

# Prometheus 监控

臧雪园

## 1. Prometheus介绍

### 1.1 什么是监控

从技术角度看，监控是度量和管理工作系统的工具和过程。但远不止于此，监控提供从系统和应用程序生成的指标到业务价值的转换。这些指标转换为用户体验的度量，为业务提供反馈，同样还向技术提供反馈，指示业务的工作状态以及持续改进。

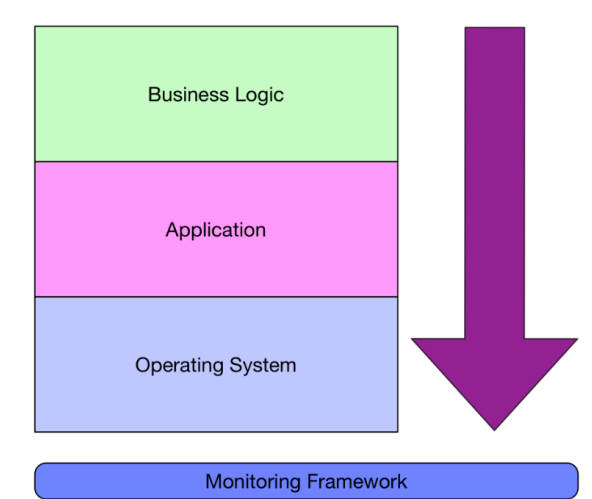
监控系统主要有两个客户：

- 技术：技术团队，开发环境等
- 业务：公司生产环境，帮助业务的持续交付等

#### (1) 监控原理

监控不应该：

- 事后监控
- 监控的不完整
- 不正确的监控
- 静态监控
- 监测不够频繁
- 没有自动化



良好的监测应提供:

- 整个世界的状态，从上到下。
- 协助故障诊断。
- 基础架构、应用程序开发和业务人员的信息源。

它应该是:

- 内置于应用程序开发和部署的设计和生命周期中。
- 在可能的情况下，自动提供自助服务。

## (2) 监控机制

应用程序监控有两种方法:

- 探测
- 自省



执行监控也有两种方法:

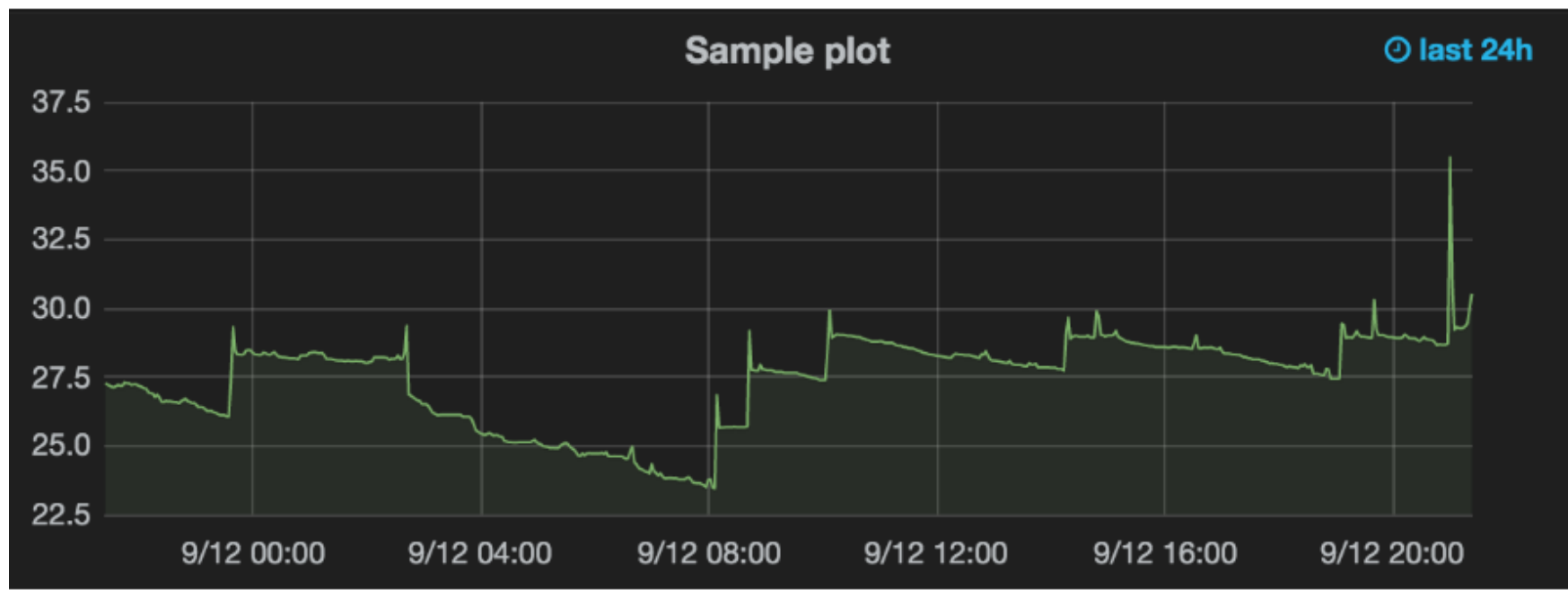
- 推
- 拉

监测数据的类型主要有两种:

- Metrics
- Logs

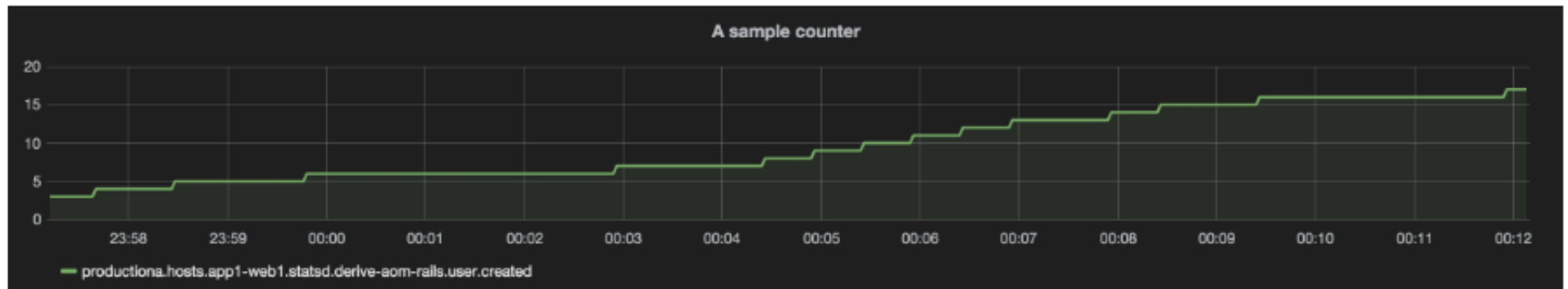
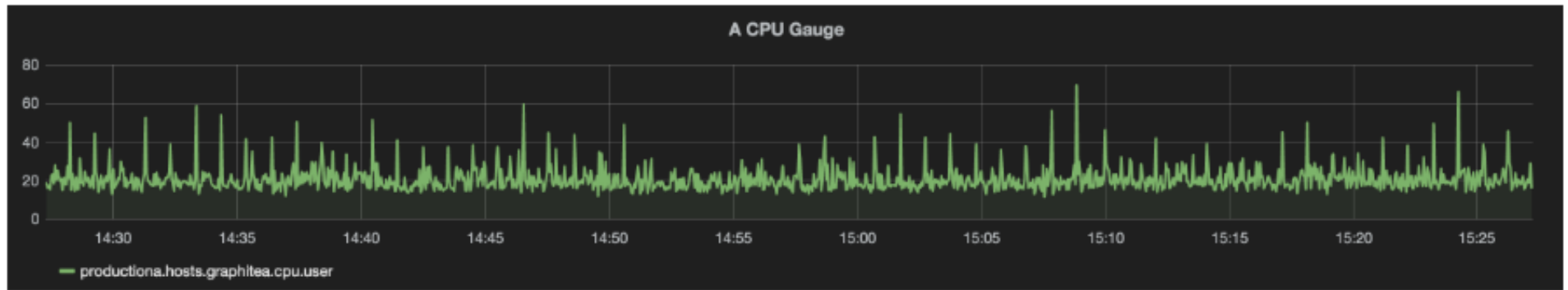
### (3) Metric(度量)

