

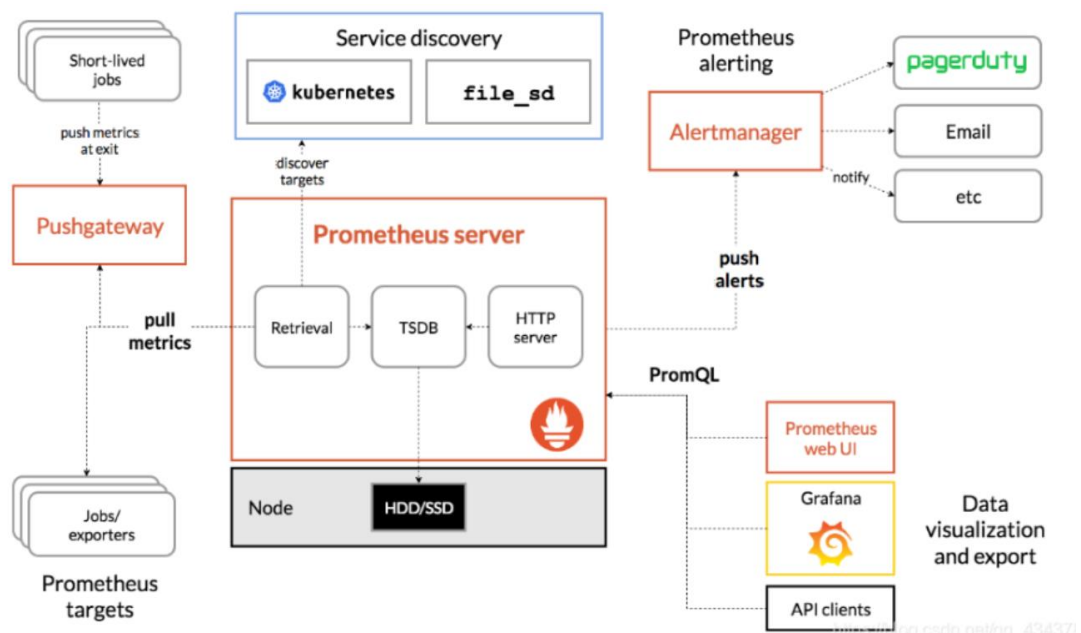
# Prometheus+Gafana 部署企业级监控

## （一）监控系统的组成

### 1、Prometheus

Prometheus 是一个开放性的监控解决方案，用户可以非常方便的安装和使用 Prometheus 并且能够非常方便的对其进行扩展。Prometheus 作为一个时序数据库，其实它和大家熟知的 Mysql 是一类的东西，都是存储数据，提供查询的，它存储了计算机系统在各个时间点上的监控数据。而 Grafana 仪表盘上的数据，就是通过查询 Prometheus 获取的。Prometheus 主要用于对基础设施的监控，包括服务器(CPU、MEM 等)、数据库(MYSQL、PostgreSQL 等)、Web 服务等，几乎所有东西都可以通过 Prometheus 进行监控。而它的数据，则是通过配置，建立与数据源的联系来获取的。

架构图：



Prometheus server - 收集和存储时间序列数据

Client Library: 客户端库，为需要监控的服务生成相应的

metrics 并暴露给 - Prometheus server。当 Prometheus server 来 pull 时，直接返回实时状态的 metrics。

pushgateway - 对于短暂运行的任务，负责接收和缓存时间序列数据，同时也是一个数据源

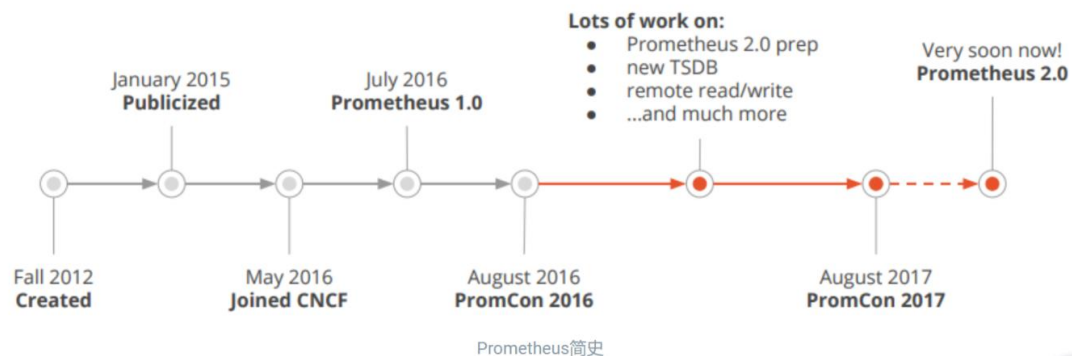
exporter - 各种专用 exporter，面向硬件、存储、数据库、HTTP 服务等

alertmanager - 处理报警

webUI 等, 其他各种支持的工具, 本身的界面值适合用来语句查询, 数据可视化, 需要第三方组件, 比如 Grafana。

## (1) 来源

Prometheus 受启发于 Google 的 Borgmon 监控系统 (相似的 Kubernetes 是从 Google 的 Borg 系统演变而来), 从 2012 年开始由前 Google 工程师在 Soundcloud 以开源软件的形式进行研发, 并且于 2015 年早期对外发布早期版本。



## (2) 优点

### ① 易于管理，无依赖存储

Prometheus 核心部分只有一个单独的二进制文件, 不存在任何的第三方依赖 (数据库, 缓存等等)。唯一需要的就是本地磁盘, 因此不会有潜在级联故障的风险。

### ② 强大的数据模型

所有采集的监控数据均以指标 (metric) 的形式保存在内置的时间序列数据库当中 (TSDB)。所有的样本除了基本的指标名称以外, 还包含一组用于描述该样本特征的标签。

如下所示:

```
http_request_status{code='200',content_path='/api/path', environment='produment'} =>
[value1@timestamp1,value2@timestamp2...]

http_request_status{code='200',content_path='/api/path2', environment='produment'} =>
[value1@timestamp1,value2@timestamp2...]
```

每一条时间序列由指标名称 (Metrics Name) 以及一组标签 (Labels) 唯一标识。每条时间序列按照时间的先后顺序存储一系列的样本值。

表示维度的标签可能来源于你的监控对象的状态，比如 `code=404` 或者 `content_path=/api/path`。也可能来源于你的环境定义，比如 `environment=prod`。基于这些 Labels 我们可以方便地对监控数据进行聚合，过滤，裁剪。

### ③ 强大的查询语言 PromQL

Prometheus 内置了一个强大的数据查询语言 PromQL。通过 PromQL 可以实现对监控数据的查询、聚合。同时 PromQL 也被应用于数据可视化(如 Grafana)以及告警当中。

### ④ 高效易于集成

对于单一 Prometheus Server 实例而言它可以处理：

- 数以百万的监控指标
- 每秒处理数十万的数据点。
- 使用 Prometheus 可以快速搭建监控服务，并且可以非常方便地在应用程序中进行集成。目前支持：Java, JMX, Python, Go, Ruby, .Net, Node.js 等等语言的客户端 SDK，基于这些 SDK 可以快速让应用程序纳入到 Prometheus 的监控当中，或者开发自己的监控数据收集程序

## 2、Grafana

是一个监控仪表系统，由 Grafana Labs 公司开源的一个系统监测（System Monitoring）工具。帮助用户简化监控的复杂度，用户只需要提供需要监控的数据，它就可以生成各种可视化仪表。同时它还支持报警功能，可以在系统出现问题时通知用户。并且 Grafana 不仅仅只支持 Prometheus 作为查询的数据库，它还支持如下：

