

Prometheus监控系统

Prometheus简介

什么是Prometheus?

Prometheus是由前 Google 工程师从 2012 年开始在 Soundcloud以开源软件的形式进行研发的系统监控和告警工具包，自此以后，许多公司和组织都采用了 Prometheus 作为监控告警工具。Prometheus 的开发者和用户社区非常活跃，它现在是一个独立的开源项目，可以独立于任何公司进行维护。为了证明这一点，Prometheus 于 2016 年 5 月加入CNCF基金会，成为继 Kubernetes 之后的第二个 CNCF 托管项目。

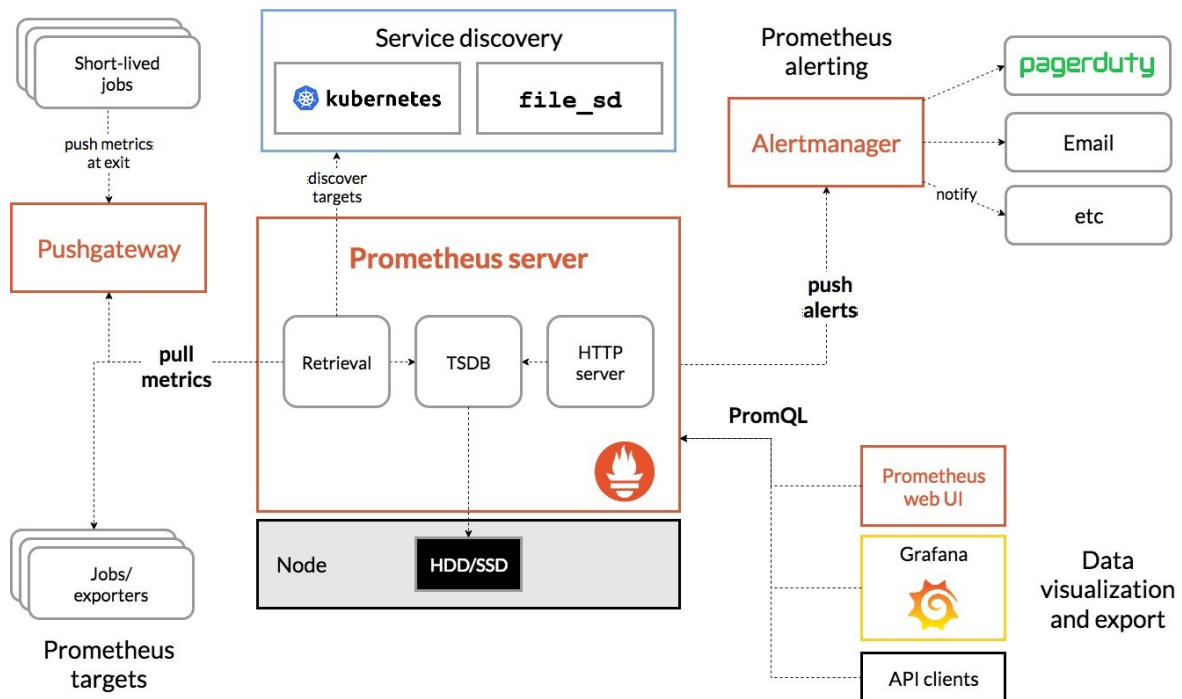
Prometheus的优势

Prometheus 的主要优势有：

- 由指标名称和键/值对标签标识的时间序列数据组成的多维数据模型
- 强大的查询语言 PromQL
- 不依赖分布式存储；单个服务节点具有自治能力。
- 时间序列数据是服务端通过 HTTP 协议主动拉取获得的。
- 也可以通过中间网关来推送时间序列数据
- 可以通过静态配置文件或服务发现来获取监控目标。
- 支持多种类型的图表和仪表盘。

Prometheus的组件、架构

Prometheus 的整体架构以及生态系统组件如下图所示：



Prometheus Server 直接从监控目标中或者间接通过推送网关来拉取监控指标，它在本地存储所有抓取到的样本数据，并对此数据执行一系列规则，以汇总和记录现有数据的新时间序列或生成告警。可以通过 Grafana或者其他工具来实现监控数据的可视化。

- Prometheus server是Prometheus架构中的核心组件，基于go语言编写而成，无第三方依赖关系，可以独立部署在物理服务器上、云主机、Docker容器内。主要用于收集每个目标数据，并存储为时间序列数据，对外可提供数据查询支持和告警规则配置管理。

Prometheus服务器可以对监控目标进行静态配置管理或者动态配置管理，它将监控采集到的数据按照时间序列存储在本地磁盘的时序数据库中（当然也支持远程存储），自身对外提供了自定义的PromQL语言，可以对数据进行查询和分析

- Client Library是用于检测应用程序代码的客户端库。在监控服务之前，需要向客户端库代码添加检测，从而事先Prometheus中metric的类型。
- Exporter用于输出被监控组件信息的HTTP接口统称为Exporter（导出器）。目前互联网公司常用的组件大部分都有Expoter供直接使用，比如Nginx、MySQL、linux系统信息等。
- Pushgateway是指用于支持短期临时或批量计划任务工作的汇聚节点。主要用于短期的job，此类存在的job时间较短，可能在Prometheus来pull之前就自动消失了。所以针对这类job，设计成可以直接向Pushgateway推送metric，这样Prometheus服务器端便可以定时去Pushgateway拉去metric
- Alertmanager主要用于处理Prometheus服务器端发送的alerts信息，对其去除重数据、分组并路由到正确的接收方式，发出告警，支持丰富的告警方式。
- Service Discovery：动态发现待监控的target，从而完成监控配置的重要组件，在容器环境中尤为重要，该组件目前由Prometheus Server内建支持

Prometheus适用于什么场景

Prometheus适用于记录文本格式的时间序列，它既适用于以机器为中心的监控，也适用于高度动态的面向服务的监控，在微服务的世界中，它对多维数据收集和查询的支持有特殊优势。Prometheus是专为提高系统可靠性而设计的，它可以在断电期间快速诊断问题，每个Prometheus Server都是相互独立的，不依赖于网络存储或者其他远程服务。当基础架构出现问题时，你可以通过Prometheus快速定位故障点，而且不会消耗大量的基础架构资源。

Prometheus不适合什么场景

Prometheus非常重视可靠性，即使在出现故障的情况下，你也可以随时统计有关系统的可用系统信息。如果你需要百分之百的准确度，例如按请求数量计费，那么Prometheus可能不太适合你，因为它收集的数据可能不够详细完整精确。