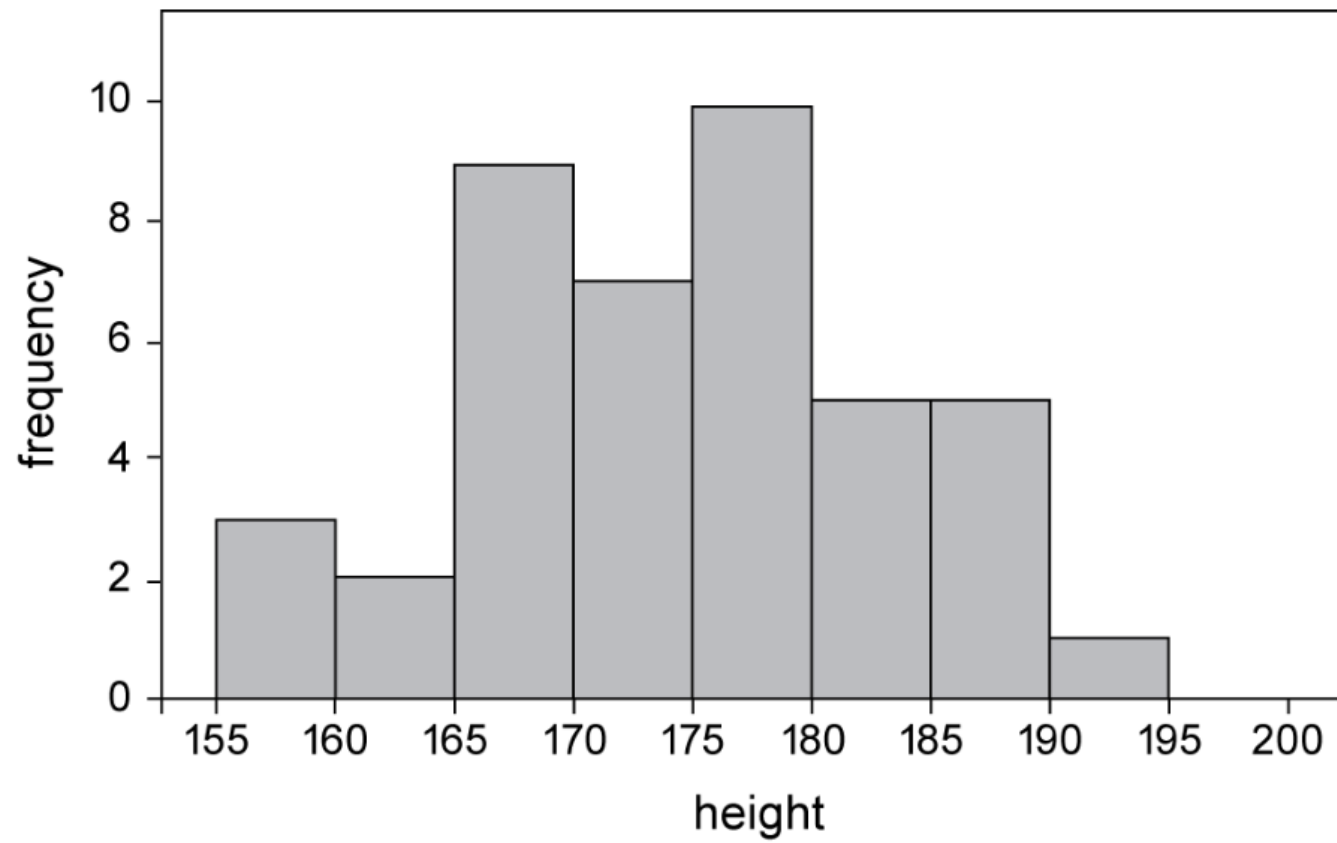
度量是对软件或硬件组件属性的度量。为了使度量有用，我们跟踪它的状态，通常随时间记录数据点。这些数据点称为观测值。观察由值、时间戳以及有时描述观察的一系列属性(如源或标记)组成。观测的集合称为**时间序列**。



## 度量的类型

- 仪表 (Gauges)
- 计数器 (Counters)
- 直方图





## 度量总结

通常，单个指标的值对我们来说并不有用。相反，度量的可视化需要对其应用数学变换。例如，我们可以将统计函数应用到我们的度量或度量组中。我们可以应用的一些常见功能包括：

- 计数或n -计数在特定时间间隔内的观测次数。
- Sum - To - Sum是在特定的时间间隔内将所有观测值相加。
- 平均值— 提供特定时间间隔内所有值的平均值。
- 中值——中值是我们的值的死中心:正好50%的值低于它，50%高于它。
- 百分比——测量一组观测中某一特定百分比下降的值。
- 标准差-表示我们度量的分布中与均值的标准差。这测量了数据集中的变化。0的标准差意味着分布等于数据的平均值。较高的偏差意味着数据分布在一系列值上。
- 变化率——变化率表示时间序列中数据之间的变化程度。

## (4) 通知和警报

### 通知和警报的区别

良好的通知和警报系统，应考虑的问题：

- 通知什么问题
- 谁来通知 altermanager
- 如何通知。 sms , email, wechart, dingding
- 通知间隔
- 什么时候停止

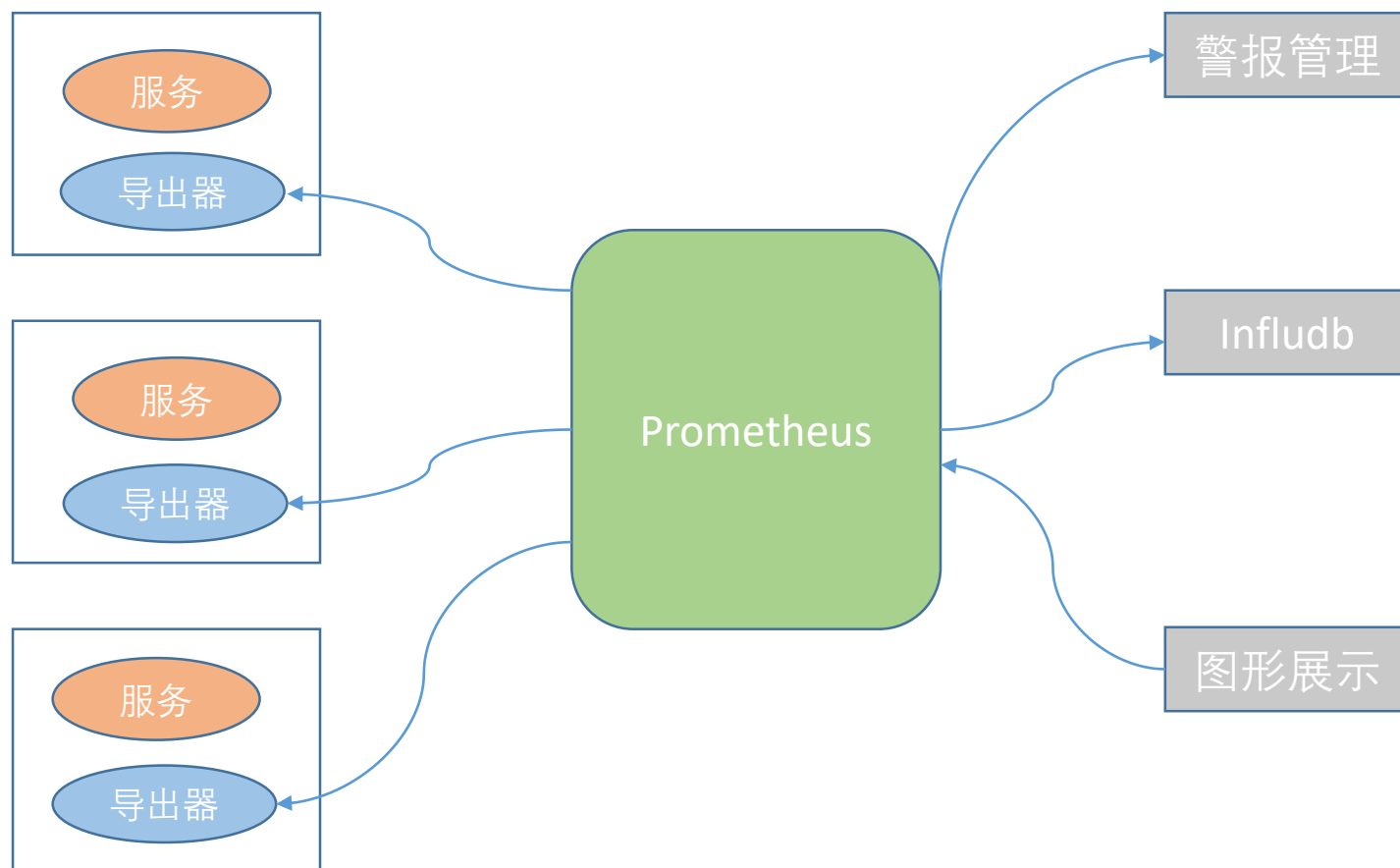


## 1.2 Prometheus 介绍

### (1) Prometheus由来

普罗米修斯的灵感来自于谷歌的Borgmon。它最初是由马特·t·普劳德(Matt T. Proud)作为一个研究项目开发的，普劳德曾是谷歌(google)的一名雇员。在普劳德加入SoundCloud之后，他与另一位工程师朱利叶斯·沃尔兹(Julius Volz)合作，认真开发普罗米修斯。其他开发人员也参与了这项工作，并继续在SoundCloud内部进行开发，最终于2015年1月公开发布。

