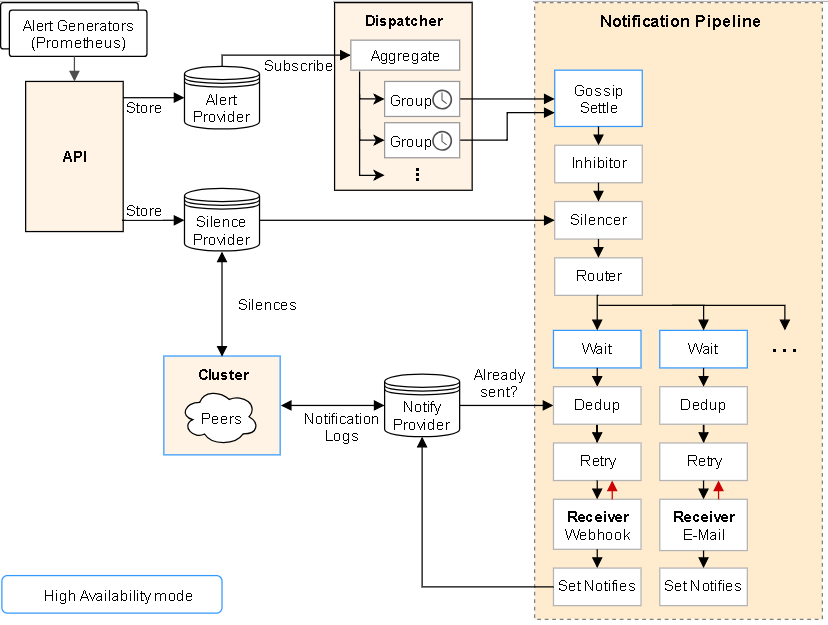
Prometheus成功的把一条告警发给了Altermanager，而Altermanager并不是简简单单的直接发送出去，这样就会导致告警信息过多，重要告警被淹没。所以需要对告警做合理的收敛。