相关概念

数据模型

Prometheus所有采集的监控数据均以指标的形式保存在内置的时间序列数据库当中（TSDB）：属于同一指标名称、同一标签集合的、有时间戳标记的数据流。除了存储的时间序列，Prometheus还可以根据查询请求产生临时的、衍生的时间序列作为返回结果。

指标名称和标签

每一条时间序列由指标名称（Metric Name）以及一组标签（键值对）唯一标识。其中指标的名称（Metric Name）可以反映被监控样本的含义（例如，http_request_total可以看出来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，同时必须匹配正则表达式 `[a-zA-Z_:][a-zA-Z0-9_:]*`。

注意

冒号用来表示用户自定义的记录规则，不能在 exporter 中或监控对象直接暴露的指标中使用冒号来定义指标名称。

通过使用标签，Prometheus开启了强大的多维数据模型：对于相同的指标名称，通过不同标签列表的集合，会形成特定的度量维度实例（例如，所有包含度量名称为 `/api/tracks` 的 http 请求，打上 `method=POST` 的标签，就会形成具体的 http 请求）。查询语言在这些指标和标签列表的基础上进行过滤和聚合，改变任何度量指标上的任何标签值（包括添加或删除指标），都会创建新的时间序列。

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_][a-zA-Z0-9_]*`。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。

样本

在时间序列中的每一个点称为样本，样本由以下三部分组成：

- 指标（metric）：指标名称和描述当前样本特征的labelset；
- 时间戳：一个精确到时间毫秒的时间戳
- 样本值：一个浮点型数据表示当前样本的值

表示方式

通过如下表示方式表示指定名称和指定标签集合的时间序列

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

例如，指标名称为 `api_http_requests_total`，标签为 `method="POST"` 和 `handler="/messages"` 的时间序列可以表示为：

```
1 api_http_requests_total{method="POST", handler="/messages"}
```

指标类型

Prometheus的客户端库中提供了四种核心的指标类型。但这些类型只是在客户端库（客户端可以根据不同的数据类型调用不同的api接口）和在线协议中，实际在Prometheus Server中并不对指标类型进行区分，而是简单地把这些指标统一视为无类型的时间序列

Counter计数器

Counter类型代表一种样本数据单调递增的指标，即**只增不减**，除非监控系统发生了重置。例如，你可以使用counter类型的指标表示服务请求总数、已经完成任务数、错误发生的次数等。

counter类型数据可以让用户方便的了解事件发生的速率的变化，在PromQL内置的相关操作函数可以提供相应的分析，比如HTTP应用请求量来进行说明：

```
1 //通过rate()函数获取HTTP请求量的增长率
2 rate(http_requests_total[5m])
3 //查询当前系统中，访问量前10的HTTP地址
4 topk(10, http_requests_total)
```

Gauge仪表盘

Gauge类型代表一种样本数据可以任意变化的指标，即**可增可减**。Gauge通常用于像温度或者内存使用率这种指标数据，也可以表示能随时请求增加会减少的总数，例如当前并发请求的数量。



对于Gauge类型的监控指标，通过PromQL内置的函数delta（）可以获取样本在一段时间内的变化情况，例如，计算cpu温度在两小时内的差异：

```
1 | delta(cpu_temp_celsius{host="zeus"}[2h])
```

还可以通过PromQL内置函数predict_linear（）基于简单线性回归的方式，对样本数据的变化趋势做出预测。例如，基于两小时的样本数据，来预测主机可用磁盘空间在4个小时之后的剩余情况

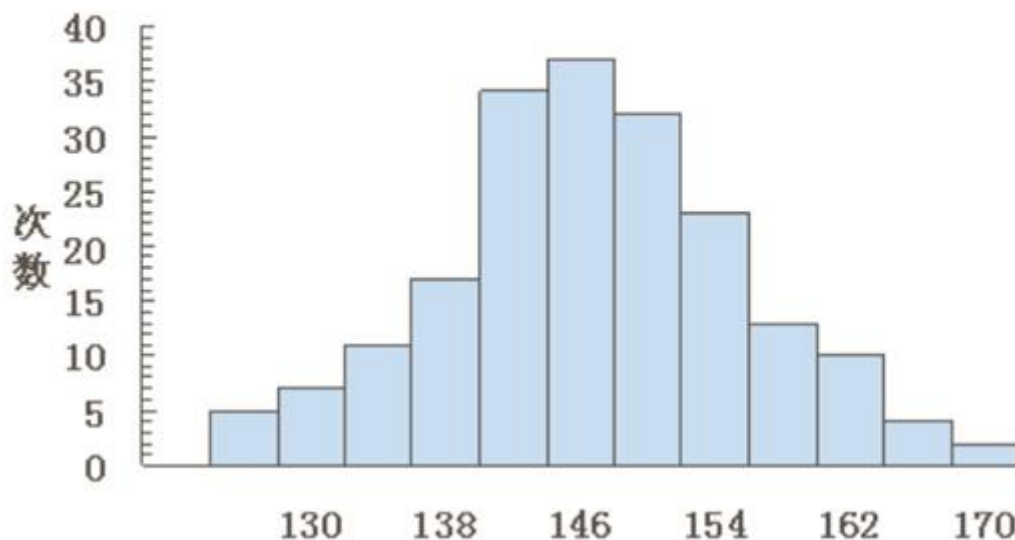
```
1 | predict_linear(node_filesystem_free{job="node"}[2h], 4 * 3600) < 0
```

Histogram直方图

在大多数情况下人们都倾向于使用某些量化指标的平均值，例如cpu的平均使用率、页面的平均响应时间。这种方式的问题很明显，以系统api调用的平均响应时间为例：如果大多数api请求都维持在100ms的响应时间范围内，而个别请求的响应时间需要5秒，那么就会导致某些web页面的响应落到中位数的情况，而这种现象被称为**长尾问题**。

为了区分是平均的慢还是长尾的慢，最简单的方式就是按照请求延迟的范围进行分组。例如，统计延迟在0-10ms之间的请求数有多少而10-20ms之间的请求数又有多少。通过这种方式可以快速分析系统慢的原因。Histogram和Summary都是为了能够解决这样问题的存在，通过Histogram和Summary类型的监控指标我们可以快速了解监控样本的分布情况。