## (2) Prometheus 架构



## METRIC 集合

为了获取端点，普罗米修斯定义了一个称为目标的配置，称为 **Scrape(刮刮)**

## 服务发现

- 用户提供的静态资源列表。
- 基于文件的发现—例如，使用配置管理工具生成在Prometheus中自动更新的资源列表。
- 自动发现

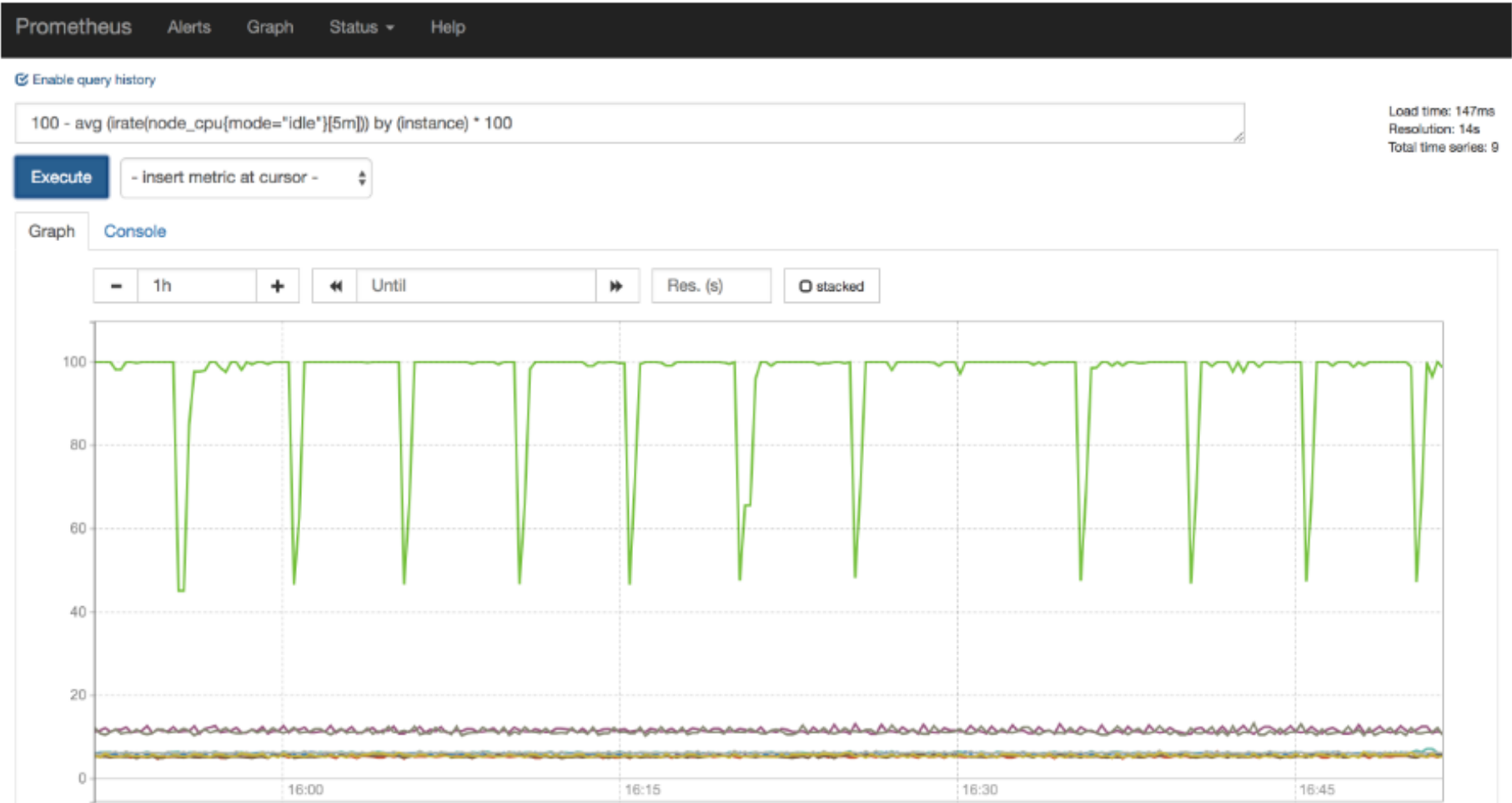
## 聚合和提醒

服务器还可以查询和聚合时间序列数据，并可以创建规则来记录常用的查询和聚合。这允许您从现有的时间序列中创建新的时间序列

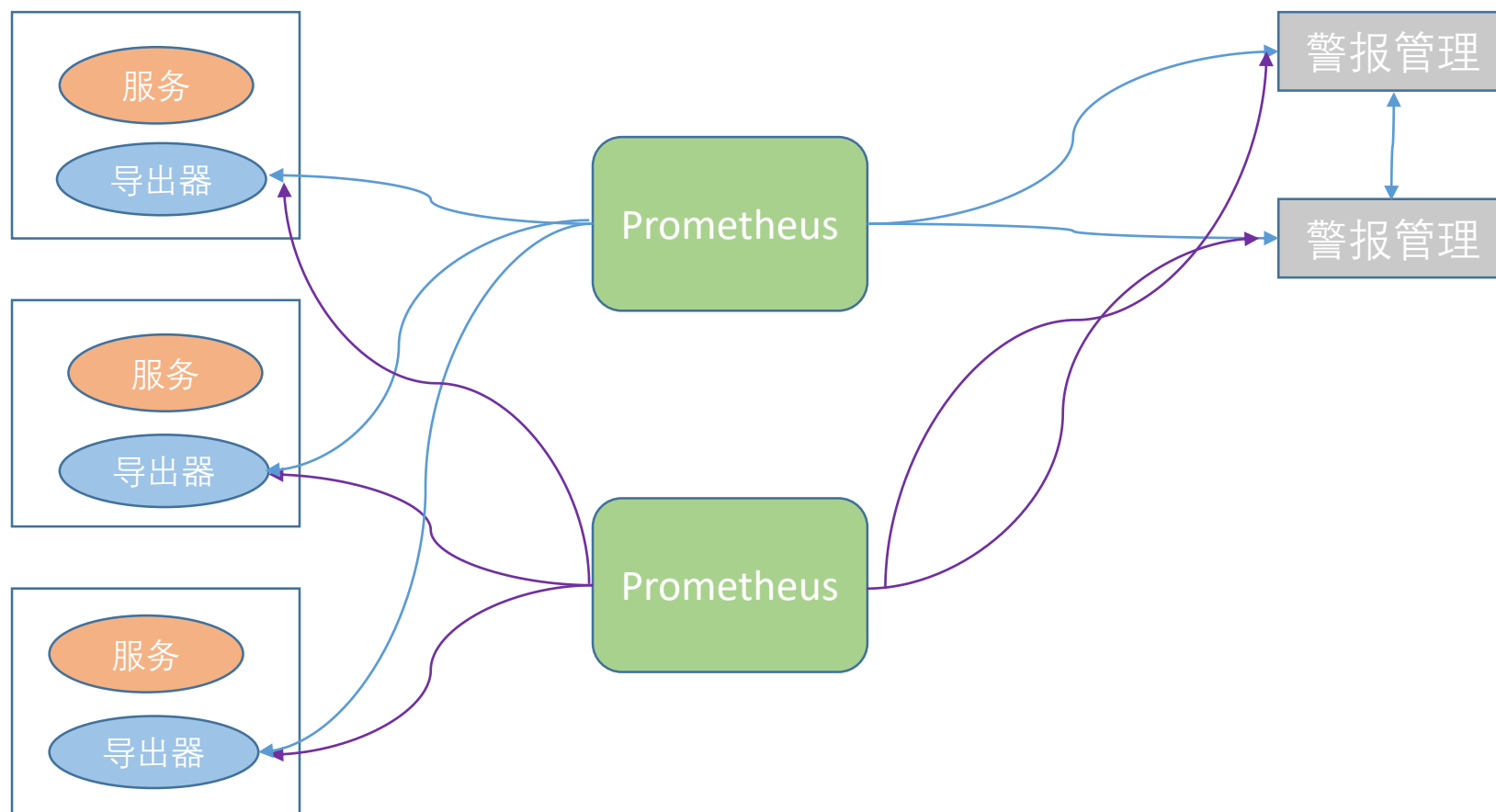
普罗米修斯也可以定义警报规则，但没有内置警报工具，而是通过外在的 Alertmanager

## 查询数据

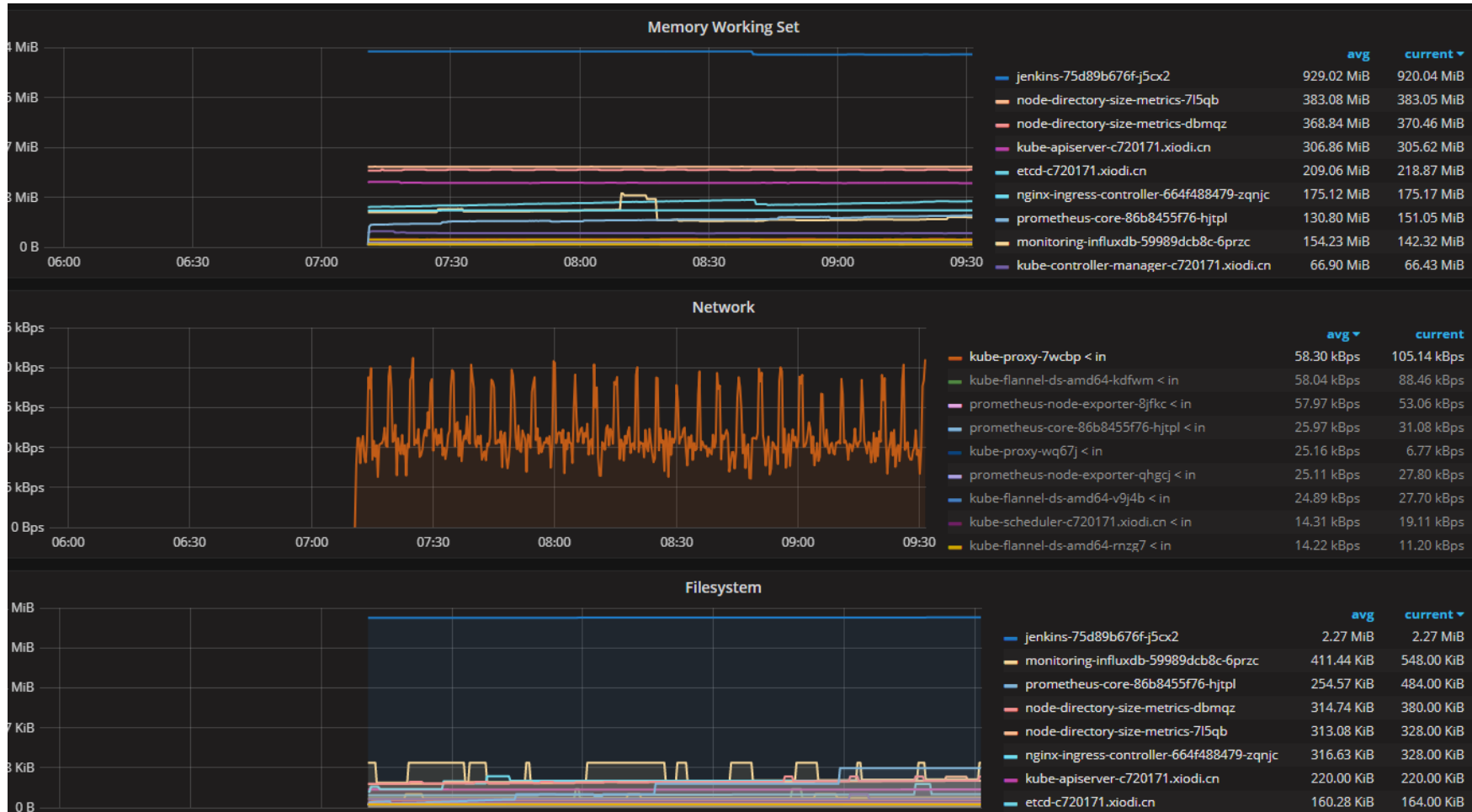
Prometheus服务器也提供了内置的查询语言PromQL;一个表达式浏览器;以及一个图形界面，您可以使用它来查看服务器上的数据。



## 冗余和高可用性



## 可视化 (Grafana)



## 1.3 Prometheus 数据及安全模型

### (1) Metric 名字

时间序列的名称通常描述收集的时间序列数据的一般性质——例如，`website_visits_total` 作为网站访问总数。名称可以包含ASCII字母、数字、下划线和冒号。