- Prometheus
- Graphite
- OpenTSDB
- InfluxDB
- MySQL/PostgreSQL
- Microsoft SQL Serve
- 等等

其实 Prometheus 开发了一套仪表盘系统 [PromDash](#) 不过很快这套系统就被废弃了，官方开始推荐使用 Grafana 来对 Prometheus 的指标数据进行可视化，

这不仅是因为 Grafana 的功能非常强大，而且它和 Prometheus 可以完美的无缝融合。

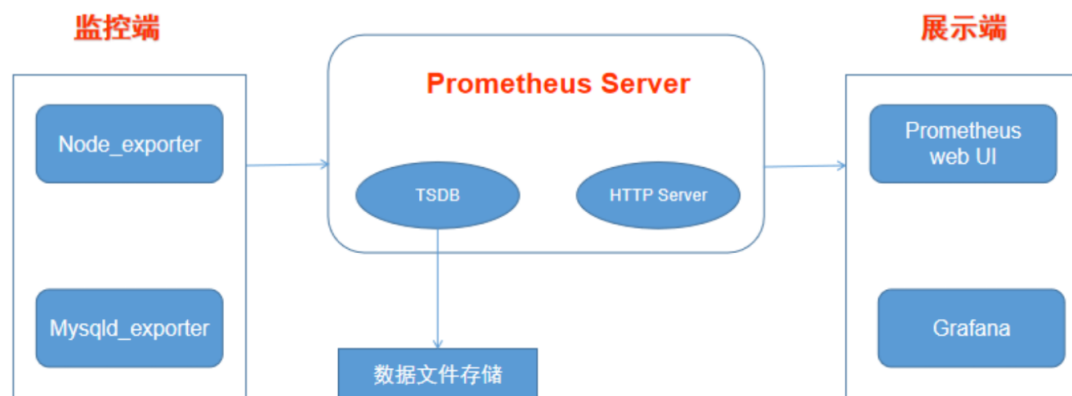
安装 Grafana，可以使用最简单的 [Docker 安装方式](#)：

```
$ docker run -d -p 3000:3000 grafana/grafana
```

运行上面的 docker 命令，Grafana 就安装好了！你也可以采用其他的安装方式，参考 [官方的安装文档] (<http://docs.grafana.org/>)。

### 3、数据源

在 Prometheus 的架构设计中，Prometheus 并不直接服务监控特定的目标，就比如我们监控 linux 系统，Prometheus 不会自己亲自去监控 linux 的各项指标。其主要任务负责数据的收集，存储并且对外提供数据查询支持。一般是一个 Exporter 服务提供的。



## （二）Prometheus 重要组成部分

### 1、Exporter

Exporter 是一个相对开放的概念，不是专门指某一个程序。它可以是一个独立运行的程序，独立于监控目标以外(如 Node Exporter 程序，独立于操作系统，却能获取到系统各类指标)。也可以是直接内置在监控目标中的代码(如在项目代码层面接入普罗米修斯 API，实现指标上报)。总结下来就是，只要能够向 Prometheus 提供标准格式的监控样本数据，那就是一个 Exporter。Prometheus 周期性的从 Exporter 暴露的 HTTP 服务地址(通常是/metrics)拉取监控样本数据。

一般来说可以将 Exporter 分为 2 类：

- 直接采集：这一类 Exporter 直接内置了对 Prometheus 监控的支持，比如 cAdvisor, Kubernetes, Etcd, Gokit 等，都直接内置了用于向 Prometheus 暴露监控数据的端点。
- 间接采集：间接采集，原有监控目标并不直接支持 Prometheus，因此我们需要通过 Prometheus 提供的 Client Library 编写该监控目标的监控采集程序。例如：Mysql Exporter, JMX Exporter, Consul Exporter 等。

后面如果要在我们的 C++ 项目里采集业务数据的话就得去利用间接采集的方法获取数据，可以利用第三方库 [prometheus-cpp](#)

## 2、AlertManager

在 Prometheus Server 中支持基于 PromQL 创建告警规则，如果满足 PromQL 定义的规则，则会产生一条告警，而告警的后续处理流程则由 AlertManager 进行管理。在 AlertManager 中我们可以与邮件，Slack 等等内置的通知方式进行集成，也可以通过 Webhook 自定义告警处理方式。AlertManager 即 Prometheus 体系中的告警处理中心。

## 3、PromQL

Prometheus 通过 PromQL 用户可以非常方便地对监控样本数据进行统计分析，PromQL 支持常见的运算操作符，同时 PromQL 中还提供了大量的内置函数可以实现对数据的高级处理。PromQL 作为 Prometheus 的核心能力除了实现数据的对外查询和展现，同时告警监控也是依赖 PromQL 实现的。

### (1) 基本用法

#### ①查询时间序列

当 Prometheus 通过 Exporter 采集到相应的监控指标样本数据后，我们就可以通过 PromQL 对监控样本数据进行查询。

当我们直接使用监控指标名称查询时，可以查询该指标下的所有时间序列。  
如：

```
prometheus_http_requests_total  
等同于：  
prometheus_http_requests_total{}
```

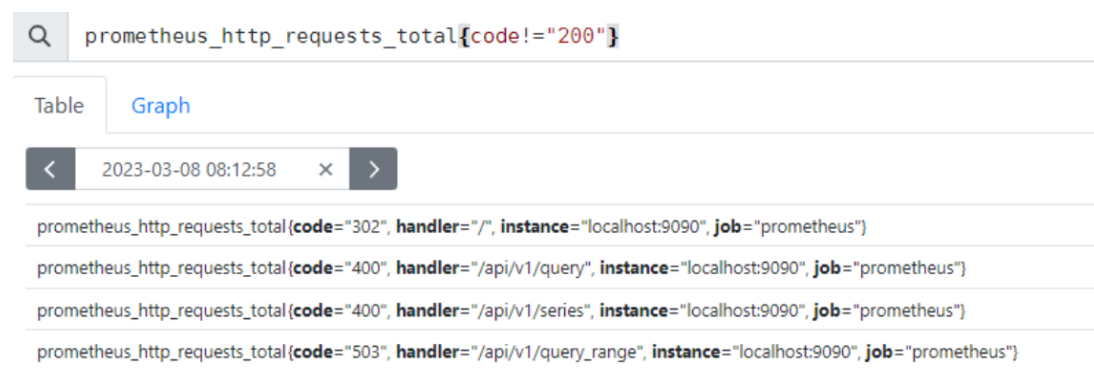
该表达式会返回指标名称为 `prometheus_http_requests_total` 的所有时间序列

PromQL 还支持用户根据时间序列的标签匹配模式来对时间序列进行过滤，目前主要支持两种匹配模式：完全匹配和正则匹配。

➤ PromQL 支持使用 `=` 和 `!=` 两种完全匹配模式：

- 通过使用 `label=value` 可以选择那些标签满足表达式定义的时间序列；

- 反之使用 `label!=value` 则可以根据标签匹配排除时间序列；



The screenshot shows a query interface with a search bar containing the query `prometheus_http_requests_total{code!="200"}`. Below the search bar are tabs for 'Table' and 'Graph'. The 'Table' tab is active, displaying a table with four rows of query results. Each row shows the metric name followed by its labels in curly braces.