直方图



Histogram在一段时间范围内对数据进行采样（通常是请求持续时间或响应大小等），并将其计入可配置的存储桶（bucket）中，后续可通过指定区间筛选样本，也可以统计样本总数，最后一般将数据展示为直方图。

Histogram类型的样本会提供三种指标（假设指标名称为）：

- 样本的值分布在 bucket 中的数量，命名为 `<basename>_bucket{le="<上边界>"}`。解释的更通俗易懂一点，这个值表示指标值小于等于上边界的所有样本数量。

```
1 // 在总共2次请求当中。http 请求响应时间 <=0.005 秒 的请求次数为0
2 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.005",} 0.0
3 // 在总共2次请求当中。http 请求响应时间 <=0.01 秒 的请求次数为0
4 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.01",} 0.0
5 // 在总共2次请求当中。http 请求响应时间 <=0.025 秒 的请求次数为0
6 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.025",} 0.0
7 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.05",} 0.0
8 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.075",} 0.0
9 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.1",} 0.0
10 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.25",} 0.0
11 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.5",} 0.0
12 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="0.75",} 0.0
13 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="1.0",} 0.0
14 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="2.5",} 0.0
15 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
  "GET",code="200",le="5.0",} 0.0
```

```

15 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
   "GET",code="200",le="7.5",} 2.0
16 // 在总共2次请求当中。http 请求响应时间 <=10 秒 的请求次数为 2
17 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
   "GET",code="200",le="10.0",} 2.0
18 io_namespace_http_requests_latency_seconds_histogram_bucket{path="/",method=
   "GET",code="200",le="+Inf",} 2.0
19

```

- 所有样本值的大小总和，命名为 `<basename>_sum`。

```

1 // 实际含义： 发生的2次 http 请求总的响应时间为 13.107670803000001 秒
2 io_namespace_http_requests_latency_seconds_histogram_sum{path="/",method="GET
  ",code="200",} 13.107670803000001

```

- 样本总数，命名为 `<basename>_count`。值和 `<basename>_bucket{le="+Inf"}` 相同。

```

1 // 实际含义： 当前一共发生了 2 次 http 请求
2
   io_namespace_http_requests_latency_seconds_histogram_count{path="/",method="G
   ET",code="200",} 2.0

```

bucket 可以理解为是对数据指标值域的一个划分，划分的依据应该基于数据值的分布。注意后面的采样点是包含前面的采样点的，假设 `xxx_bucket{...,le="0.01"}` 的值为 10，而 `xxx_bucket{...,le="0.05"}` 的值为 30，那么意味着这 30 个采样点中，有 10 个是小于 10 ms 的，其余 20 个采样点的响应时间是介于 10 ms 和 50 ms 之间的。

Summary摘要

Summary即概率图，类似于Histogram，常用于跟踪与时间相关的数据。典型的应用包括请求持续时间、响应大小等。Summary同样提供样本的count和sum功能；还提供quantiles功能，可以按百分比划分跟踪结果，例如，quantile取值0.95，表示取样本里的95%数据。Histogram需要通过`<basename>bucket`计算quantile，而Summary直接存储了quantile的值。

Jobs和Instances

在Prometheus中，任何被采集的目标，即每一个暴露监控样本数据的HTTP服务都称为一个实例instance，通常对应于单个进程。而具有相同采集目的实例集合称为作业job。

Prometheus快速开始

使用二进制文件部署

- 打开下载网址<https://prometheus.io/download/>，下载自己想要的版本

```

1 [root@server1 ~]# wget
   https://github.com/prometheus/prometheus/releases/download/v2.25.0/prometheus
   -2.25.0.linux-amd64.tar.gz

```

- 获取软件包的哈希值，与官网提供的软件包的哈希值进行对比，保证下载的Prometheus软件包的完整性


```
1 [root@server1 ~]# sha256sum prometheus-2.25.0.linux-amd64.tar.gz
2 d163e41c56197425405e836222721ace8def3f120689fe352725fe5e3ba1a69d prometheus-
  2.25.0.linux-amd64.tar.gz
```

- 解压缩二进制软件包到指定的安装目录，运行Prometheus

```
1 [root@server1 ~]# mkdir /data
2 [root@server1 ~]# tar -zxvf prometheus-2.25.0.linux-amd64.tar.gz -C /data/
3 [root@server1 ~]# cd /data/
4 [root@server1 data]# chown -R root:root prometheus-2.25.0.linux-amd64
5 [root@server1 data]# ln -sv prometheus-2.25.0.linux-amd64 prometheus
6 "prometheus" -> "prometheus-2.25.0.linux-amd64"
```

- 启动Prometheus，会输出如下信息，此时当终端关闭或者按下ctrl + c服务会自动关闭

```
1 [root@server1 prometheus]# ./prometheus
2 level=info ts=2021-02-28T06:03:36.885Z caller=main.go:366 msg="No time or
  size retention was set so using the default time retention" duration=15d
3 level=info ts=2021-02-28T06:03:36.885Z caller=main.go:404 msg="Starting
  Prometheus" version="(version=2.25.0, branch=HEAD,
  revision=a6be548dbc17780d562a39c0e4bd0bd4c00ad6e2)"
4 level=info ts=2021-02-28T06:03:36.885Z caller=main.go:409 build_context="
  (go=go1.15.8, user=root@615f028225c9, date=20210217-14:17:24)"
5 level=info ts=2021-02-28T06:03:36.885Z caller=main.go:410 host_details="
  (Linux 3.10.0-693.el7.x86_64 #1 SMP Tue Aug 22 21:09:27 UTC 2017 x86_64
  server1 (none))"
6 level=info ts=2021-02-28T06:03:36.885Z caller=main.go:411 fd_limits="
  (soft=1024, hard=4096)"
7 level=info ts=2021-02-28T06:03:36.885Z caller=main.go:412 vm_limits="
  (soft=unlimited, hard=unlimited)"
8 level=info ts=2021-02-28T06:03:36.891Z caller=web.go:532 component=web
  msg="Start listening for connections" address=0.0.0.0:9090
9 level=info ts=2021-02-28T06:03:36.895Z caller=main.go:779 msg="Starting TSDB
  ..."
10 level=info ts=2021-02-28T06:03:36.897Z caller=ts_config.go:191
  component=web msg="TLS is disabled." http2=false
11 level=info ts=2021-02-28T06:03:36.930Z caller=head.go:668 component=tsdb
  msg="Replaying on-disk memory mappable chunks if any"
12 level=info ts=2021-02-28T06:03:36.930Z caller=head.go:682 component=tsdb
  msg="On-disk memory mappable chunks replay completed" duration=5.69µs
13 level=info ts=2021-02-28T06:03:36.930Z caller=head.go:688 component=tsdb
  msg="Replaying WAL, this may take a while"
14 level=info ts=2021-02-28T06:03:36.933Z caller=head.go:740 component=tsdb
  msg="WAL segment loaded" segment=0 maxSegment=0
15 level=info ts=2021-02-28T06:03:36.933Z caller=head.go:745 component=tsdb
  msg="WAL replay completed" checkpoint_replay_duration=38.416µs
  wal_replay_duration=2.357106ms total_replay_duration=2.638813ms
16 level=info ts=2021-02-28T06:03:36.934Z caller=main.go:799
  fs_type=XFS_SUPER_MAGIC
17 level=info ts=2021-02-28T06:03:36.934Z caller=main.go:802 msg="TSDB started"
18 level=info ts=2021-02-28T06:03:36.934Z caller=main.go:928 msg="Loading
  configuration file" filename=prometheus.yml
```