### (2) 标签 (Labels)

以 `_` 为前缀的标签名称保留给普罗米修斯内部使用。

### (3) 时间序列元素

Listing: Time series notation.

```
<time series name>{<label name>=<label value>, ...}
```

Listing: Example time series.

```
total_website_visits{site="MegaApp", location="NJ",  
instance="webserver", job="web"}
```



#### (4) Metric 保留时长

普罗米修斯是为短期监视和警报需求而设计的。默认情况下，它在本地数据库中保存了15天的时间序列。如果您希望保留更长时间的数据，建议的方法是将所需的数据发送到远程的第三方平台。普罗米修斯具有向外部数据存储写入的能力。

#### (5) 安全模型

普罗米修斯可以通过多种方式进行配置和部署。它对信任做了两个宽泛的假设：

- 不受信任的用户将能够访问普罗米修斯服务器的HTTP API，从而访问数据库中的所有数据。
- 只有受信任的用户才能访问Prometheus及其组件的命令行、配置文件、规则文件和运行时配置。

由于Prometheus 2.0, HTTP API的一些管理元素默认会被禁用。

因此，普罗米修斯及其组件不提供任何服务器端身份验证、授权或加密。如果您在一个更安全的环境中工作，您将需要实现额外的控制——例如，通过使用反向代理提前终止Prometheus服务器，或者代理您的 exporter。

## (7) Prometheus 生态系统

- prometheus
- Alertmanager
- exporters
- agent

## 2. Prometheus 安装

### 2.1 在 centos7 上安装

```
# wget https://github.com/prometheus/prometheus/releases/download/v2.5.0/prometheus-2.5.0.linux-amd64.tar.gz
# tar xf prometheus-2.5.0.linux-amd64.tar.gz
# cp prometheus-2.5.0.linux-amd64/{prometheus,promtool} /usr/local/bin/
# prometheus --version

# cd prometheus-2.5.0.linux-amd64
# cp prometheus.yml prometheus.yml.orig

# mkdir -p /etc/prometheus
# cp prometheus.yml /etc/prometheus/

# promtool check config /etc/prometheus/prometheus.yml
# prometheus --config.file "/etc/prometheus/prometheus.yml"
```

## 2.2 通过 Docker 安装

// 通过 docker 安装

```
# wget -O /etc/yum.repos.d/docker-ce.repo https://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo
```

```
# docker run -d -p 9090:9090 prom/prometheus
```

```
# docker run -d -p 9090:9090 -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml prom/prometheus
```

热配置:

```
curl -X POST http://localhost:9090/-/reload
```

参数 开启

//热载入配置,2.0之后，默认是关闭的，需要 --web.enable-lifecycle

```
kill -HUP pid
```

## 3. 监控节点和容器

### 3.1 监控节点

#### (1) 安装 Node exporter

```
# wget https://github.com/prometheus/node\_exporter/releases/download/v0.16.0/node\_exporter-0.16.0.linux-amd64.tar.gz
```

```
# tar xf node_exporter-*
```

```
# cp node_exporter-*/node_exporter /usr/local/bin/
```

```
# node_exporter --version
```

```
# node_exporter --help
```

### (3) 配置 文本文件收集器

```
# mkdir -p /var/lib/node_exporter/textfile_collector
# echo 'metadata{role="docker_server",datacenter="NJ"} 1' | sudo tee
/var/lib/node_exporter/textfile_collector/metadata.prom
```

注：文本文件收集器，默认是加载的，我们只需要指定目录 `--collector.textfile.directory=""`

### (4) 启用 system 收集器

采用 `--collector.systemd.unit-whitelist=".+"` ，使用正则表达式

### (5) 运行节点导出器

```
//# node_exporter --collector.textfile.directory /var/lib/node_exporter/textfile_collector --collector.systemd --
collector.systemd.unit-whitelist="(docker|sshd|rsyslog).service"--web.listen-address="0.0.0.0:9600" --web.telemetry-
path="/node_metrics"
```

```
# nohup node_exporter --collector.textfile.directory /var/lib/node_exporter/textfile_collector --collector.systemd --
collector.systemd.unit-whitelist="(docker|sshd|rsyslog).service" > node_exporter.out 2>&1 &
```

默认监听端口 9100

注意：两外两个node节点（172，173），同样创建文本收集器、启用systemd，然后运行 node\_exporter

不要复制粘贴

(6) 在 prometheus 服务器配置 scrap

```
# vim /etc/prometheus/prometheus.yml
```

```
.....
```

```
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'node'
    static_configs:
      - targets: ['192.168.20.172:9100', '192.168.20.173:9100', '192.168.20.174:9100']
```

open <http://192.168.20.174:9090> // 应该能看到很多 metric

(7) 在服务器上过滤收集器

```
# curl https://raw.githubusercontent.com/aishangwei/prometheus-demo/master/prometheus/prometheus.yml
```

注： 使用 params 参数配合 collect， 过滤想要的数

```
# curl -g -X GET http://192.168.20.172:9100/metrics?collect[]=cpu
```

## 3.2 监控 docker

### (1) 安装 cadvisor

```
# curl https://raw.githubusercontent.com/aishangwei/prometheus-demo/master/docker/cadvisor\_run.sh
```

```
# docker run \  
  --volume=/:/rootfs:ro \  
  --volume=/var/run:/var/run:rw \  
  --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
```

```
curl http://192.168.20.172:8080
```

```
curl http://192.168.20.172:8080/metrics
```



## (2) 配置 scrap

# curl <https://raw.githubusercontent.com/aishangwei/prometheus-demo/master/prometheus/prometheus2.yml>

```
- systemd
- job_name: 'docker'
  static_configs:
    - targets: ['192.168.20.172:8080', '192.168.20.173:8080', '192.168.20.174:8080']
```

curl http://192.168.20.174:9090

## 3.3 标签的配置使用

### (1) 重新贴标签

考虑到要明智地使用标签，我们为什么要给事物重新命名呢？一句话：控制。在一个集中的、复杂的监视环境中，您有时无法控制您正在监视的所有资源以及它们公开的监视数据。重新标记允许您在您的环境中控制、管理和潜在地标准化度量。一些最常见的用例是：

- 删除不必要的指标。
- 从指标中删除敏感或不需要的标签。
- 添加、编辑或修改指标的标签值或标签格式。

记住有两个阶段我们可以重新命名。第一阶段是重新标记来自服务发现的目标。这对于将来自服务发现的元数据标签的信息应用到您的度量上的标签非常有用。这是在作业内部的`relabel_configs`块中完成的。

第二个阶段是在刮刮（`scape`）之后，但在保存到存储系统之前。这使我们能够确定我们保存了哪些指标，删除了哪些指标，以及这些指标将是什么样子。这是在我们的工作中的`metric_relabel_configs`块中完成的。

记住这两个阶段最简单的方法是：`relabel_configs`发生在刮刮之前，`metric_relabel_configs` 发生在刮刮之后。

## (2) 删除 metrics

```
# curl https://raw.githubusercontent.com/aishangwei/prometheus-demo/master/prometheus/prometheus3.yml
```

```
# curl https://raw.githubusercontent.com/aishangwei/prometheus-demo/master/prometheus/prometheus4.yml // 如果是多个标签，可以使用逗号隔开
```

注意：

[go 正则表达式的库 RegExp使用](#)  
[RE 表达式语法](#)

[表达式测试1](#)

[表达式测试2](#)

(container\_tasks\_state|container\_memory\_failures\_total)  
匹配

- container\_tasks\_state
- container\_memory\_failures\_total

如果我们指定了多个源标签，需要用 ; 分开：  
regex1;regex2;regex3

### (3) 更换标签的值

```
- job_name: 'docker'
  static_configs:
    - targets: ['192.168.20.172:8080', '192.168.20.173:8080', '192.168.20.174:8080']
  metric_relabel_configs:
    - source_labels: [id]
      regex: '/kubepods/([a-z0-9]+);'
      replacement: '$1'
      target_label: container_id
```

注意：kubepods 根据你的实际情况修改

### (4) 删除标签

```
- job_name: 'docker'
  static_configs:
    - targets: ['192.168.20.172:8080', '192.168.20.173:8080', '192.168.20.174:8080']
  metric_relabel_configs:
    - regex: 'kernelVersion'
      action: labeldrop
```

注意：还有一个反向操作 labelkeep

参考地址：[https://prometheus.io/docs/prometheus/latest/configuration/configuration/#%3Cmetric\\_relabel\\_config%3E](https://prometheus.io/docs/prometheus/latest/configuration/configuration/#%3Cmetric_relabel_config%3E)

← → ↻ ⓘ 不安全 | 192.168.20.174:9090/graph?g0.range\_input=1h&g0.expr=cadvisor\_version\_info&g0.tab=1