Query
<code>prometheus_http_requests_total{code="302", handler="/", instance="localhost:9090", job="prometheus"}</code>
<code>prometheus_http_requests_total{code="400", handler="/api/v1/query", instance="localhost:9090", job="prometheus"}</code>
<code>prometheus_http_requests_total{code="400", handler="/api/v1/series", instance="localhost:9090", job="prometheus"}</code>
<code>prometheus_http_requests_total{code="503", handler="/api/v1/query_range", instance="localhost:9090", job="prometheus"}</code>

➤ PromQL 还可以支持使用正则表达式作为匹配条件，多个表达式之间使用 `|` 进行分离：

- 使用 `label=~regex` 表示选择那些标签符合正则表达式定义的时间序列；

- 反之使用 `label!~regex` 进行排除；

例如，如果想查询多个环节下的时间序列序列可以使用如下表达式：

```
prometheus_http_requests_total{environment=~"staging|testing|development",method!
="GET
"}
排除用法
prometheus_http_requests_total{environment!~"staging|testing|development",method!
="GET
"}
```

## ②范围查询

直接通过类似于 PromQL 表达式 `httprequeststotal` 查询时间序列时，返回值中只会包含该时间序列中的最新的一个样本值，这样的返回结果我们称之为瞬时向量。而相应的这样的表达式称之为 瞬时向量表达式。

而如果我们想过去一段时间范围内的样本数据时，我们则需要使用区间向量表达式。区间向量表达式和瞬时向量表达式之间的差异在于在区间向量表达式中我们需要定义时间选择的范围，时间范围通过时间范围选择器 `[]` 进行定义。例如，通过以下表达式可以选择

最近 5 分钟内的所有样本数据：

```
prometheus_http_requests_total{}[5m]
```

该表达式将会返回查询到的时间序列中最近 5 分钟的所有样本数据

通过区间向量表达式查询到的结果我们称为区间向量。除了使用 `m` 表示分钟以外，

PromQL 的时间范围选择器支持其它时间单位：

```
s - 秒  
m - 分钟  
h - 小时  
d - 天  
w - 周  
y - 年
```

## ③时间位移操作

在瞬时向量表达式或者区间向量表达式中，都是以当前时间为基准：

```
prometheus_http_requests_total{} # 瞬时向量表达式，选择当前最新的数据  
prometheus_http_requests_total{}[5m] # 区间向量表达式，选择以当前时间为基准，5 分钟内的数据
```

而如果我们想查询，5 分钟前的瞬时样本数据，或昨天一天的区间内的样本数据呢？这个时候我们就可以使用位移操作，位移操作的关键字为 `offset`。可以使用 `offset` 时间位移操作：

```
prometheus_http_requests_total{} offset 5m  
prometheus_http_requests_total{}[1d] offset 1d
```

## ④使用聚合操作

一般来说，如果描述样本特征的标签(label)在并非唯一的情况下，通过

PromQL 查询数据，会返回多条满足这些特征维度的时间序列。而 PromQL 提供的聚合操作可以用来对这些时间序列进行处理，形成一条新的时间序列：

```
# 查询系统所有 http 请求的总量
sum(prometheus_http_requests_total)
# 按照 mode 计算主机 CPU 的平均使用时间
avg(node_cpu_seconds_total) by (mode)
# 按照主机查询各个主机的 CPU 使用率
sum(sum(irate(node_cpu_seconds_total{mode!='idle'}[5m])) / sum(irate(node_cpu_seconds_total[5m]))) by (instance)
```

## ⑤标量和字符串

除了使用瞬时向量表达式和区间向量表达式以外，PromQL 还直接支持用户使用标量(Scalar)和字符串(String)。

➤ 标量 (Scalar)：一个浮点型的数字值

标量只有一个数字，没有时序。 例如：10

需要注意的是，当使用表达式 `count(prometheus_http_requests_total)`，返回的数据类型，依然是瞬时向量。用户可以通过内置函数 `scalar()` 将单个瞬时向量转换为标量。

➤ 字符串 (String)：一个简单的字符串值

直接使用字符串，作为 PromQL 表达式，则会直接返回字符串。

```
"this is a string"
'these are unescaped: \n \\ \t'
`these are not unescaped: \n ' " \t`
```

## (2) PromQL 操作符

使用 PromQL 除了能够方便的按照查询和过滤时间序列以外，PromQL 还支持丰富的操作符，用户可以使用这些操作符对进一步的对事件序列进行二次加工。这些操作符包括：数学运算符，逻辑运算符，布尔运算符等等。



## ①数学运算

PromQL 支持的所有数学运算符如下所示：

```
+ (加法)
- (减法)
* (乘法)
/ (除法)
% (求余)
^ (幂运算)
```

## ②布尔运算

Prometheus 支持以下布尔运算符如下：

```
== (相等)
!= (不相等)
> (大于)
< (小于)
>= (大于等于)
<= (小于等于)
```

## (3) 集合运算符

使用瞬时向量表达式能够获取到一个包含多个时间序列的集合，我们称为瞬时向量。通过集合运算，可以在两个瞬时向量与瞬时向量之间进行相应的集合操作。

目前，Prometheus 支持以下集合运算符：

```
and (并且)
or (或者)
unless (排除)
```

`vector1 and vector2` 会产生一个由 `vector1` 的元素组成的新的向量。该向量包含 `vector1` 中完全匹配 `vector2` 中的元素组成。

`vector1 or vector2` 会产生一个新的向量，该向量包含 `vector1` 中所有的样本数据，以及 `vector2` 中没有与 `vector1` 匹配到的样本数据。

`vector1 unless vector2` 会产生一个新的向量，新向量中的元素由 `vector1` 中没有与 `vector2` 匹配的元素组成。

## （4）操作符优先级

对于复杂类型的表达式，需要了解运算操作的运行优先级。例如，查询主机的 CPU 使用率，可以使用表达式：

```
100 * (1 - avg (irate(node_cpu_seconds_total{mode='idle'}[5m])) by(job) )
```

其中 `irate` 是 PromQL 中的内置函数，用于计算区间向量中时间序列每秒的即时增长率。在 PromQL 操作符中优先级由高到低依次为：

```
^
*, /, %
+, -
==, !=, <=, =, >
and, unless
or
```

## （5）PromQL 聚合操作

Prometheus 还提供了下列内置的聚合操作符，这些操作符作用域瞬时向量。可以将瞬时表达式返回的样本数据进行聚合，形成一个新的时间序列

```
sum (求和)
min (最小值)
max (最大值)
avg (平均值)

stddev (标准差)
stdvar (标准差异)
count (计数)
count_values (对 value 进行计数)
bottomk (后 n 条时序)
topk (前 n 条时序)
quantile (分布统计)
```

使用聚合操作的语法如下：

```
<aggr-op>([parameter,] <vector expression>) [without|by (<label list>)]
```

其中只有 `count_values`，`quantile`，`topk`，`bottomk` 支持参数 (parameter)。

without 用于从计算结果中移除列举的标签，而保留其它标签。by 则正好相反，结果向量中只保留列出的标签，其余标签则移除。通过 without 和 by 可以按照样本的问题对数据进行聚合。

### (三) 样本数据

一般来说访问 Exporter 暴露的 HTTP 服务，就能获取到了一系列的监控指标。而这些监控指标便是 Prometheus 可以采集到当前主机所有监控指标的样本数据。