- 热加载更新配置

在Prometheus日常维护中，一定会对配置文件prometheus.yml进行再编辑操作，通常对Prometheus服务进行重新启动操作即可完成对配置文件的加载。当然也可以通过动态的热加载来更新prometheus.yml中的配置信息
查看进程id，向进程发送SIGHUP信号

```
1 # kill -HUP pid
```

通过HTTP API发送post请求到-/reload

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

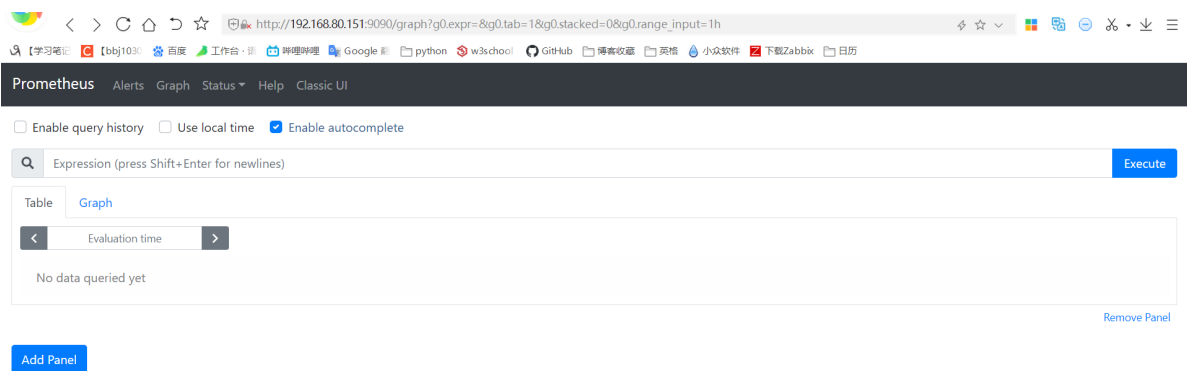
- 检查配置文件的语法正确性

```
1 [root@server1 ~]# cd /data/prometheus
2 [root@server1 prometheus]# ls
3 console_libraries  consoles  data  LICENSE  NOTICE  prometheus
  prometheus.yml  promtool
4 [root@server1 prometheus]# ./promtool check config prometheus.yml
5 Checking prometheus.yml
6 SUCCESS: 0 rule files found
```

- 关闭防火墙和selinux

```
1 [root@server1 prometheus]# systemctl stop firewalld
2 [root@server1 prometheus]# setenforce 0
```

- 浏览器可以正常访问则说明部署Prometheus server成功（端口号默认是9090）



Exporter

简介

在Prometheus的核心组件中，Exporter是重要的组成部分，在实际中监控样本数据的收集都是由Exporter完成的，Prometheus服务器只需要定时从这些Exporter提供的HTTP服务获取数据即可。官方提供了多种常用的Exporter，比如用于对数据库监控的mysqld_exporter和redis_exporter等。

Exporter本质上是将收集的数据转化为对应的文本格式，并提供HTTP接口，供Prometheus定期采集数据。

Exporter类型

- 直接采集型
 - 这类Exporter直接内置了响应的应用程序，用于向Prometheus直接提供target数据支持。这样设计的好处是，可以更好地监控各自系统内部的运行状态，同时也适合更多自定义监控指标的项目实施。
- 间接采集型
 - 原始监控目标并不直接支持Prometheus，需要我们使用Prometheus提供的客户端库编写该监控目标的监控采集数据，用户可以将该程序独立运行，取获取指定的各类监控数据值。例如，由于Linux操作系统自身并不能直接支持Prometheus，用户无法从操作系统层面上直接提供对Prometheus的支持，因此单独提供Node Exporter，还有数据库或网站HTTP应用类等Exporter。

文本数据格式

在Prometheus的监控环境中，所有返回监控样本数据的Exporter程序，均需要遵守Prometheus规范，即基于文本的数据格式，其特点是具有更好的跨平台和可读性。