应用 Company edu Good\_Web Network Others Servers temp Book

Prometheus Alerts Graph Status ▾ Help

☐ Enable query history

cadvisor\_version\_info

Execute

- insert metric at cursor - ▾

Graph

Console

Element

Value

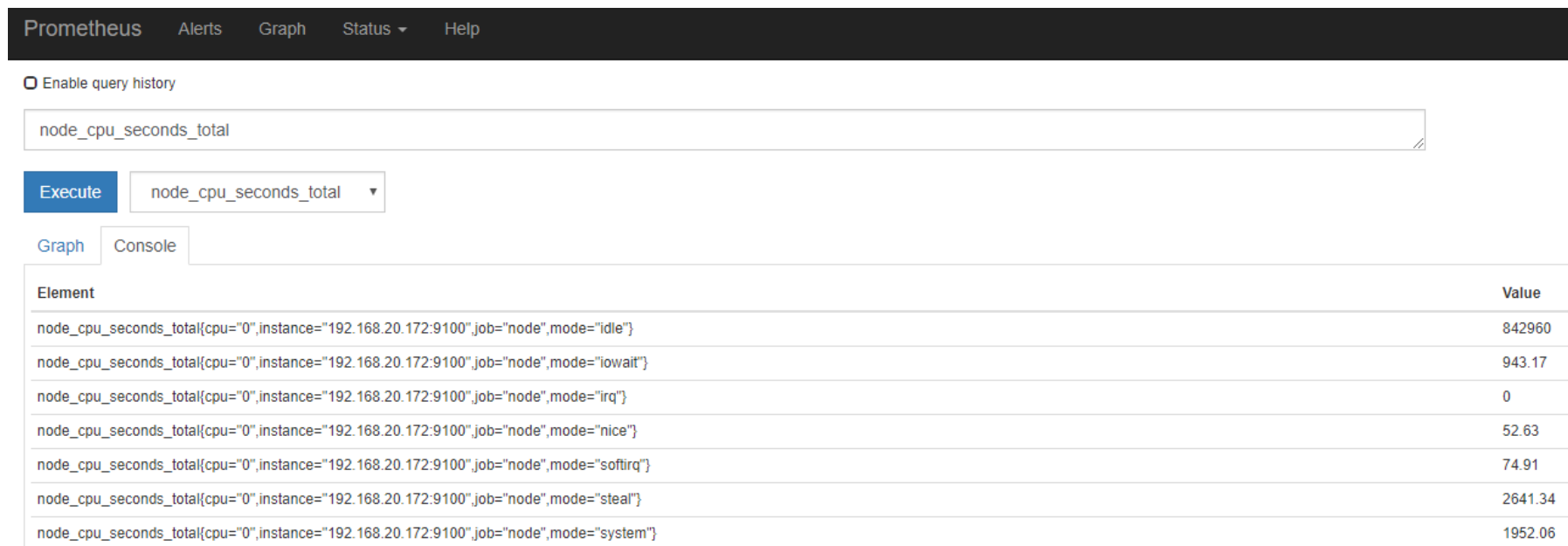
no data

Add Graph

## 3.4 三位一体和使用方法

### (1) CPU利用率

从指标下拉菜单中选择node\_cpu\_seconds\_total并单击Execute。



The screenshot shows the Prometheus web interface. At the top, there's a navigation bar with 'Prometheus', 'Alerts', 'Graph', 'Status', and 'Help'. Below this, there's a checkbox for 'Enable query history'. The main query input field contains 'node\_cpu\_seconds\_total'. To the left of the input is an 'Execute' button, and to the right is a dropdown menu showing 'node\_cpu\_seconds\_total'. Below the input field, there are two tabs: 'Graph' and 'Console'. The 'Console' tab is active, displaying a table of results.

Element	Value
node_cpu_seconds_total{cpu="0",instance="192.168.20.172:9100",job="node",mode="idle"}	842960
node_cpu_seconds_total{cpu="0",instance="192.168.20.172:9100",job="node",mode="iowait"}	943.17
node_cpu_seconds_total{cpu="0",instance="192.168.20.172:9100",job="node",mode="irq"}	0
node_cpu_seconds_total{cpu="0",instance="192.168.20.172:9100",job="node",mode="nice"}	52.63
node_cpu_seconds_total{cpu="0",instance="192.168.20.172:9100",job="node",mode="softirq"}	74.91
node_cpu_seconds_total{cpu="0",instance="192.168.20.172:9100",job="node",mode="steal"}	2641.34
node_cpu_seconds_total{cpu="0",instance="192.168.20.172:9100",job="node",mode="system"}	1952.06

我们从计算每个CPU模式的每秒速率开始。PromQL有一个名为irate的函数，用于计算距离向量中时间序列的每秒瞬时增长率。让我们在node\_cpu\_seconds\_total度量上使用irate函数。在查询框中输入：

```
irate(node_cpu_seconds_total{job="node"}[5m])
```

```
avg(irate(node_cpu_seconds_total{job="node"}[5m])) by (instance)
```

现在，我们将irate函数封装在avg聚合中，并添加了一个by子句，该子句通过实例标签聚合。这将产生三个新的指标，使用来自所有CPU和所有模式的值来平均主机的CPU使用情况。

```
avg (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) by (instance) * 100
```

在这里，我们查询中添加了一个值为idle的mode标签。这只查询空闲数据。我们通过实例求出结果的平均值，并将其乘以100。现在我们在每台主机上都有5分钟内空闲使用的平均百分比。我们可以把这个变成百分数用这个值减去100，就像这样：

```
100 - avg (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) by (instance) * 100
```

现在我们有三个指标，每个主机一个指标，显示5分钟窗口内使用的平均CPU百分比。

☐ Enable query history

```
100 - avg (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) by (instance) * 100
```

Execute

node\_cpu\_seconds\_total ▼

Graph

Console

Element	Value
{instance="192.168.20.172:9100"}	3.533333333255723
{instance="192.168.20.173:9100"}	3.033333333247967
{instance="192.168.20.174:9100"}	0.724999999802094

```
100 - avg (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) by (instance) * 100
```

Load time: 45m  
Resolution: 14s  
Total time series:

Execute

node\_cpu\_seconds\_total ▼

Graph Console

1h +<< Until >> Res. (s) ☐ stacked



■ {instance="192.168.20.174:9100"}  
■ {instance="192.168.20.173:9100"}  
■ {instance="192.168.20.172:9100"}



## (2) CPU Saturation(饱和度)

获取主机上CPU饱和的一种方法是跟踪负载平均，即考虑主机上的CPU数量，在一段时间内平均运行队列长度。平均少于cpu的数量通常是正常的；

要查看主机的平均负载，我们可以使用node\_load\*指标来实现这些功能。它们显示平均负荷超过1分钟，5分钟和15分钟。我们将使用一分钟的平均负载:node\_load1。

我们还需要计算主机上的cpu数量。我们可以这样使用[count聚合](#)：

```
count by (instance)(node_cpu_seconds_total{mode="idle"})
```

☐ Enable query history

count by (instance)(node\_cpu\_seconds\_total{mode="idle"})

Execute

node\_cpu\_seconds\_total ▼

Graph

Console

Element

Value

{instance="192.168.20.173:9100"}

8

{instance="192.168.20.174:9100"}

8

{instance="192.168.20.172:9100"}

8

```
node_load1 > on (instance)count by (instance)(node_cpu_seconds_total{mode="idle"})
```

### (3) 内存使用

以node\_memory为前缀的指标列表中找到它们。

☐ Enable query history

node\_memory\_MemTotal\_bytes

Execute

node\_memory\_MemTotal\_ ▼

Graph

Console

Element	Value
node_memory_MemTotal_bytes{instance="192.168.20.172:9100",job="node"}	8005124096
node_memory_MemTotal_bytes{instance="192.168.20.173:9100",job="node"}	3974131712
node_memory_MemTotal_bytes{instance="192.168.20.174:9100",job="node"}	3974131712

我们将关注node\_memory度量的一个子集，以提供我们的利用率度量：

- node\_memory\_MemTotal\_bytes - 主机上的总内存
- node\_memory\_MemFree\_bytes - 主机上的空闲内存
- node\_memory\_Buffers\_bytes - 缓冲区缓存中的内存
- node\_memory\_Cached\_bytes - 页面缓存中的内存。

所有这些指标都以字节表示。

// 计算使用内存的百分比

```
(node_memory_MemTotal_bytes - (node_memory_MemFree_bytes + node_memory_Cached_bytes +  
node_memory_Buffers_bytes)) / node_memory_MemTotal_bytes * 100
```

## (5) 监控硬盘

对于磁盘，我们只度量磁盘使用情况，而不是利用率、饱和或错误。这是因为在大多数情况下，它是可视化和警报最有用的数据。节点导出器的磁盘使用指标位于以`node_filesystem`为前缀的指标列表中。

☐ Enable query history

Load time: 63m:  
Resolution: 14s  
Total time series

Execute

node\_filesystem\_size\_byt ▼

Graph

Console

Element	Value
node_filesystem_size_bytes{device="/dev/mapper/centos-root",fstype="xfs",instance="192.168.20.172:9100",job="node",mountpoint="/"}	376883814
node_filesystem_size_bytes{device="/dev/mapper/centos-root",fstype="xfs",instance="192.168.20.173:9100",job="node",mountpoint="/"}	376883814
node_filesystem_size_bytes{device="/dev/mapper/centos-root",fstype="xfs",instance="192.168.20.174:9100",job="node",mountpoint="/"}	376883814
node_filesystem_size_bytes{device="/dev/vda1",fstype="xfs",instance="192.168.20.172:9100",job="node",mountpoint="/boot"}	106325606

`node_filesystem_size_bytes`度量显示正在被监控的每个文件系统挂载的大小。我们可以使用类似于内存度量的查询来生成主机上使用的磁盘空间百分比。

$$\frac{(\text{node\_filesystem\_size\_bytes}\{\text{mountpoint="/"}\} - \text{node\_filesystem\_free\_bytes}\{\text{mountpoint="/"}\})}{\text{node\_filesystem\_size\_bytes}\{\text{mountpoint="/"}\}} * 100$$

不过，与内存度量不同的是，我们在每个主机上的每个挂载点都有文件系统度量。因此我们添加了挂载点标签，特别是`/`。这将在被监视的每个主机上返回该文件系统的磁盘使用度量。

☐ Enable query history

```
(node_filesystem_size_bytes{mountpoint="/" } - node_filesystem_free_bytes{mountpoint="/" }) / node_filesystem_size_bytes{mountpoint="/" } * 100
```

Execute

node\_filesystem\_size\_byt ▾

Graph

Console

Element	Value
{device="/dev/mapper/centos-root",fstype="xfs",instance="192.168.20.172:9100",job="node",mountpoint="/"}	25.98980683634261
{device="/dev/mapper/centos-root",fstype="xfs",instance="192.168.20.173:9100",job="node",mountpoint="/"}	28.170137475662315
{device="/dev/mapper/centos-root",fstype="xfs",instance="192.168.20.174:9100",job="node",mountpoint="/"}	29.26993190610204
{device="rootfs",fstype="rootfs",instance="192.168.20.172:9100",job="node",mountpoint="/"}	25.98980683634261
{device="rootfs",fstype="rootfs",instance="192.168.20.173:9100",job="node",mountpoint="/"}	28.170137475662315
{device="rootfs",fstype="rootfs",instance="192.168.20.174:9100",job="node",mountpoint="/"}	29.26993190610204