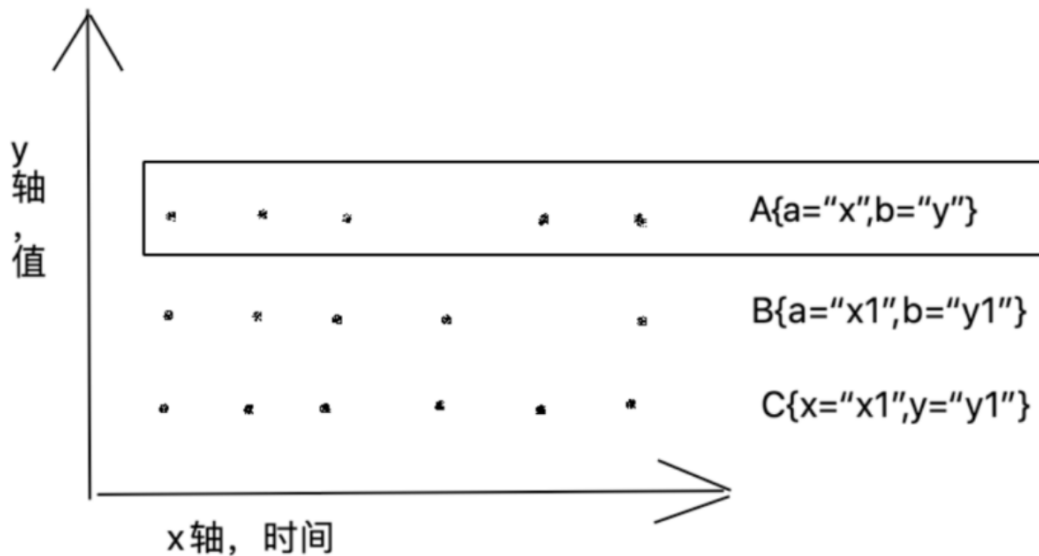
这是我部署的 node Exporter 服务的样本数据

```
go_threads 9
# HELP net_conntrack_dialer_conn_attempted_total Total number of connections attempted by the given dialer a given name.
# TYPE net_conntrack_dialer_conn_attempted_total counter
net_conntrack_dialer_conn_attempted_total{dialer_name="alertmanager"} 0
net_conntrack_dialer_conn_attempted_total{dialer_name="default"} 0
net_conntrack_dialer_conn_attempted_total{dialer_name="linux"} 1
net_conntrack_dialer_conn_attempted_total{dialer_name="mysqld_exporter"} 5332
net_conntrack_dialer_conn_attempted_total{dialer_name="prometheus"} 1
# HELP net_conntrack_dialer_conn_closed_total Total number of connections closed which originated from the dialer of a given name.
# TYPE net_conntrack_dialer_conn_closed_total counter
net_conntrack_dialer_conn_closed_total{dialer_name="alertmanager"} 0
net_conntrack_dialer_conn_closed_total{dialer_name="default"} 0
net_conntrack_dialer_conn_closed_total{dialer_name="linux"} 0
net_conntrack_dialer_conn_closed_total{dialer_name="mysqld_exporter"} 0
net_conntrack_dialer_conn_closed_total{dialer_name="prometheus"} 0
# HELP net_conntrack_dialer_conn_established_total Total number of connections successfully established by the given dialer a given name.
# TYPE net_conntrack_dialer_conn_established_total counter
net_conntrack_dialer_conn_established_total{dialer_name="alertmanager"} 0
net_conntrack_dialer_conn_established_total{dialer_name="default"} 0
net_conntrack_dialer_conn_established_total{dialer_name="linux"} 1
net_conntrack_dialer_conn_established_total{dialer_name="mysqld_exporter"} 0
net_conntrack_dialer_conn_established_total{dialer_name="prometheus"} 1
# HELP net_conntrack_dialer_conn_failed_total Total number of connections failed to dial by the dialer a given name.
# TYPE net_conntrack_dialer_conn_failed_total counter
net_conntrack_dialer_conn_failed_total{dialer_name="alertmanager",reason="refused"} 0
net_conntrack_dialer_conn_failed_total{dialer_name="alertmanager",reason="resolution"} 0
net_conntrack_dialer_conn_failed_total{dialer_name="alertmanager",reason="timeout"} 0
net_conntrack_dialer_conn_failed_total{dialer_name="alertmanager",reason="unknown"} 0
net_conntrack_dialer_conn_failed_total{dialer_name="default",reason="refused"} 0
net_conntrack_dialer_conn_failed_total{dialer_name="default",reason="resolution"} 0
net_conntrack_dialer_conn_failed_total{dialer_name="default",reason="timeout"} 0
```

一条样本数据 = 样本名称（指标）+ 标签（键值对中的键）+ 对应的值（右大括号后的值则是该监控样本监控下的具体值）

#### 1、样本(sample)

Prometheus 会将所有采集到的样本数据以\*\*时间序列(time-series)\*\*的方式保存在内存数据库中，并且定时保存到硬盘上。时间序列保存方式是指按照时间戳和值的序列顺序存放，也称之为向量(vector)。 每条时间序列通过指标名称(metrics name)和一组标签集(labelset)命名。如下图所示，可以将向量理解为一个以时间为 X 轴，值为 Y 轴的数字矩阵：



在时间序列中的每一个点(即图上的小黑点)称为一个**样本(sample)**，样本由以下三部分组成：

- 指标(metric)：metric name 和描述当前样本特征的 labelsets，也就是图中的 `A{a="x",b="y"}`；
- 时间戳(timestamp)：一个精确到毫秒的时间戳，也就是小黑点对应的 x 轴的值；
- 样本值(value)：一个 float64 的浮点型数据表示当前样本的值，即小黑点对应的 y 轴的值；

即样本可表示为：

```
A{a="x",b="y"}@1434417560938 => 94355
```

其中 1434417560938 是时间戳，94355 是值。

## 2、指标 (Metric)

在形式上，所有的指标 (Metric) 都通过如下格式标示：

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

指标的名称 (metric name) 可以反映被监控样本的含义(比如，`httprequest_total` - 表示当前系统接收到的 HTTP 请求总量)。指标名称只能由 ASCII 字符、数字、下划线以及冒号组成并必须符合正则表达式 `[a-zA-Z:~_]*`。

### 3、标签(label)

标签反映了当前样本的特征维度，通过这些维度 Prometheus 可以对样本数据进行过滤，聚合等。标签的名称只能由 ASCII 字符、数字以及下划线组成并满足正则表达式 `a-zA-Z_*`。

其中以 `_` 作为前缀的标签，是系统保留的关键字，只能在系统内部使用。标签的值则可以包含任何 Unicode 编码的字符。

## （四）实操

---

### 1、使用官方提供的 exporter 及模板库

监控 Linux 服务器各项指标和 mysql 服务各项指标

#### （1）下载并安装 exporter

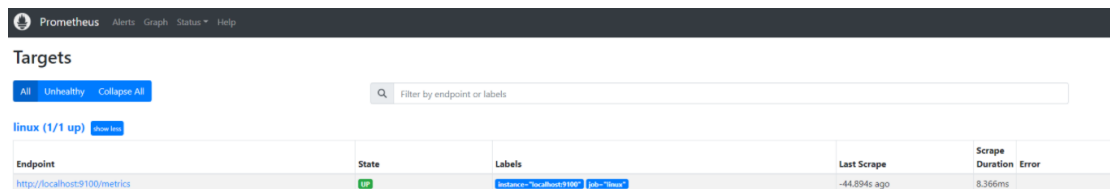
<https://prometheus.io/download/>

#### （2）修改 Prometheus 配置

```
31 - job_name: 'linux'
32   static_configs:
33     - targets: ['localhost:9100'] # 多个用,分开
34
35 - job_name: "mysqld_exporter"
36   static_configs:
37     - targets: ['localhost:9104']
38
```