- 可以使用浏览器，或者通过curl工具来获得采集数据

```
1  # 以HELP开头的行，表示metric的帮助与说明解释，可以包含当前监控指标名称和对应的说明信息
2  # 以TYPE开始的行，表示定义metric的类型
3  # 以非#开头的行即监控样本的数据，包括metric_name{label_name1,label_name2...} value
   [timestamp可选项]
4
5  [root@server1 ~]# curl 192.168.80.151:9090/metrics
6  # HELP go_gc_duration_seconds A summary of the pause duration of garbage
   collection cycles.
7  # TYPE go_gc_duration_seconds summary
8  go_gc_duration_seconds{quantile="0"} 9.4601e-05
9  go_gc_duration_seconds{quantile="0.25"} 0.000141153
10 go_gc_duration_seconds{quantile="0.5"} 0.000416738
11 go_gc_duration_seconds{quantile="0.75"} 0.001050261
12 go_gc_duration_seconds{quantile="1"} 0.008308442
13 go_gc_duration_seconds_sum 0.014675204
14 go_gc_duration_seconds_count 13
15 # HELP go_goroutines Number of goroutines that currently exist.
16 # TYPE go_goroutines gauge
17 go_goroutines 32
18 # HELP go_info Information about the Go environment.
19 # TYPE go_info gauge
20 go_info{version="go1.15.8"} 1
21 # HELP go_memstats_alloc_bytes Number of bytes allocated and still in use.
22 # TYPE go_memstats_alloc_bytes gauge
23 go_memstats_alloc_bytes 1.6849432e+07
24 # HELP go_memstats_alloc_bytes_total Total number of bytes allocated, even
   if freed.
25 # TYPE go_memstats_alloc_bytes_total counter
26 go_memstats_alloc_bytes_total 6.016028e+07
27 # HELP go_memstats_buck_hash_sys_bytes Number of bytes used by the profiling
   bucket hash table.
28 # TYPE go_memstats_buck_hash_sys_bytes gauge
29 go_memstats_buck_hash_sys_bytes 1.461736e+06
30 # HELP go_memstats_frees_total Total number of frees.
31 # TYPE go_memstats_frees_total counter
```

```

32 go_memstats_frees_total 219435
33 # HELP go_memstats_gc_cpu_fraction The fraction of this program's available
    CPU time used by the GC since the program started.
34 # TYPE go_memstats_gc_cpu_fraction gauge
35 go_memstats_gc_cpu_fraction 3.012333999853177e-05
36 # HELP go_memstats_gc_sys_bytes Number of bytes used for garbage collection
    system metadata.
37 # TYPE go_memstats_gc_sys_bytes gauge
38 go_memstats_gc_sys_bytes 5.719272e+06

```

Linux主机监控

Prometheus社区很活跃，提供了非常多类型的Exporter。可以在官网中找到自己想要的Exporter并进行下载<https://prometheus.io/download/>

由于Linux操作系统自身并不支持Prometheus，所以Prometheus官方提供了go语言编写的Node Exporter来实现对Linux操作系统主机的监控数据采集。它提供了系统内部几乎所有的标准指标，如cpu、内存、磁盘空间、磁盘I/O、系统负载和网络带宽。另外它还提供了由内核公开的大量额外监控指标，从负载平均到主板温度等。

- 下载二进制包，解压缩

```

1 [root@server2 ~]# wget
    https://github.com/prometheus/node_exporter/releases/download/v1.1.1/node_ex
    porter-1.1.1.linux-amd64.tar.gz
2
3 [root@server2 ~]# mkdir /data
4 [root@server2 ~]# tar -zxvf node_exporter-1.1.1.linux-amd64.tar.gz -C /data/
5 node_exporter-1.1.1.linux-amd64/
6 node_exporter-1.1.1.linux-amd64/LICENSE
7 node_exporter-1.1.1.linux-amd64/NOTICE
8 node_exporter-1.1.1.linux-amd64/node_exporter
9 [root@server2 ~]# cd /data/
10 [root@server2 data]# chown -R root:root node_exporter-1.1.1.linux-amd64
11 [root@server2 data]# ln -sv node_exporter-1.1.1.linux-amd64 node_exporter
12 "node_exporter" -> "node_exporter-1.1.1.linux-amd64"

```

- 启动node_exporter

```

1 [root@server2 ~]# cd /data/node_exporter
2 [root@server2 node_exporter]# ls
3 LICENSE node_exporter NOTICE
4 [root@server2 node_exporter]# ./node_exporter

```

- 与Prometheus server集成

当启动node_exporter开始工作时，node_exporter和Prometheus server还没有进行关联，二者各自独立没有关联。

可以在Prometheus server中，找到主机目录，找到主配置文件，使用其中的静态配置功能static_configs来采集node_exporter提供的数据

- 主配置文件介绍

```

1 # 配置文件解释
2 global:

```

```

3   scrape_interval: 每次数据采集的时间间隔，默认为1分钟
4   scrape_timeout: 采集请求超时时间，默认为10秒
5   evaluation_interval: 执行rules的频率，默认为1分钟
6   scrape_configs: 主要用于配置被采集数据节点操作，每一个采集配置主要由以下几个参数
7     job_name: 全局唯一名称
8     scrape_interval: 默认等于global内设置的参数，设置后可以覆盖global中的值
9     scrape_timeout: 默认等于global内设置的参数
10    metrics_path: 从targets获取metric的HTTP资源路径，默认是/metrics
11    honor_labels: Prometheus如何处理标签之间的冲突。若设置为True，则通过保留变迁来解决冲突；若设置为false，则通过重命名；
12    scheme: 用于请求的协议方式，默认是http
13    params: 数据采集访问时HTTP URL设定的参数
14    relabel_configs: 采集数据重置标签配置
15    metric_relabel_configs: 重置标签配置
16    sample_limit: 对每个被已知样本数量的每次采集进行限制，如果超过限制，该数据将被视为失败。默认值为0，表示无限制
17    -----
18    [root@server1 prometheus]# cat prometheus.yml
19    # my global config
20    global:
21      scrape_interval: 15s # Set the scrape interval to every 15 seconds.
22      Default is every 1 minute.
23      evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is
24      every 1 minute.
25      # scrape_timeout is set to the global default (10s).
26
27    # Alertmanager configuration
28    alerting:
29      alertmanagers:
30        - static_configs:
31          - targets:
32            # - alertmanager:9093
33
34    # Load rules once and periodically evaluate them according to the global
35    'evaluation_interval'.
36    rule_files:
37      # - "first_rules.yml"
38      # - "second_rules.yml"
39
40    # A scrape configuration containing exactly one endpoint to scrape:
41    # Here it's Prometheus itself.
42    scrape_configs:
43      # The job name is added as a label `job=<job_name>` to any timeseries
44      scraped from this config.
45      - job_name: 'prometheus'
46
47        # metrics_path defaults to '/metrics'
48        # scheme defaults to 'http'.
49
50        static_configs:
51          - targets: ['localhost:9090']