**告警收敛手段：**

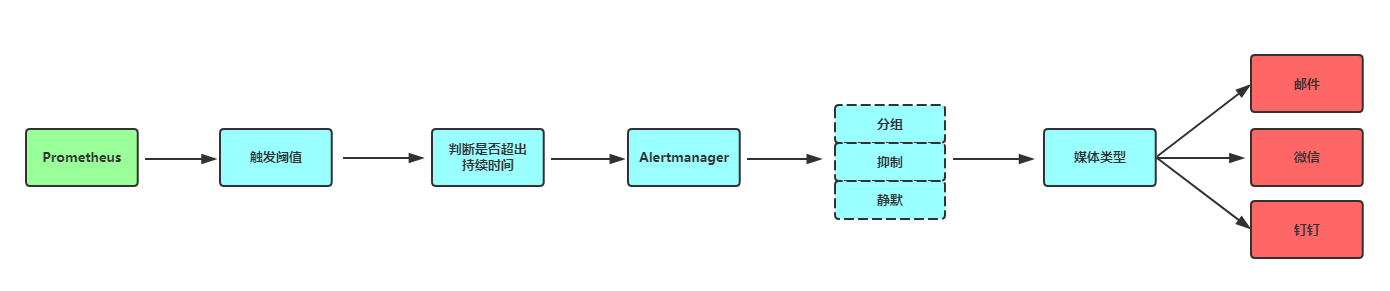
**分组（group）：**将类似性质的警报分类为单个通知

**抑制（Inhibition）：**当警报发出后，停止重复发送由此警报引发的其他警报

**静默（Silences）：**是一种简单的特定时间静音提醒的机制



## 6.7 Prometheus一条告警怎么触发的？



报警处理流程如下：

1. Prometheus Server监控目标主机上暴露的http接口（这里假设接口A），通过上述Promethes配置的'scrape\_interval'定义的时间间隔，定期采集目标主机上监控数据。
2. 当接口A不可用的时候，Server端会持续的尝试从接口中取数据，直到"scrape\_timeout"时间后停止尝试。这时候把接口的状态变为“DOWN”。
3. Prometheus同时根据配置的**"evaluation\_interval"**的时间间隔，定期（默认1min）的对**Alert Rule**进行评估；当到达评估周期的时候，发现接口A为DOWN，即UP=0为真，激活Alert，进入**“PENDING”**状态，并记录当前active的时间；
4. 当下一个alert rule的评估周期到来的时候，发现UP=0继续为真，然后判断警报**Active**的时间是否已经超出rule里的**‘for’** 持续时间，如果未超出，则进入下一个评估周期；如果时间超出，则alert的状态变为**“FIRING”**；同时调用Alertmanager接口，发送相关报警数据。
5. AlertManager收到报警数据后，会将警报信息进行分组，然后根据alertmanager配置的**“group\_wait”**时间先进行等待。等wait时间过后再发送报警信息。
6. 属于同一个Alert Group的警报，在等待的过程中可能进入新的alert，如果之前的报警已经成功发出，那么间隔**“group\_interval”**的时间间隔后再重新发送报警信息。比如配置的是邮件报警，那么同属一个group的报警信息会汇总在一个邮件里进行发送。
7. 如果Alert Group里的警报一直没发生变化并且已经成功发送，等待‘repeat\_interval’时间间隔之后再重复发送相同的报警邮件；如果之前的警报没有成功发送，则相当于触发第6条条件，则需要等待group\_interval时间间隔后重复发送。
8. 同时最后至于警报信息具体发给谁，满足什么样的条件下指定警报接收人，设置不同报警发送频率，这里有alertmanager的route路由规则进行配置。

## 6.8 编写告警规则案例

# cat rules/general.yml

groups:

- name: general.rules

rules:

- alert: InstanceDown

expr: up == 0

for: 2m

labels:

severity: error

annotations:

summary: "Instance {{ $labels.instance }} 停止工作"

description: "{{ $labels.instance }}: job {{ $labels.job }} 已经停止5分钟以上."

# cat rules/node.yml

groups:

- name: node.rules

rules:

- alert: NodeFilesystemUsage

expr: 100 - (node\_filesystem\_free\_bytes{fstype=~"ext4|xfs"} / node\_filesystem\_size\_bytes{fstype=~"ext4|xfs"} \* 100) > 80

for: 2m

labels:

severity: warning

annotations:

summary: "{{$labels.instance}}: {{$labels.mountpoint }} 分区使用过高"

description: "{{$labels.instance}}: {{$labels.mountpoint }} 分区使用大于 80% (当前值: {{ $value }})"

- alert: NodeMemoryUsage

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

for: 2m

labels:

severity: warning

annotations:

summary: "{{$labels.instance}}: 内存使用过高"

description: "{{$labels.instance}}: 内存使用大于 80% (当前值: {{ $value }})"

- alert: NodeCPUUsage

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

for: 2m

labels:

severity: warning

annotations:

summary: "{{$labels.instance}}: CPU使用过高"

description: "{{$labels.instance}}: CPU使用大于 80% (当前值: {{ $value }})"

# cat rules/reload.yml

groups:

- name: prometheus.rules

rules:

- alert: AlertmanagerReloadFailed

expr: alertmanager\_config\_last\_reload\_successful == 0

for: 10m

labels:

severity: warning

annotations:

summary: "{{$labels.instance}}: Alertmanager配置重新加载失败"

description: "{{$labels.instance}}: Alertmanager配置重新加载失败"

- alert: PrometheusReloadFailed

expr: prometheus\_config\_last\_reload\_successful == 0

for: 10m

labels:

severity: warning

annotations:

summary: "{{$labels.instance}}: Prometheus配置重新加载失败"

description: "{{$labels.instance}}: Prometheus配置重新加载失败"