### (3) 重启 Prometheus


(4) 打开 Prometheus 面板，发现已经出现了名为 linux 的 target



The screenshot shows the Prometheus web interface. At the top, there's a navigation bar with 'Prometheus', 'Alerts', 'Graph', 'Status', and 'Help'. Below it, the 'Targets' section is active. A search bar says 'Filter by endpoint or labels'. Under 'linux (1/1 up)', there's a table with one row:

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9100/metrics	UP	instance="localhost:9100" job="linux"	-44.894s ago	8.366ms	

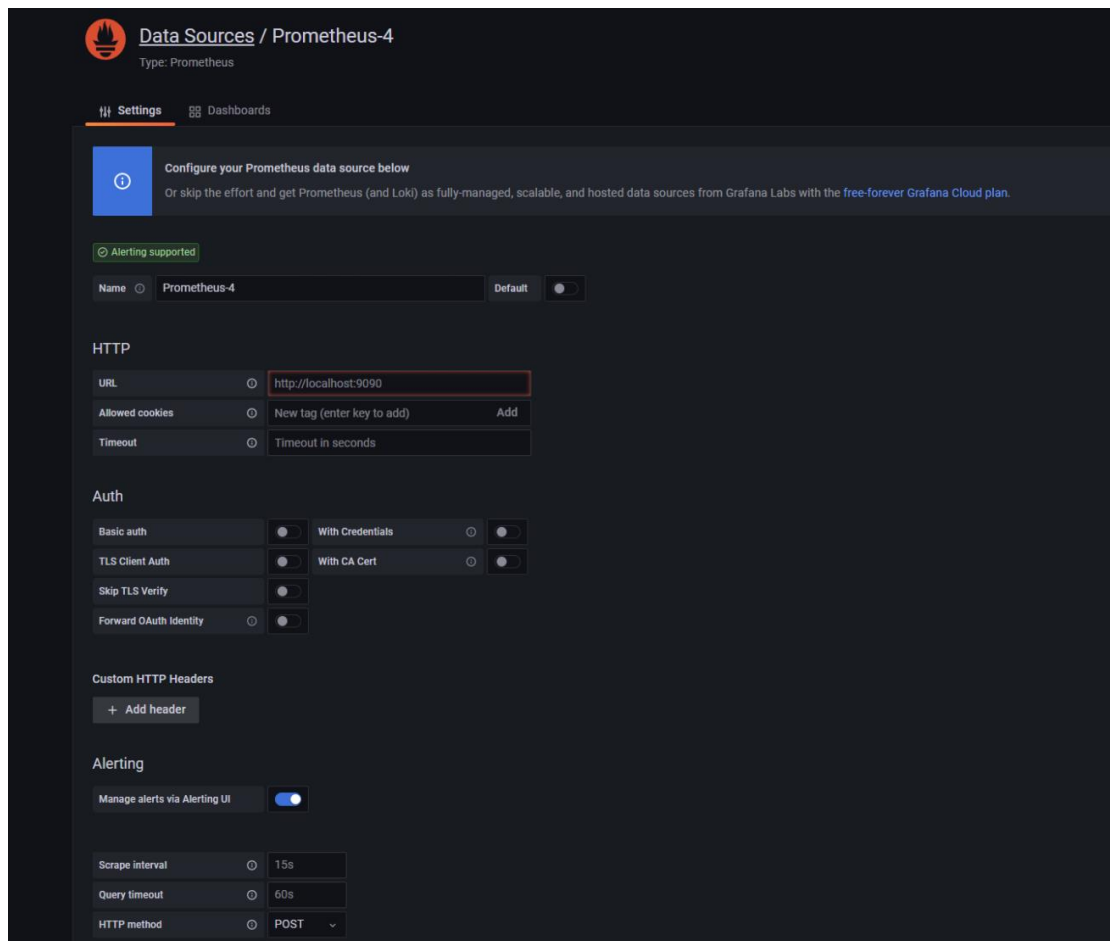
### (5) 通过 http 服务获取到样本数据



The screenshot shows a web browser at the URL `10.132.16.36:9100/metrics`. The page displays a list of Prometheus metrics for the 'linux' target. The metrics include garbage collection statistics, goroutine counts, and memory usage details.

```
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.5756e-05
go_gc_duration_seconds{quantile="0.25"} 2.8287e-05
go_gc_duration_seconds{quantile="0.5"} 3.0129e-05
go_gc_duration_seconds{quantile="0.75"} 4.23e-05
go_gc_duration_seconds{quantile="1"} 6.539e-05
go_gc_duration_seconds_sum 0.128410795
go_gc_duration_seconds_count 3746
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 9
# HELP go_info Information about the Go environment.
# TYPE go_info gauge
go_info{version="go1.19.3"} 1
# HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
# TYPE go_memstats_alloc_bytes gauge
go_memstats_alloc_bytes 2.185408e+06
# HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even if freed.
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 6.737064032e+09
# HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling bucket hash table.
# TYPE go_memstats_buck_hash_sys_bytes gauge
go_memstats_buck_hash_sys_bytes 1.756419e+06
# HELP go_memstats_frees_total Total number of frees.
# TYPE go_memstats_frees_total counter
go_memstats_frees_total 9.0405554e+07
# HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection system metadata.
# TYPE go_memstats_gc_sys_bytes gauge
go_memstats_gc_sys_bytes 9.453056e+06
# HELP go_memstats_heap_alloc_bytes Number of heap bytes allocated and still in use.
# TYPE go_memstats_heap_alloc_bytes gauge
go_memstats_heap_alloc_bytes 2.185408e+06
# HELP go_memstats_heap_idle_bytes Number of heap bytes waiting to be used.
# TYPE go_memstats_heap_idle_bytes gauge
go_memstats_heap_idle_bytes 4.13696e+06
# HELP go_memstats_heap_inuse_bytes Number of heap bytes that are in use.
# TYPE go_memstats_heap_inuse_bytes gauge
go_memstats_heap_inuse_bytes 3.923968e+06
# HELP go_memstats_heap_objects Number of allocated objects.
# TYPE go_memstats_heap_objects gauge
go_memstats_heap_objects 12648
# HELP go_memstats_heap_released_bytes Number of heap bytes released to OS.
# TYPE go_memstats_heap_released_bytes gauge
go_memstats_heap_released_bytes 3.260416e+06
# HELP go_memstats_heap_sys_bytes Number of heap bytes obtained from system.
# TYPE go_memstats_heap_sys_bytes gauge
go_memstats_heap_sys_bytes 8.060928e+06
# HELP go_memstats_last_gc_time_seconds Number of seconds since 1970 of last garbage collection.
# TYPE go_memstats_last_gc_time_seconds gauge
go_memstats_last_gc_time_seconds 1.6781775069119754e+09
# HELP go_memstats_lookups_total Total number of pointer lookups.
```

## （6）创建一个 datasource



The screenshot shows the Grafana 'Data Sources' configuration page for a Prometheus data source named 'Prometheus-4'. The page is dark-themed and includes a top navigation bar with 'Settings' and 'Dashboards' tabs. A blue information box at the top instructs the user to configure the Prometheus data source below, mentioning the Grafana Cloud plan. Below this, a green status bar indicates 'Alerting supported'. The configuration is organized into several sections: 'Name' (Prometheus-4), 'HTTP' (URL: http://localhost:9090, Allowed cookies, Timeout), 'Auth' (Basic auth, TLS Client Auth, Skip TLS Verify, Forward OAuth Identity), 'Custom HTTP Headers' (Add header button), 'Alerting' (Manage alerts via Alerting UI toggle), and 'Scrape interval' (15s), 'Query timeout' (60s), and 'HTTP method' (POST).