```

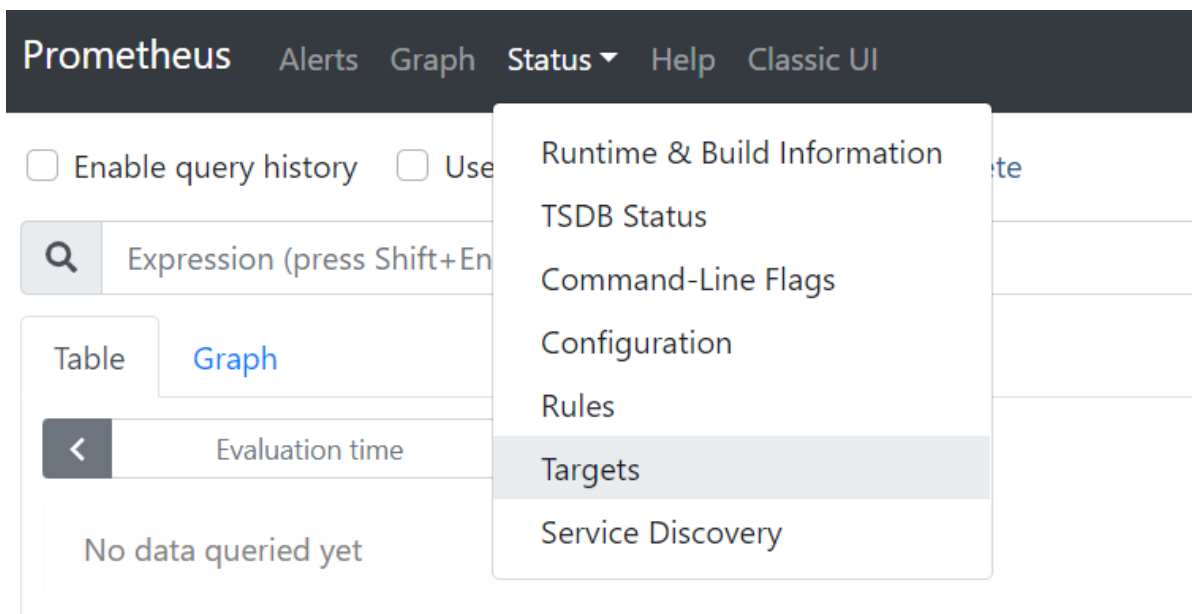
- 在默认配置文件的基础上，重新编辑配置文件，添加job与node_exporter进行关联

```

1 [root@server1 prometheus]# vim prometheus.yml
2 scrape_configs:
3   # The job name is added as a label `job=<job_name>` to any timeseries
   scraped from this config.
4   - job_name: 'prometheus'
5
6   # metrics_path defaults to '/metrics'
7   # scheme defaults to 'http'.
8
9   static_configs:
10    - targets: ['localhost:9090']
11 - job_name: 'node_exporter'
12   static_configs:
13    - targets: [192.168.80.152:9100]

```

重启服务即可成功关联



Prometheus Alerts Graph Status Help Classic UI					
Targets					
All Unhealthy					
node_exporter (1/1 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.80.152:9100/metrics	UP	instance="192.168.80.152:9100" job="node_exporter"	19.346s	37.632ms	
prometheus (1/1 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://localhost:9090/metrics	UP	instance="localhost:9090" job="prometheus"	6.359s	16.023ms	

或者通过搜索up找到node_exporter

☐ Enable query history ☐ Use local time ☒ Enable autocomplete up

Table

Graph

<

Evaluation time


>

`up(instance="192.168.80.152:9100", job="node_exporter")``up(instance="localhost:9090", job="prometheus")`

metrics数据采

- cpu数据采集

对于cpu数据采集的主要监控指标是node_cpu_seconds_total

Prometheus	Alerts	Graph	Status ▾	Help	Classic UI
<input type="checkbox"/> Enable query history	<input type="checkbox"/> Use local time	<input checked="" type="checkbox"/> Enable autocomplete			
 node_cpu_seconds_total					Ex
Table	Graph				Load time: 30ms Resolution: 14s Result
<	2021-02-28 09:41:51	x	>		
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="idle")					7499.52
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="iowait")					2.5
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="irq")					0
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="nice")					0.03
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="softirq")					0.15
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="steal")					0
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="system")					3.01
node_cpu_seconds_total(cpu="0", instance="192.168.80.152:9100", job="node_exporter", mode="user")					0.5
node_cpu_seconds_total(cpu="1", instance="192.168.80.152:9100", job="node_exporter", mode="idle")					7502.76
node_cpu_seconds_total(cpu="1", instance="192.168.80.152:9100", job="node_exporter", mode="iowait")					0.65
node_cpu_seconds_total(cpu="1", instance="192.168.80.152:9100", job="node_exporter", mode="irq")					0
node_cpu_seconds_total(cpu="1", instance="192.168.80.152:9100", job="node_exporter", mode="nice")					0.09
node_cpu_seconds_total(cpu="1", instance="192.168.80.152:9100", job="node_exporter", mode="softirq")					0.39

可以通过PromQL（后面会介绍这种查询语言）表达式进行查询，计算每核cpu每秒的空闲时间，然后对主机上的所有cpu求平均值

```
avg without(cpu,mode) (rate(node_cpu_seconds_total {mode="idle"} [1m]))
```

- 内存信息采集

```
1 [root@server1 prometheus]# free -b
2          total          used         free        shared  buff/cache
3 Mem:    1911857152  164589568  1512689664    9031680   234577920
4 Swap:    2147479552           0   2147479552
```

PrometheusAlertsGraphStatus▼HelpClassic UI

☐ Enable query history☐ Use local time☒ Enable autocomplete

Qnode_memory_MemTotal_bytes

Execute

TableGraph

Load time: 22msResolution: 14sResult series: 1

<2021-02-28 09:41:51x>

node_memory_MemTotal_bytes(instance="192.168.80.152:9100",job="node_exporter")1911857152

Remove Panel

PrometheusAlertsGraphStatus▼HelpClassic UI

☐ Enable query history☐ Use local time☒ Enable autocomplete

Qnode_memory_MemFree_bytes

Execute

TableGraph

Load time: 15msResolution: 14sResult series: 1

<2021-02-28 09:41:51x>

node_memory_MemFree_bytes(instance="192.168.80.152:9100",job="node_exporter")1396269056

Remove Panel

PrometheusAlertsGraphStatus▼HelpClassic UI

☐ Enable query history☐ Use local time☒ Enable autocomplete

Qnode_memory_MemAvailable_bytes

Execute

TableGraph

Load time: 22msResolution: 14sResult series: 1

<2021-02-28 09:41:51x>

node_memory_MemAvailable_bytes(instance="192.168.80.152:9100",job="node_exporter")1562206208