# 7. 监控Kubernetes资源与应用

## 7.1 监控方案

老的监控系统无法感知这些动态创建的服务，已经不适合容器化的场景

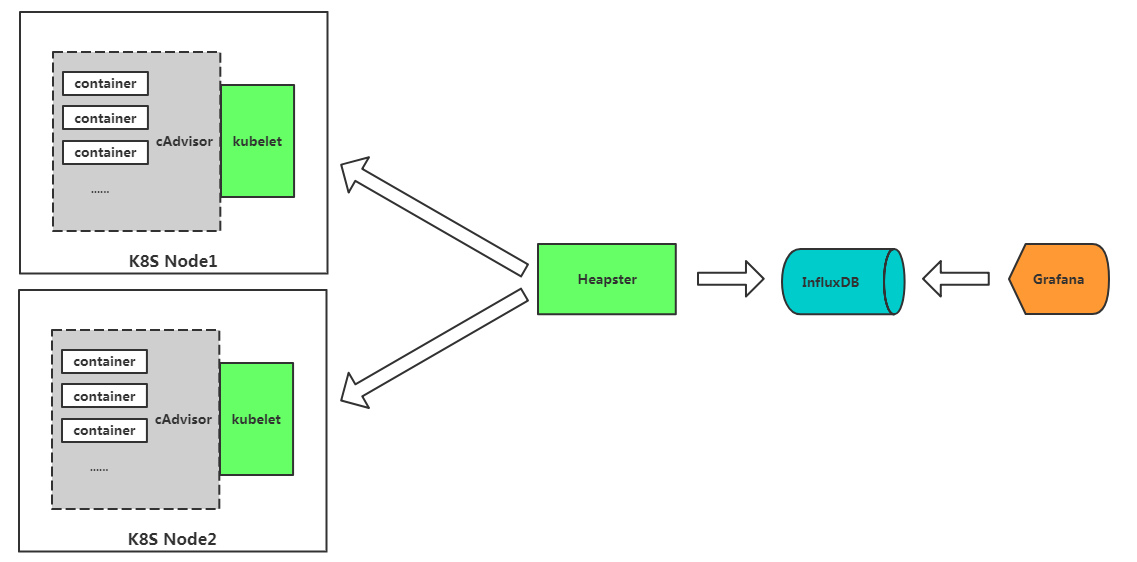
|  |  |  |  |
| --- | --- | --- | --- |
| cAdvisor+Heapster+InfluxDB+Grafana | Y | 简单 | 容器监控 |
| cAdvisor/exporter+Prometheus+Grafana | Y | 扩展性好 | 容器，应用，主机全方面监控 |

对于非容器化业务来说，像Zabbix，open-falcon已经在企业深入使用。

而Prometheus新型的监控系统的兴起来源容器领域，所以重点是放在怎么监控容器。

随着容器化大势所趋，如果传统技术不随着改变，将会被淘汰，基础架构也会发生新的技术栈组合。

**cAdvisor+InfluxDB+Grafana**



cAdvisor:是谷歌开源的一个容器监控工具，采集主机上容器相关的性能指标数据。比如CPU、内存、网络、文件系统等。

Heapster是谷歌开源的集群监控数据收集工具，会所有节点监控数据，Heapster作为一个pod在集群中运行，通过API获得集群中的所有节点，然后从节点kubelet暴露的10255汇总数据。目前社区用Metrics Server 取代 Heapster。