**Data Sources / Prometheus-4**  
Type: Prometheus

**Settings** | Dashboards

**Configure your Prometheus data source below**  
Or skip the effort and get Prometheus (and Loki) as fully-managed, scalable, and hosted data sources from Grafana Labs with the [free-forever Grafana Cloud plan](#).

**Alerting supported**

**Name** Prometheus-4 **Default** ☒

**HTTP**

**URL**

**Allowed cookies**  **Add**

**Timeout**

**Auth**

**Basic auth** ☒ **With Credentials** ☐ ☒

**TLS Client Auth** ☒ **With CA Cert** ☐ ☒

**Skip TLS Verify** ☒

**Forward OAuth Identity** ☐ ☒

**Custom HTTP Headers**

**+ Add header**

**Alerting**

**Manage alerts via Alerting UI** ☒

**Scrape interval**

**Query timeout**

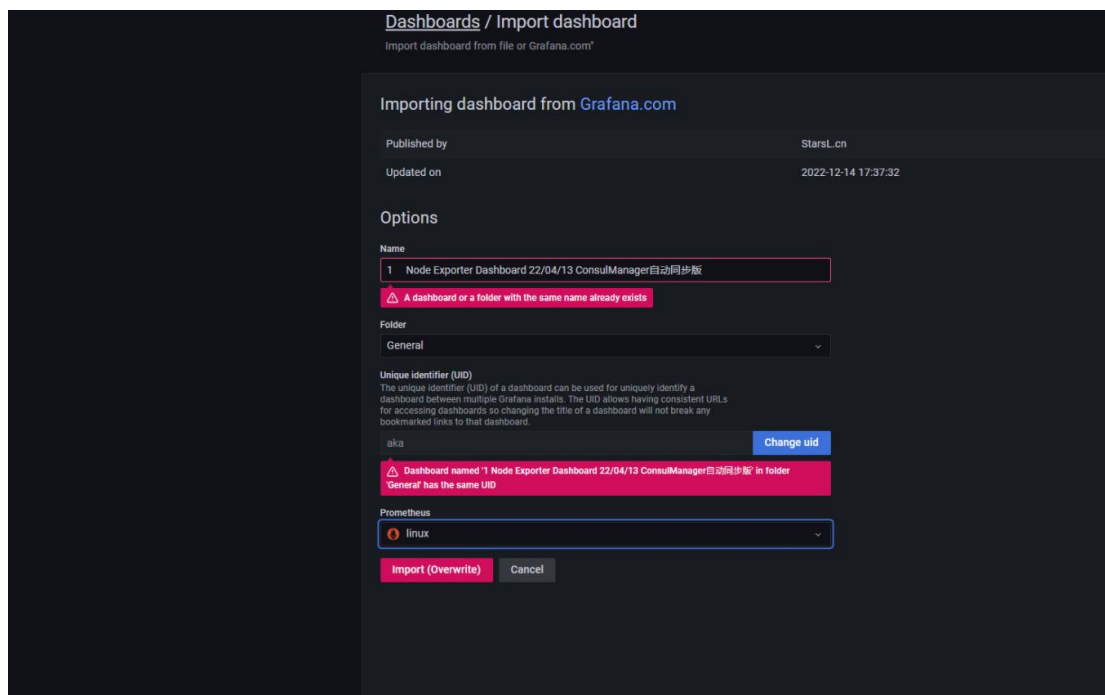
**HTTP method**

## （7）导入官方提供的 Dashboard

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

再添加本次要使用的模板

选择数据源，导入该模板就可以了。



## 2、自定义

Prometheus 提供了[官方库](#)用于采集并暴露监控数据，下面让我们来快速的来使用官方库来采集程序内相关的数据，以及其它一些基本简单的示例，并使用 Prometheus 监控服务来采集指标展示数据。

这里有一些官方的客户端库和一些非官方的客户端库，囊括了大部分语言：[链接](#)

### （1）Prometheus 指标的四种类型

#### ① Counter（计数器）

counter metric 是一个只能递增的 value

看一下它的接口

```
type Counter interface {
    Metric
    Collector

    // Inc increments the counter by 1. Use Add to increment it by arbitrary
    // non-negative values.
    Inc()
    // Add adds the given value to the counter. It panics if the value is <
    // 0.
```



```
Add(float64)
}
```

Inc 方法，默认+1；Add 方法，可以增加自定义的数量。

适用场景：

- 记录不同的 API 的请求数量
- 记录业务里某个错误码触发数量
- 记录一段时间内风控规则的使用数量

## ② Gauge（仪表盘）

是一个可增可减的指标

接口：

```
type Gauge interface {
    Metric
    Collector

    // Set sets the Gauge to an arbitrary value.
    Set(float64)
    // Inc increments the Gauge by 1. Use Add to increment it by arbitrary
    // values.
    Inc()
    // Dec decrements the Gauge by 1. Use Sub to decrement it by arbitrary
    // values.
    Dec()
    // Add adds the given value to the Gauge. (The value can be negative,
    // resulting in a decrease of the Gauge.)
    Add(float64)
    // Sub subtracts the given value from the Gauge. (The value can be
    // negative, resulting in an increase of the Gauge.)
    Sub(float64)

    // SetToCurrentTime sets the Gauge to the current Unix time in seconds.
    SetToCurrentTime()
}
```

适用场景：

- 记录服务的内存的占用
- 记录服务 CPU 的占用

- 记录队列的长度
- 某时刻协议来的数量等

### ③Histograms（直方图、柱状图）

histograms 是一个 直方图度量类型，用于测量落在定义的桶中的数据的值。比如用在测量我们的请求大部分的延迟是落在哪一个区间。这个时候 Prometheus 不会存储每一次请求所消耗的时间，而是会将每一个请求按照消耗时间看是分配到哪一个 bucket。默认的 buckets 有：.005, .01, .025, .05, .075, .1, .25, .5, .75, 1, 2.5, 5, 7.5, 10。一般来说很少有超过 10 秒的请求了。当然如果我们需要，也是可以定制化。

接口：

```
type Histogram interface {  
    Metric  
    Collector  
  
    // Observe adds a single observation to the histogram. Observations are  
    // usually positive or zero. Negative observations are accepted but  
    // prevent current versions of Prometheus from properly detecting  
    // counter resets in the sum of observations. (The experimental Native  
    // Histograms handle negative observations properly.) See  
    // https://prometheus.io/docs/practices/histograms/#count-and-sum-of-  
    observations  
    // for details.  
    Observe(float64)  
}
```

使用场景：

- 比如我们的 API 服务的请求耗时，在所有的 bucket 分布情况。
- 比如我们消费者处理某个事件的耗时，在所有的 bucket 分布情况。

### ④ Summaries（）

summaries 和 histograms 有很多相似的地方。而不同的地方有以下几点：

- Histograms 是基于桶来统计数据的，而 Summaries 是基于分位数来统计数据的。

- histograms 分位数的计算是在 Prometheus 上面，而 summaries 是在 APP 服务上就进行了计算。因此 summaries 也没办法针对多个应用进行聚合。
- summaries 适用于需要计算准确的分位数，但不能确定值的范围是什么。

```
// To create Summary instances, use NewSummary.  
type Summary interface {  
    Metric  
    Collector  
  
    // Observe adds a single observation to the summary. Observations are  
    // usually positive or zero. Negative observations are accepted but  
    // prevent current versions of Prometheus from properly detecting  
    // counter resets in the sum of observations. See  
    // https://prometheus.io/docs/practices/histograms/#count-and-sum-of-observations  
    // for details.  
    Observe(float64)  
}
```