Remove Panel

- 磁盘信息采集

PrometheusAlertsGraphStatus▼HelpClassic UI

☐ Enable query history☐ Use local time☒ Enable autocomplete

Qnode_disk_io_time_seconds_total

Execute

TableGraph

Load time: 21msResolution: 14sResult series: 4

<2021-02-28 09:41:51x>

node_disk_io_time_seconds_total(device="dm-0",instance="192.168.80.152:9100",job="node_exporter")	5.564
node_disk_io_time_seconds_total(device="dm-1",instance="192.168.80.152:9100",job="node_exporter")	0.003
node_disk_io_time_seconds_total(device="sda",instance="192.168.80.152:9100",job="node_exporter")	5.808
node_disk_io_time_seconds_total(device="sr0",instance="192.168.80.152:9100",job="node_exporter")	0.009000000000000001

Remove Panel

- 文件系统采集

PrometheusAlertsGraphStatus▼HelpClassic UI

☐ Enable query history☐ Use local time☒ Enable autocomplete

Qnode_filesystem_size_bytes

Execute

TableGraph

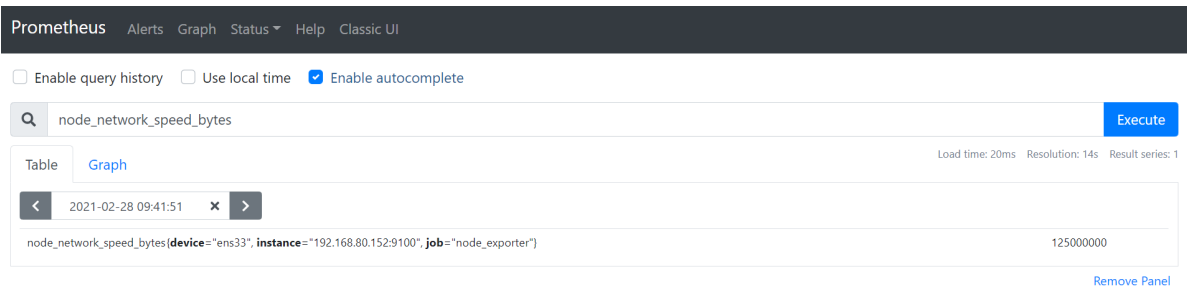
Load time: 18msResolution: 14sResult series: 5

<2021-02-28 09:41:51x>

node_filesystem_size_bytes(device="/dev/mapper/centos-root",fstype="xfs",instance="192.168.80.152:9100",job="node_exporter",mountpoint="/")	18238930944
node_filesystem_size_bytes(device="/dev/sda1",fstype="xfs",instance="192.168.80.152:9100",job="node_exporter",mountpoint="/boot")	1063256064
node_filesystem_size_bytes(device="rootfs",fstype="rootfs",instance="192.168.80.152:9100",job="node_exporter",mountpoint="/")	18238930944
node_filesystem_size_bytes(device="tmpfs",fstype="tmpfs",instance="192.168.80.152:9100",job="node_exporter",mountpoint="/run")	955928576
node_filesystem_size_bytes(device="tmpfs",fstype="tmpfs",instance="192.168.80.152:9100",job="node_exporter",mountpoint="/run/user/0")	191188992

Remove Panel

- 网络采集



任意一个Exporter都会提供足够多的metric，我们在学习的时候也不需要关心具体有多少metric，每个metric具体意思（其实见名知义大概也可以猜到）

MySQL监控

- 官网下载mysqld_exporter二进制包解压缩

```
1 [root@server2 ~]# wget
https://github.com/prometheus/mysqld_exporter/releases/download/v0.12.1/mysq
ld_exporter-0.12.1.linux-amd64.tar.gz
2 [root@server2 ~]# tar -xzf mysqld_exporter-0.12.1.linux-amd64.tar.gz -C
/data/
3 mysqld_exporter-0.12.1.linux-amd64/
4 mysqld_exporter-0.12.1.linux-amd64/NOTICE
5 mysqld_exporter-0.12.1.linux-amd64/mysqld_exporter
6 mysqld_exporter-0.12.1.linux-amd64/LICENSE
7 [root@server2 ~]# cd /data/
8 [root@server2 data]# chown root:root mysqld_exporter-0.12.1.linux-amd64 -R
9 [root@server2 data]# ln -sv mysqld_exporter-0.12.1.linux-amd64
mysqld_exporter
10 "mysqld_exporter" -> "mysqld_exporter-0.12.1.linux-amd64"
11 [root@server2 ~]# sha256sum mysqld_exporter-0.12.1.linux-amd64.tar.gz
12 133b0c281e5c6f8a34076b69ade64ab6cac7298507d35b96808234c4aa26b351
mysqld_exporter-0.12.1.linux-amd64.tar.gz
```

- 创建MySQL授权用户

```
1 [root@server2 ~]# yum install mariadb-server.x86_64 -y
2 [root@server2 ~]# systemctl start mariadb
3 [root@server2 ~]# mysql_secure_installation
4 [root@server2 ~]# mysql -uroot -p
5 Enter password:
6 Welcome to the MariaDB monitor.  Commands end with ; or \g.
7 Your MariaDB connection id is 10
8 Server version: 5.5.68-MariaDB MariaDB Server
9
10 Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
11
12 Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.
13
14 MariaDB [(none)]> create user 'mysqld_exporter'@'%' identified by '1';
15 Query OK, 0 rows affected (0.00 sec)
16
17 MariaDB [(none)]> grant process,replication client,select on *.* to
'mysqld_exporter'@'%';
18 Query OK, 0 rows affected (0.00 sec)
19
```

```

20 MariaDB [(none)]> flush privileges;
21 Query OK, 0 rows affected (0.00 sec)
22 MariaDB [(none)]> select host,user from mysql.user;
23 +-----+-----+
24 | host      | user      |
25 +-----+-----+
26 | %         | mysql_exporter |
27 | 127.0.0.1 | root      |
28 | ::1       | root      |
29 | localhost | root      |
30 +-----+-----+
31 4 rows in set (0.00 sec)

```

- 配置数据库认证，并启动服务，默认端口号是9104

```

1 [root@server2 mysql_exporter]# pwd
2 /data/mysql_exporter
3
4 [root@server2 mysql_exporter]# vim .mysql_exporter.cnf
5 [client]
6 user=mysql_exporter
7 password=1
8
9 [root@server2 mysql_exporter]# ./mysql_exporter --config.my-
  cnf='.mysql_exporter.cnf'