InfluxDB：时序数据库，存储监控数据。

Grafana：可视化展示。Grafana提供一个易于配置的仪表盘UI，可以轻松定制和扩展。

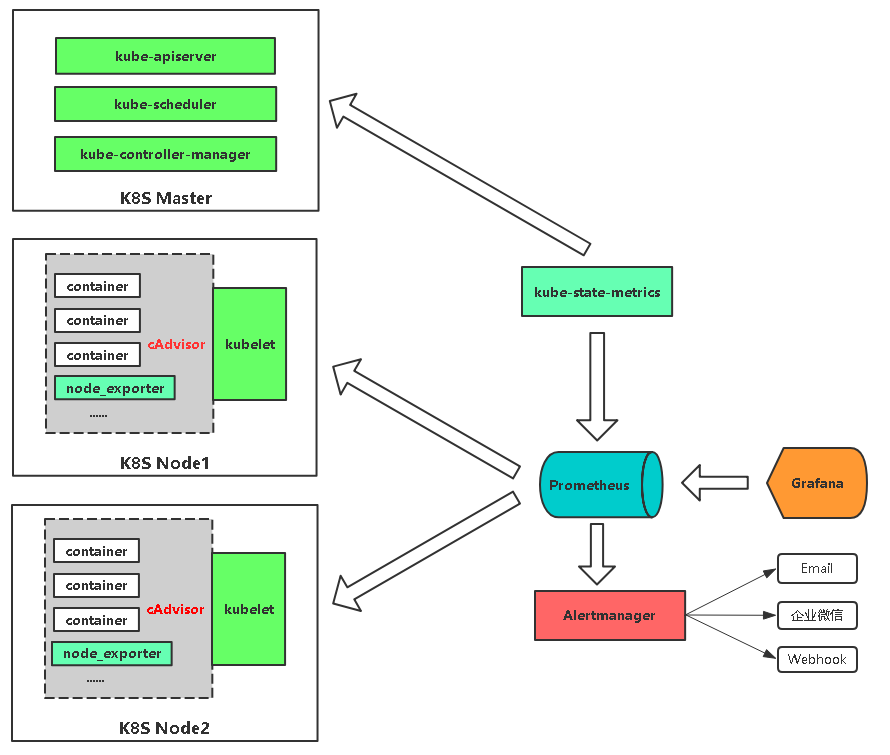
**不足**

* 功能单一，无法做到对K8S资源对象监控
* 告警单一，利用Grafana基本告警
* 依赖Influxdb，而Influxdb单点（高可用商业才支持），缺少开源精神
* heapster在1.8+弃用，metrics-server取代

**cAdvisor/exporter+Prometheus+Grafana**

Prometheus+Grafana是监控告警解决方案里的后起之秀

通过各种exporter采集不同维度的监控指标，并通过Prometheus支持的数据格式暴露出来，Prometheus定期pull数据并用Grafana展示，异常情况使用AlertManager告警。



* + 通过cadvisor采集容器、Pod相关的性能指标数据，并通过暴露的/metrics接口用prometheus抓取
  + 通过prometheus-node-exporter采集主机的性能指标数据，并通过暴露的/metrics接口用prometheus抓取
  + 应用侧自己采集容器中进程主动暴露的指标数据（暴露指标的功能由应用自己实现，并添加平台侧约定的annotation，平台侧负责根据annotation实现通过Prometheus的抓取）
  + 通过kube-state-metrics采集k8s资源对象的状态指标数据，并通过暴露的/metrics接口用prometheus抓取
  + 通过etcd、kubelet、kube-apiserver、kube-controller-manager、kube-scheduler自身暴露的/metrics获取节点上与k8s集群相关的一些特征指标数据。

## 7.2 监控指标

**Kubernetes本身监控**

* Node资源利用率
* Node数量
* Pods数量（Node）
* 资源对象状态

**Pod监控**

* Pod数量（项目）
* 容器资源利用率
* 应用程序

## 7.3 实现思路

|  |  |  |
| --- | --- | --- |
| **监控指标** | **具体实现** | **举例** |
| Pod性能 | cAdvisor | 容器CPU，内存利用率 |
| Node性能 | node-exporter | 节点CPU，内存利用率 |
| K8S资源对象 | kube-state-metrics | Pod/Deployment/Service |