比如我们的 API 服务的请求耗时，大部分是落在哪个区间。

## (2) cgo

Prometheus 是 go 语言写的，官方库也是 go、java、python 等，没有 C 和 C++ 官方库，而非官方库 prometheus-cpp 的使用看起来较为繁琐，不太友好，所以我就直接使用 go 语言写代码，包装接口，再利用 cgo 机制，将接口让 C 程序或者 C++ 程序调用

Go 语言通过自带的一个叫 CGO 的工具来支持 C 语言函数调用，同时我们可以用 Go 语言导出 C 动态库接口给其他语言使用。

## (五) 利用 Grafana 搭建告警系统

我利用 qq 邮箱向开发人员发送告警邮件

客户端-->设置->账户

POP3/IMAP/SMTP/Exchange/CardDAV/CalDAV服务

开启服务：

POP3/SMTP服务 (如何使用 Foxmail 等软件收发邮件？)

已开启 | 关闭

IMAP/SMTP服务 (什么是 IMAP，它又是如何设置？)

已关闭 | 开启

Exchange服务 (什么是Exchange，它又是如何设置？)

已关闭 | 开启

CardDAV/CalDAV服务 (什么是CardDAV/CalDAV，它又是如何设置？)

已关闭 | 开启

(POP3/IMAP/SMTP/CardDAV/CalDAV服务均支持SSL连接。如何设置？)

温馨提示：在第三方登录QQ邮箱，可能存在邮件泄露风险，甚至危害Apple ID安全。建议使用QQ邮箱手机版登录。  
继续获取授权码登录第三方客户端邮箱 [②](#)。 [生成授权码](#)



收取选项：

最近30天

▾

的邮件

☐ 收取“我的文件夹”

☐ 收取“QQ邮件订阅”

☐ SMTP发信后保存到服务器

(以上收取选项对POP3/IMAP/SMTP/Exchange均生效。 [了解更多](#))

☐ 收取垃圾邮件隔周提醒

(该收取选项只对POP3生效。 [我使用了IMAP/Exchange协议，怎么办？](#))

同步选项：

☐ 禁止收信软件删信 (为什么会有收信软件删信？)

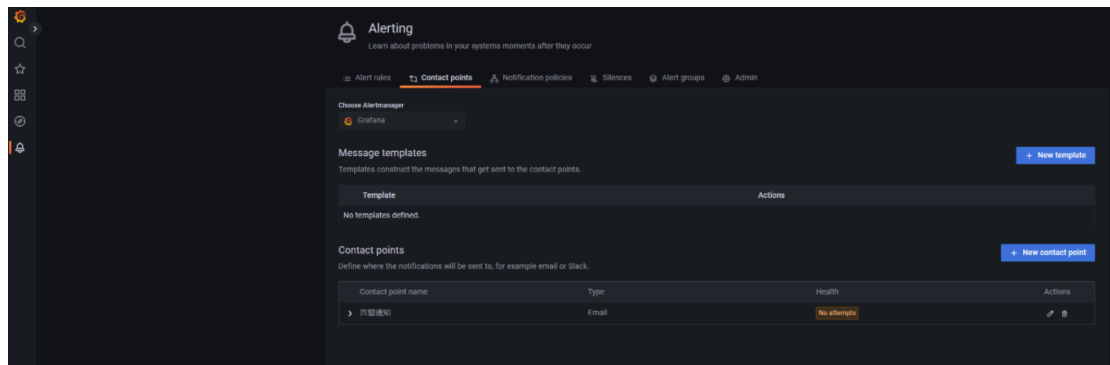
(该收取选项只对POP3生效。)

## (2) 开启 smtp 配置

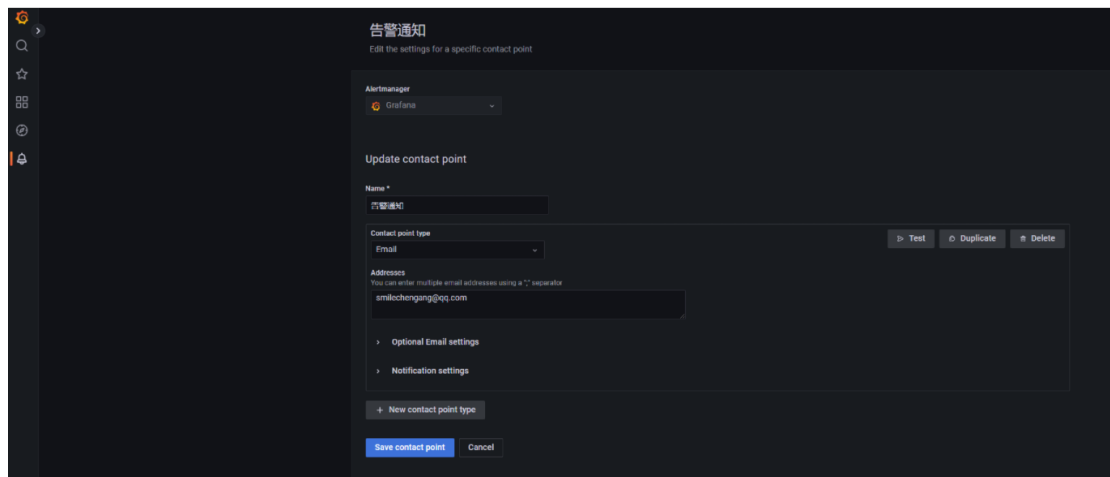
打开 Granfana 的配置文件，修改以下配置

```
685 ##### SMTP / Emailing #####
686 [smtp]
687 enabled = true
688 host = smtp.exmail.qq.com:465
689 user = smilechengang@qq.com
690 # If the password contains # or ; you have to wrap it with triple quotes. Ex ""#password;""
691 password = ivv333epbjc[ ]
692 ;cert_file =
693 ;key_file =
694 skip_verify = true
695 from_address = smilechengang@qq.com
696 from_name = Grafana
697 # EHLO identity in SMTP dialog (defaults to instance_name)
698 ;ehlo_identity = dashboard.example.com
699 # SMTP startTLS policy (defaults to 'OpportunisticStartTLS')
700 ;startTLS_policy = NoStartTLS
701
```