如果我们想要或者需要，我们现在可以为配置的特定挂载点添加额外的查询。要监视一个称为/data的挂载点，我们将使用：

```
(node_filesystem_size_bytes{mountpoint="/data"} - node_filesystem_free_bytes{mountpoint="/data"}) /
node_filesystem_size_bytes{mountpoint="/data"} * 100
```

或者我们可以使用正则表达式来匹配多个挂载点。

```
(node_filesystem_size_bytes{mountpoint=~"/|/run"} - node_filesystem_free_bytes{mountpoint=~"/|/run"}) /
node_filesystem_size_bytes{mountpoint=~"/|/run"} * 100
```

可以看到，我们已经更新了挂载点标签，将操作符从=改为=~，这告诉Prometheus右手边的值将是一个正则表达式。然后我们匹配了/run和/ 文件系统。

**注意：**

- (1) 不能使用匹配空字符串的正则表达式。
- (2) 对于不匹配的正则表达式，还有一个!~运算符。

## 预计多长时间磁盘爆满

```
predict_linear(node_filesystem_free_bytes{mountpoint="/" }[1h], 4*3600) < 0
```

```
predict_linear(node_filesystem_free_bytes{job="node"}[1h], 4*3600) < 0
```

// 一旦小于就会返回一个负数，然后触发报警 [predict\\_linear 函数](#)

## (5) 监控服务状态

systemd收集器的数据，我们这里只收集了 Docker SSH 和 Rsyslog

Load time: 3  
Resolution: '  
Total time se

Execute

Graph Console

Element	Value
node_systemd_unit_state{instance="192.168.20.172:9100",job="node",name="docker.service",state="activating"}	0
node_systemd_unit_state{instance="192.168.20.172:9100",job="node",name="docker.service",state="active"}	1
node_systemd_unit_state{instance="192.168.20.172:9100",job="node",name="docker.service",state="deactivating"}	0
node_systemd_unit_state{instance="192.168.20.172:9100",job="node",name="docker.service",state="failed"}	0
node_systemd_unit_state{instance="192.168.20.172:9100",job="node",name="docker.service",state="inactive"}	0

`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

`node_systemd_unit_state{name="docker.service"} == 1` // 返回当前服务的状态

**注：**比较二进制运算符:==。这将检索所有值为1、名称标签为docker.service的指标。

## (6) up metric

监视特定节点状态的另一个有用指标：up，如果实例是健康的，度量就被设置为 1，失败返回 - 或 0

☐ Enable query history

Execute

up

Graph

Console

Element	Value
up{instance="192.168.20.172:8080",job="docker"}	1
up{instance="192.168.20.172:9100",job="node"}	1
up{instance="192.168.20.173:8080",job="docker"}	0
up{instance="192.168.20.173:9100",job="node"}	1
up{instance="192.168.20.174:8080",job="docker"}	0
up{instance="192.168.20.174:9100",job="node"}	1
up{instance="localhost:9090",job="prometheus"}	1

许多 exporter 都有专门的指标来识别服务的最后一次成功的 scrape

比如 cAdvisor **container\_last\_seen**

MySQL **mysqlup**



## (7) Metadata metric

最后，让我们看看使用节点导出器的Textfile收集器创建的指标元数据。

```
metadata{role="docker_server",datacenter="NJ"} 1
```

```
metadata{datacenter != "NJ"}
```

[算术和比较二进制运算符](#)

### 3.5 Vector matches （向量匹配）

有两种向量匹配:一对一和多对一(或一对多)。

一对一匹配

```
node_systemd_unit_state{name="docker.service"} == 1 and on (instance, job) metadata{datacenter="BJ"}
```

多对一和一对多匹配是指“one”一侧的每个向量元素可以与“many”一侧的多个元素匹配。使用group\_left或group\_right修饰符显式地指定这些匹配，其中left或right决定了哪个向量具有更高的基数

参考地址：<https://prometheus.io/docs/prometheus/latest/querying/operators/#vector-matching>

## 3.6 持久查询

到目前为止，我们只是在表达式浏览器中运行查询。虽然查看该查询的输出很有趣，但结果被卡在普罗米修斯服务器上，是暂时的。有三种方法可以使我们的持久查询（不用每次都要输入查询规则）：

- 记录规则 — 从查询中创建新的指标。
- 警报规则 - 从查询生成警报。
- 可视化 — 使用像Grafana这样的仪表盘来可视化查询。

### (1) 记录规则

记录规则是一种计算新时间序列的方法，特别是从输入的时间序列中聚合的时间序列。我们可以这样做：

- 跨多个时间序列生成聚合。
- 预计算昂贵的查询，即消耗大量时间或计算能力的查询。
- 生成一个时间序列，我们可以用它来生成警报。

### (2) 配置记录规则

记录规则存储在Prometheus服务器上，存储在Prometheus服务器加载的文件中。规则是自动计算的，频率由prometheus.yml全局块中的evaluation\_interval参数控制。

```
# mkdir -p rules
# cd rules
# touch node_rules.yml
```

// 在 prometheus.yml 文件中包含

```
rule_files:
  - "rules/node_rules.yml"
```

### (3) 添加记录规则

命名规则建议：

- level:metric:operations

例如： instance: node\_cpu:avg\_rate5m

```
groups:
- name: node_rules
  interval: 10s
  rules:
  - record: instance:node_cpu:avg_rate5m
    expr: 100 - avg (irate(node_cpu_seconds_total{job="node",mode="idle"}[5m])) by (instance) * 100
    labels:
      metric_type: aggregation
```

☐ Enable query history

instance:node\_cpu:avg\_rate5m

Execute

instance:node\_cpu:avg\_rate5m ▾

Graph

Console

Element	Value
instance:node_cpu:avg_rate5m(instance="192.168.20.172:9100",metric_type="aggregation")	4.183333333348855
instance:node_cpu:avg_rate5m(instance="192.168.20.173:9100",metric_type="aggregation")	2.2833333329375023
instance:node_cpu:avg_rate5m(instance="192.168.20.174:9100",metric_type="aggregation")	2.383333332246838

groups:

- name: node\_rules

rules:

- record: instance:node\_cpu:avg\_rate5m

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

- record: instance:node\_memory\_usage:percentage

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

- record: instance:root:node\_filesystem\_usage:percentage

expr: (node\_filesystem\_size\_bytes{mountpoint="/" } - node\_filesystem\_free\_bytes{mountpoint="/" }) / node\_filesystem\_size\_bytes{mountpoint="/" } \* 100

### 3.7 可视化

需要注意的是，普罗米修斯一般不用于长期数据保留——默认值是15天的时间序列。这意味着，普罗米修斯关注的是更直接的监控问题，而不是可视化和仪表板更重要的其他系统。使用表达式浏览器、绘制Prometheus用户界面内部的图形以及构建相应的警报，这些方法往往比构建广泛的仪表盘更能体现普罗米修斯时间序列数据的实用性。

Grafana支持在Linux、Microsoft Windows和Mac OS x上运行。


#### (1) 在 centos 上安装

```
# wget https://s3-us-west-2.amazonaws.com/grafana-releases/release/grafana-5.3.2-1.x86\_64.rpm
```

```
# yum -y localinstall grafana-5.3.2-1.x86_64.rpm
```

```
# systemctl start grafana-server && systemctl enable grafana-server
```

```
open http://192.168.20.174:3000
```