## **7.4 在K8S中部署Prometheus**

https://github.com/kubernetes/kubernetes/tree/master/cluster/addons/prometheus

源码目录：kubernetes/cluster/addons/prometheus

## 7.5 在K8S中部署Grafana与可视化

grafana 是一个可视化面板，有着非常漂亮的图表和布局展示，功能齐全的度量仪表盘和图形编辑器，支持 Graphite、zabbix、InfluxDB、Prometheus、OpenTSDB、Elasticsearch 等作为数据源，比 Prometheus 自带的图表展示功能强大太多，更加灵活，有丰富的插件，功能更加强大。

https://grafana.com/grafana/download

**推荐模板：**

* 集群资源监控：3119
* 资源状态监控 ：6417
* Node监控 ：9276

## **7.8 监控K8S集群Node与Pod**

Prometheus社区提供的[NodeExporter](https://github.com/prometheus/node_exporter)项目可以对主机的关键度量指标进行监控，通过Kubernetes的DeamonSet可以在各个主机节点上部署有且仅有一个NodeExporter实例，实现对主机性能指标数据的监控，但由于容器隔离原因，使用容器NodeExporter并不能正确获取到宿主机磁盘信息，故此本课程将NodeExporter部署到宿主机。

使用文档：https://prometheus.io/docs/guides/node-exporter/

GitHub：https://github.com/prometheus/node\_exporter

exporter列表：https://prometheus.io/docs/instrumenting/exporters/

node-exporter所采集的指标主要有:

node\_cpu\_\*

node\_disk\_\*

node\_entropy\_\*

node\_filefd\_\*

node\_filesystem\_\*

node\_forks\_\*

node\_intr\_total\_\*

node\_ipvs\_\*

node\_load\_\*

node\_memory\_\*

node\_netstat\_\*

node\_network\_\*

node\_nf\_conntrack\_\*

node\_scrape\_\*

node\_sockstat\_\*

node\_time\_seconds\_\*

node\_timex \_\*

node\_xfs\_\*

目前cAdvisor集成到了kubelet组件内，可以在kubernetes集群中每个启动了kubelet的节点使用cAdvisor提供的metrics接口获取该节点所有容器相关的性能指标数据。

cAdvisor对外提供服务的默认端口为\*\*\*4194\*\*\*，主要提供两种接口:

* Prometheus格式指标接口：nodeIP:4194/metrics(或者通过kubelet暴露的cadvisor接口nodeIP:10255/metrics/cadvisor);
* WebUI界面接口：nodeIP:4194/containers/

以上接口的数据都是按prometheus的格式输出的。

## **7.7 监控K8S资源对象**

<https://github.com/kubernetes/kube-state-metrics>

kube-state-metrics是一个简单的服务，它监听Kubernetes API服务器并生成有关对象状态的指标。它不关注单个Kubernetes组件的运行状况，而是关注内部各种对象的运行状况，例如部署，节点和容器。

kube-state-metrics采集了k8s中各种资源对象的状态信息：

kube\_daemonset\_\*

kube\_deployment\_\*

kube\_job\_\*

kube\_namespace\_\*

kube\_node\_\*

kube\_persistentvolumeclaim\_\*

kube\_pod\_container\_\*

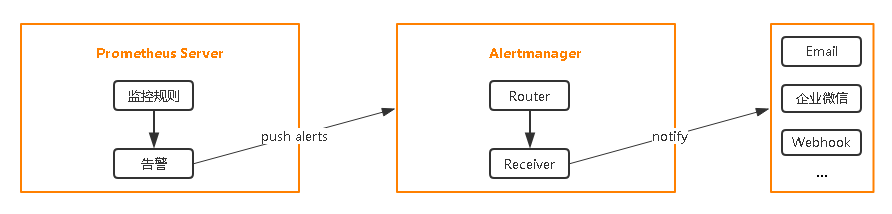
kube\_pod\_\*

kube\_replicaset\_\*

kube\_service\_\*

kube\_statefulset\_\*

## 7.8 在K8S中部署Alertmanager



设置告警和通知的主要步骤如下：

1. 部署Alertmanager
2. 配置Prometheus与Alertmanager通信
3. 配置告警
   1. prometheus指定rules目录
   2. configmap存储告警规则
   3. configmap挂载到容器rules目录
   4. 增加alertmanager告警配置