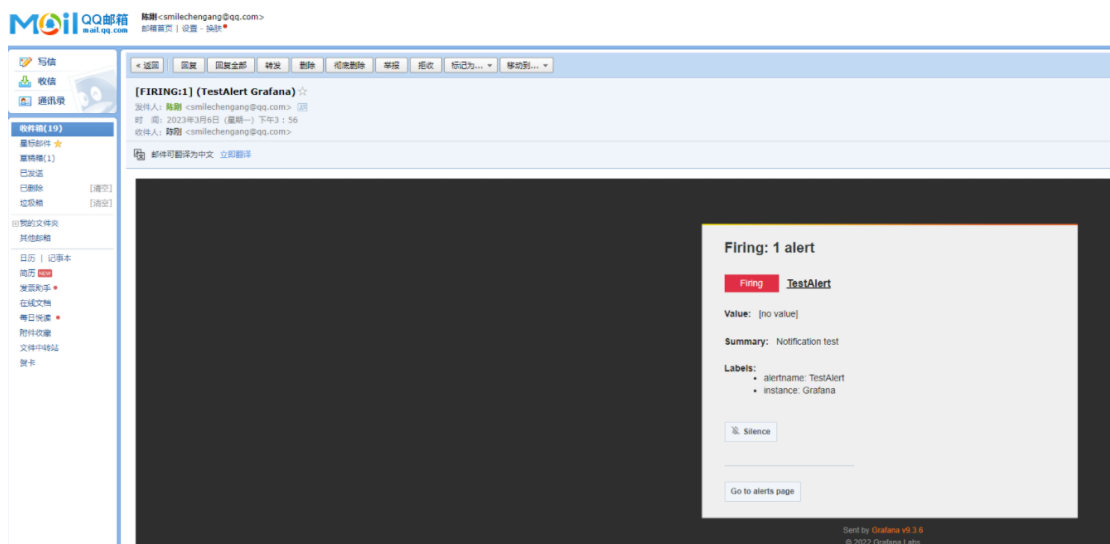
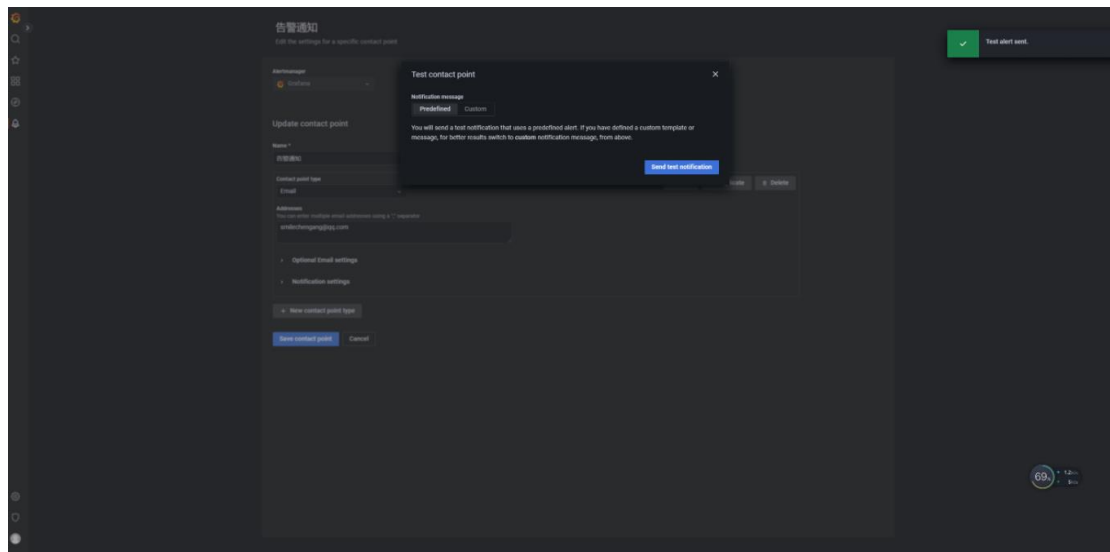
### (3) 在 Granfana 里添加一个告警通道



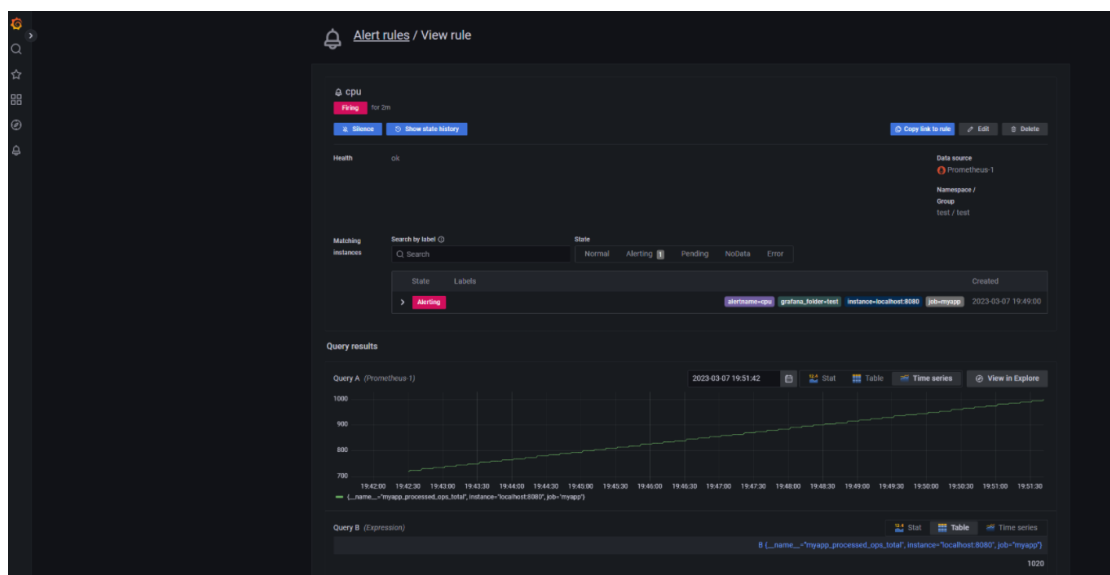
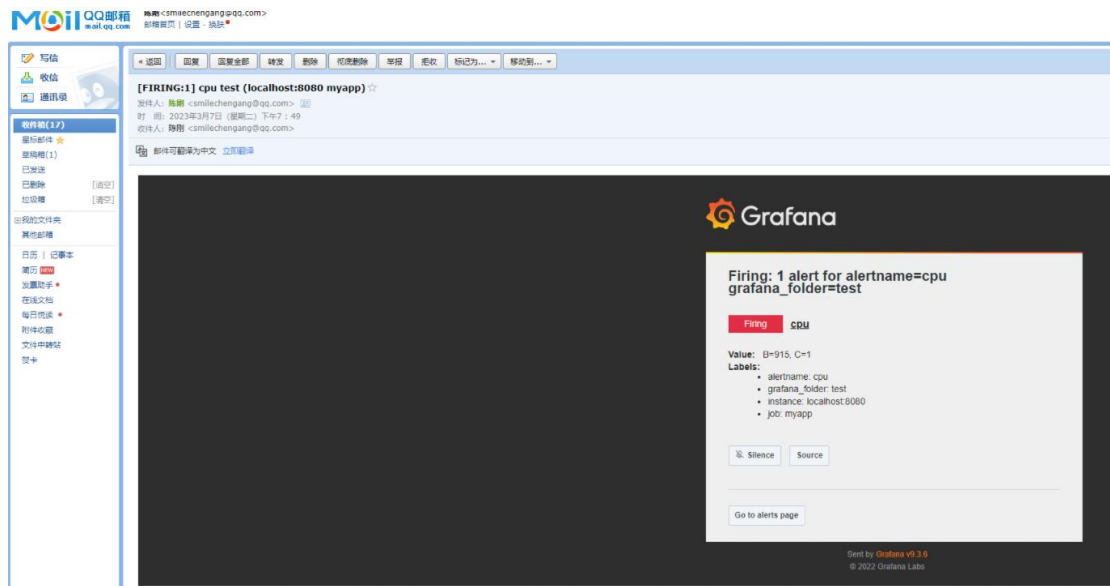
### (4) 配置一下要接收这个告警邮件的账户



## (5) 测试一下，发送成功了



## (6) 设置一个监控项



至此我们就搭建好了一套可视化监控告警系统!!!

参考资料:

prometheus 参考资料:

<https://yunlzheng.gitbook.io/prometheus-book/>

<https://zhuanlan.zhihu.com/p/434353542>

prometheus git 仓库: <https://github.com/prometheus/prometheus>

搭建环境: [https://blog.csdn.net/qq\\_31725371/article/details/114697770](https://blog.csdn.net/qq_31725371/article/details/114697770)

自定义绘图: <https://blog.csdn.net/admin321123/article/details/127590704>

官方提供的仪表盘: <https://grafana.com/grafana/dashboards/?dataSource=influxdb>

cgo: <http://caibaojian.com/go/09.0.html>