## Data Sources / Prometheus



Type: Prometheus

Settings

Dashboards


Name	Prometheus		Default	<input checked="" type="checkbox"/>
Type	Prometheus			

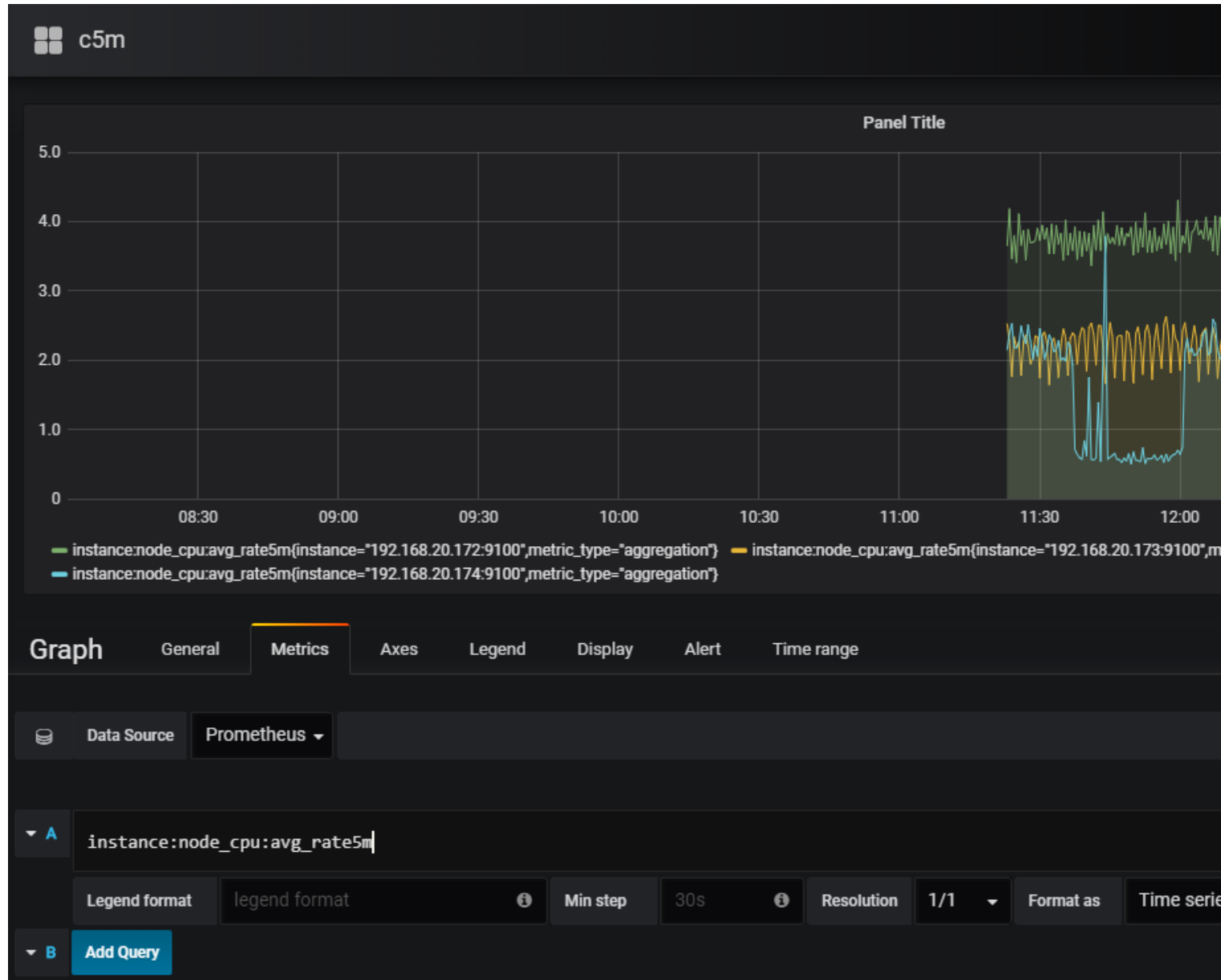
### HTTP

URL	http://192.168.20.174:9090	
Access	Browser	

Help ▶

### Auth

Basic Auth	<input type="checkbox"/>	With Credentials		<input type="checkbox"/>
------------	--------------------------	------------------	---	--------------------------





## 4. 服务发现

手动配置监控目标显然不适合我们大批量监控节点，特别是kubernetes

prometheus 通过服务发现解决这个问题，自动机制来检测，分类和识别新的和变更的目标。可以通过下面三种方式：

- 通过配置管理工具填充的文件接收目标列表。
- 查询API(如Amazon AWS API)以获取目标列表。
- 使用DNS记录返回目标列表。

### 4.1 基于文件发现

基于文件的发现只是比静态配置更高级的一小步，但是它对于配置管理工具的配置非常有用。通过基于文件的发现，普罗米修斯使用文件中指定的目标。这些文件通常由另一个系统生成，例如配置管理系统，如Puppet、Ansible，或者从另一个源(如CMDB)查询。定期运行脚本或查询，或触发它们(重新)填充这些文件。然后，普罗米修斯按照指定的时间表从这些文件中重新加载目标。

这些文件可以是YAML或JSON格式，并包含定义的目标列表，就像我们在静态配置中定义它们一样。

```
# cd /etc/prometheus
# mkdir -pv targets/{nodes,docker}

# cat /etc/prometheus/prometheus.yml
.....
scrape_configs:
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']
  - job_name: 'node'
    file_sd_configs:
      - files:
        - targets/nodes/*.json
        refresh_interval: 5m
  - job_name: docker
    file_sd_configs:
      - files:
        - targets/docker/*.json
        refresh_interval: 5m
```

注意：也可以使用该参数：prometheus\_sd\_file\_mtime\_seconds 最近一次更新 发现文件的时间

```
# cat targets/nodes/nodes.json
```

```
[{  
  "targets": [  
    "192.168.20.174:9100",  
    "192.168.20.175:9100",  
    "192.168.20.176:9100"  
  ]  
}]
```

```
# cat targets/docker/daemons.json
```

```
[{  
  "targets": [  
    "192.168.20.172:8080",  
    "192.168.20.173:8080",  
    "192.168.20.174:8080"  
  ],  
  "labels": {  
    "datacenter": "nj"  
  }  
}]
```

```
// 也可以使用下面的这种方式 (YAML)
# cat /etc/prometheus/targets/nodes/nodes.json
- targets:
  - "192.168.20.172:8080"
  - "192.168.20.173:8080"
  - "192.168.20.174:8080"
```

## 4.2 DNS服务发现

DNS服务发现依赖于查询A、AAAA或SRV DNS记录。

# 基于 SRV 记录发现

scrape\_configs:

- job\_name: 'webapp'

  - dns\_sd\_configs:

    - names: ['\_prometheus.\_tcp.xiodi.cn']

**注意：** \_prometheus 为服务名称， \_tcp 为协议， xiodi.cn 为域名

# 基于 A 记录

- job\_name: 'webapp'

  - dns\_sd\_configs:

    - names: ['c720174.xiodi.cn']

      - type: A

      - port: 9090

**\_prometheus 属性** ? x

服务位置(SRV) 安全

域(M): xiodi.cn

服务(S): \_prometheus

协议(P): \_tcp

优先级(O): 0

权重(W): 0

端口号(N): 9090

提供此服务的主机(H):  
c720174.xiodi.cn

确定 取消 应用(A) 帮助

## 5. 告警和告警管理

### 5.1 告警的介绍

在前面当中，我们已经安装、配置并使用Prometheus了。现在，我们需要了解如何从监视数据生成有用的警报。普罗米修斯是一个划分的平台，度量的收集和存储与警报是分开的。警报由称为Alertmanager的工具提供，这是监视环境的独立部分。警报规则在Prometheus服务器上定义。这些规则可以触发事件，然后将其传播到Alertmanager。Alertmanager随后决定如何处理各自的警报，处理复制之类的问题，并决定在发送警报时使用什么机制:实时消息、电子邮件或其它工具。

常见的反人类模式设计：

警报方法中最常见的反模式是发送太多警报。太多的警报相当于监控“喊狼来了的男孩”。收件人将变得麻木，对警告和不理会他们。关键的警报常常被淹没在不重要的更新的洪流中。

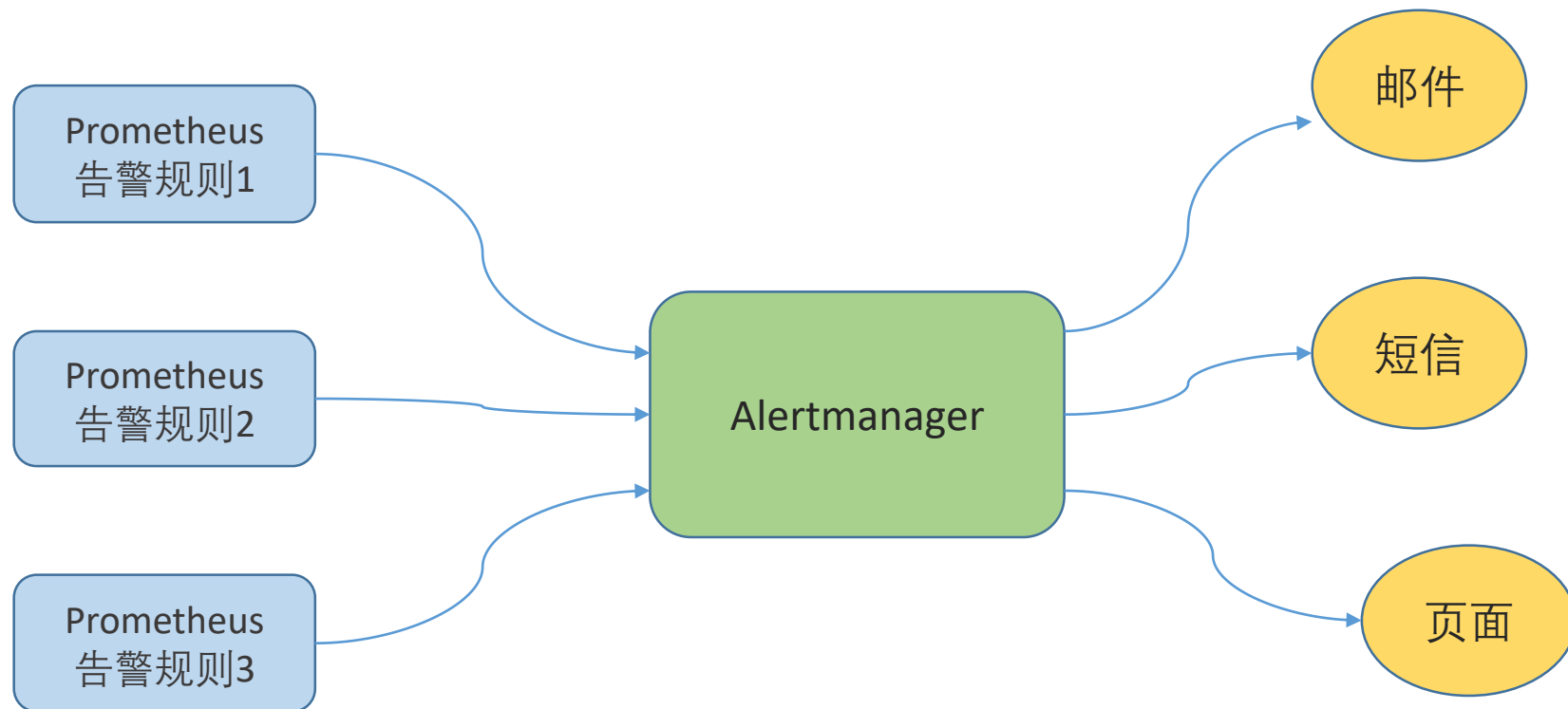
第二个最常见的反模式是警告的错误分类。

第三个最常见的反模式是发送无用的警告

良好的警示有一些关键特征：

- 嘈杂的提醒会导致警觉疲劳，最终，警告会被忽略。
- 应该设置正确的警报优先级。如果警报是紧急的，那么应该将其快速路由到负责响应的一方。如果警报不是紧急的，我们应该以适当的速度发送它，以便在需要时作出响应。
- 警报应该包含适当的上下文，使它们立即有用。

## Alertmanager 介绍





## 5.2 配置Alertmanager告警

### (1) 安装 Alertmanager

下载地址：

地址1: <https://prometheus.io/download/#alertmanager>

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

```
# wget https://github.com/prometheus/alertmanager/releases/download/v0.15.2/alertmanager-0.15.2.linux-amd64.tar.gz
```

```
# tar xf alertmanager-0.15.2.linux-amd64.tar.gz
```

```
# cp alertmanager-0.15.2.linux-amd64/{alertmanager,amtool} /usr/local/bin/
```

```
# alertmanager --version
```

### (2) 配置 Alertmanager

```
# mkdir -pv /etc/alertmanager
```

```
# cat /etc/alertmanager/alertmanager.yml
```

global:

```
smtp_smarthost: 'smtp.126.com:25'  
smtp_from: 'xxxxxx@126.com'  
smtp_auth_username: 'xxxxxx@126.com'  
smtp_auth_password: 'xxxxxx'  
smtp_require_tls: false
```

route:

```
receiver: mail
```

receivers:

```
- name: 'mail'  
  email_configs:  
    - to: '756686600@qq.com'
```

[邮件设置参考](#)  
[webhook接收器](#)

### (3) 启动 alertmanager

```
# alertmanager --config.file alertmanager.yml
```

open <http://192.168.20.174:9093>

### (4) 在 Prometheus 上添加 Alertmanager

.....

```
alerting:
```

```
  alertmanagers:
```

```
  - static_configs:
```

```
    - targets:
```

```
      - 192.168.20.174:9093
```

.....

### (5) 在 prometheus 上添加对 alertmanager 的监控

.....

```
  - job_name: 'alertmanager'
```

```
    static_configs:
```

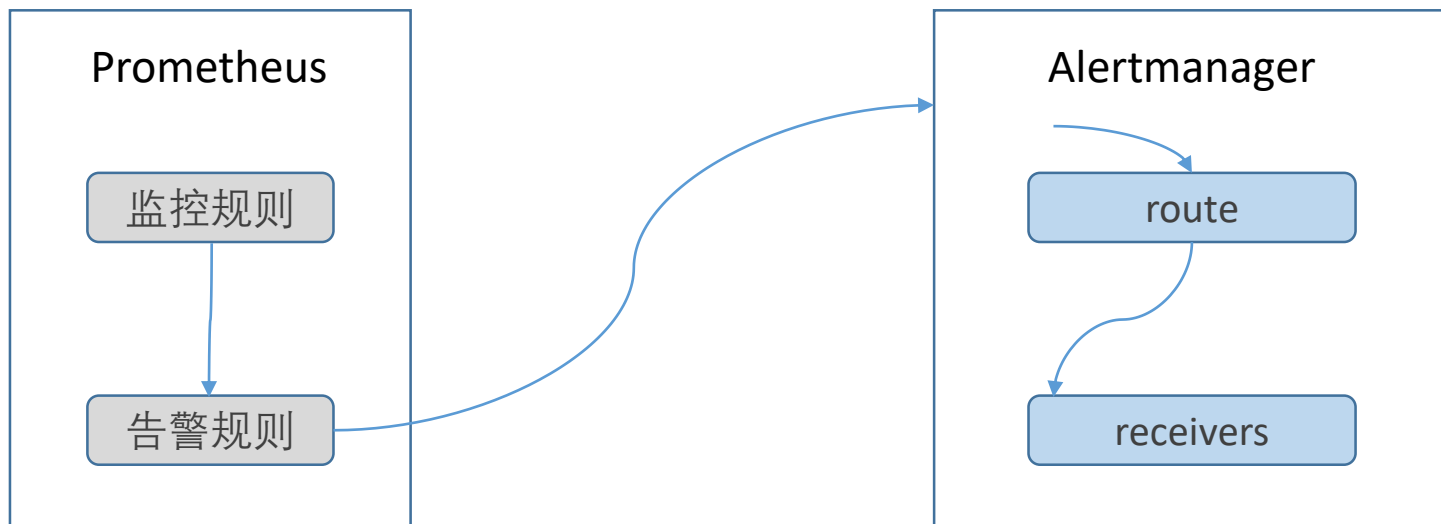
```
      - targets: ['192.168.20.174:9093']
```

## (6) 在 prometheus 添加告警规则

```
# vim /etc/prometheus/rules/node_alerts.yml
groups:
- name: node_alerts
  rules:
  - alert: HighNodeCPU
    expr: instance:node_cpu:avg_rate5m > 4
    for: 2m
    labels:
      severity: warning
    annotations:
      summary: High Node CPU for 1 hour
      console: Thank you Test
```

## (7) 把告警规则加入 prometheus 配置文件

```
# cd /etc/prometheus/rules
# vim node_alertes.yml
.....
rule_files:
- "rules/*_rules.yml"
- "rules/*_alerts.yml"
.....
```



1 alert for

[View In AlertManager](#)

**[1] Firing**

**Labels**

alertname = HighNodeCPU

instance = 192.168.20.172:9100

metric\_type = aggregation

severity = warning

**Annotations**

console = Thank you Test

summary = High Node CPU for 1 hour

[Source](#)

## 5.3 添加新的告警

### (1) 添加新的告警规则

```
# vim /etc/prometheus/rules/node_alertes.yml
groups:
- name: node_alerts
  rules:
.....
- alert: DiskWillFillIn4Hours
  expr: predict_linear(node_filesystem_free_bytes{mountpoint="/"}[1h], 4*3600) < 0
  for: 5m
  labels:
    severity: critical
  annotations:
    summary: Disk on {{ $labels.instance }} will fill in approximately 4 hours.
- alert: InstanceDown
  expr: up{job="node"} == 0
  for: 1m
  labels:
    severity: critical
  annotations:
    summary: Host {{ $labels.instance }} of {{ $labels.job }} is Down!
```

## 5.4 添加 Prometheus 告警

```
# vim rules/prometheus_alertes.yml
```

```
groups:
```

```
- name: prometheus_alerts
```

```
  rules:
```

```
    - alert: PrometheusConfigReloadFailed
```

```
      expr: prometheus_config_last_reload_successful == 0
```

```
      for: 1m
```

```
      labels:
```

```
        severity: warning
```

```
      annotations:
```

```
        description: Reloading Prometheus configuration has failed on {{ $labels.instance }}.
```

```
    - alert: PrometheusNotConnectedToAlertmanagers
```

```
      expr: prometheus_notifications_alertmanagers_discovered < 2
```

```
      for: 1m
```

```
      labels:
```

```
        severity: warning
```

```
      annotations:
```

```
        description: Prometheus {{ $labels.instance }} is not connected to any Alertmanagers
```



## //添加 systemd 服务告警

```
# vi rules/service_alertes.yml
groups:
- name: service_alerts
  rules:
  - alert: NodeServiceDown
    expr: node_systemd_unit_state{state="active"} != 1
    for: 10s
    labels:
      severity: critical
    annotations:
      summary: Service {{ $labels.name }} on {{ $labels.instance }} is no longer active!
      description: 监控中心向您报告： - "挨踢的，您的服务挂了？"
```

## 5.5 Altermanager 路由

```
# vim /etc/altermanager/altermanager.yml
```

```
route:
```

```
  group_by: ['instance']
```

```
  group_wait: 30s
```

```
  group_interval: 5m
```

```
  repeat_interval: 3h
```

```
  receiver: email
```

```
  routes:
```

```
    - match:
```

```
      severity: critical
```

```
      receiver: pager
```

```
    - match_re:
```

```
      severity: ^(warning|critical)$
```

```
      receiver: support_team
```

```
receivers:
```

```
- name: 'email'
```

```
  email_configs:
```

```
    - to: '756686600@qq.com'
```

```
- name: 'support_team'
```

```
  email_configs:
```

```
    - to: '995595198@qq.com'
```

```
- name: 'pager'
```

```
  email_configs:
```

```
    - to: 'alert-pager@example.com'
```

group\_by: 根据 label(标签)进行匹配，如果是多个，就要多个都匹配  
group\_wait: 30s 等待该组的报警，看有没有一起合伙搭车的  
group\_interval: 5m 下一次报警开车时间  
repeat\_interval: 3h 重复报警时间

注意：

新报警报警时间：上一次报警之后的 group\_interval 时间

重复的报警，下次报警时间为：group\_interval + repeat\_interval

## 5.6 通过 Alertmanager 控制静默

### (1) 通过 UI 创建静默

Alertmanager Alerts Silences Status

## New Silence

Start

2018-11-09T07:24:33.368Z ✓

Duration

5m ✓

End

2018-11-09T07:29:33.368Z ✓

Matchers Alerts affected by this silence.

Name

severity ✓

+

Value

critical

☐ Regex

Creator

testone ✓

Comment

aishangwei ✓

Preview Alerts

Create

Reset

# Silence

[Edit](#)[Expire](#)

ID 23d9dd15-bb48-49d7-91b5-92a000fc16f0

Starts at 2018-11-09 07:33:56

Ends at 2018-11-09 09:33:06

Updated at 2018-11-09 07:33:56

Created by testone

Comment aishangwei test

State active

Matchers `severity="critical"`

Affected alerts

**Silenced alerts: 1**

- `state=active` `severity=critical` `name=rsyslog.service` `job=node` `instance=192.168.20.172:9100`  
`alertname=NodeServiceDown`

## (2) 通过命令行创建静默

```
// # amtool --alertmanager.url=http://192.168.20.174:9093 silence add  
alertname=InstancesGone service=application1 -c "test1"  
# amtool --alertmanager.url=http://192.168.20.174:9093 silence add severity=critical -c "test2"
```

注意：默认创建的静默时间为 1h，也可以手工指定，匹配两个标签，然后会返回一个静默 ID

```
# amtool --alertmanager.url=http://192.168.20.174:9093 silence query    // 查询当前静默列表  
  
# amtool --alertmanager.url=http://localhost:9093 silence expire 784ac68d-33ce-4e9b-8b95-  
431a1e0fc268    // 该静默失效
```

注意：amtool 如果不指定 --alertmanager，默认会在 \$HOME/.config/amtool/config.yml 或 /etc/amtools/config.yml 查询

```
// config.yml  
alertmanager.url: "http://192.168.20.174:9093"  
author: test@example.com  
comment_required: true
```

```
# amtool silence add --comment "App1 maintenance" alertname=~'Instance.*' service=application1  
  
# amtool silence add --author "test" --duration "2h" alertname=InstancesGone service=application1
```