```

- Prometheus集成

```

1 # 配置文件
2 - job_name: 'mysql_exporter'
3   scrape_interval: 10s
4   static_configs:
5     - targets: [192.168.80.152:9104]

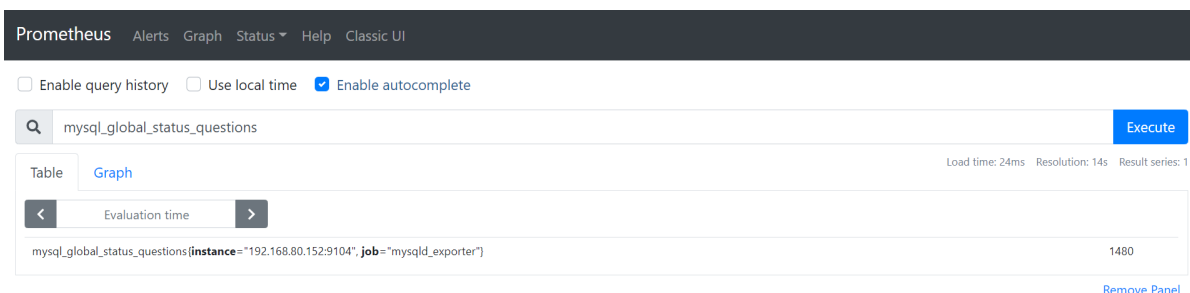
```

- 重新启动服务即可

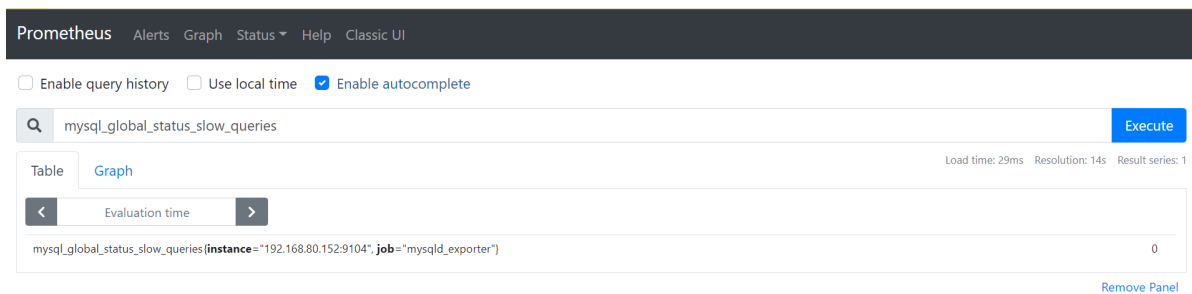
metrics数据采集

MySQL数据库的性能状态监控内容非常多，但通常必不可少的内容包括查询吞吐量、查询执行性能、连接情况、缓冲池使用情况等。

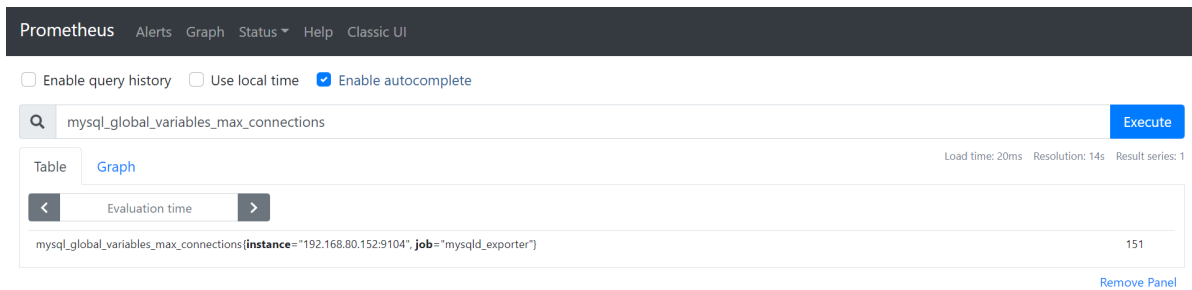
- 查询吞吐量，MySQL客户端应用程序发送的所有查询语句，该计数器都是递增的



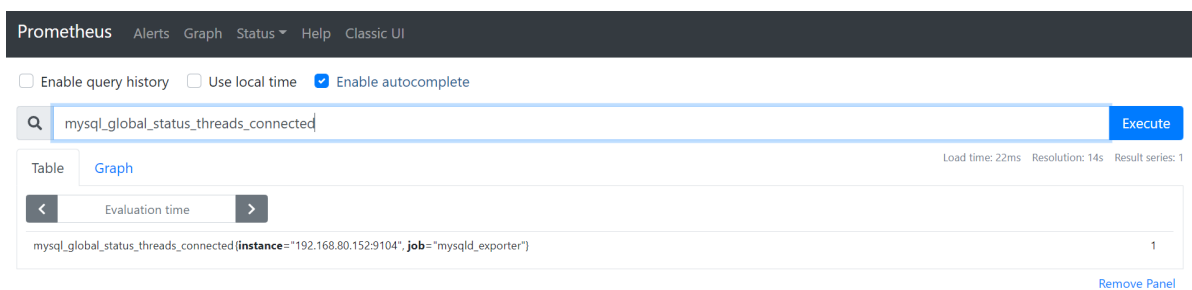
- 查询执行性能，每当查询时间超过预先设定的慢查询时间计数器都会递增



- 连接情况，查询MySQL最大连接数，防止连接数量过大导致服务器过载运行



- 查看当前连接数



- 缓存池使用情况



服务发现

Prometheus服务发现能够自动化检测分类，并且能够识别新目标和变更目标。也就是说，可以自动发现并监控目标或变更目标，动态进行数据采集和处理。

基于文件的服务发现

- 准备JSON格式的文件

```
1 [root@server1 ~]# cd /data/prometheus
2 [root@server1 prometheus]# mkdir targets
3 [root@server1 prometheus]# vim targets/dev_node.json
4 [
5 {
6     "targets": "192.168.80.152:9100",
7     "labels": {
```

```
8      "env": "dev_webgame"
9    }
10  }
11  ]
12  -----
13  或者这里是准备yaml文件，那么下面相应的配置文件需要与yaml匹配
14  - targets:
15    - "192.168.80.152:9100"
```

- 修改配置文件

```
1 [root@server1 prometheus]# vim /data/prometheus/prometheus.yml
2   - job_name: 'node_service_discovery'
3     file_sd_configs:
4       - files:
5         - targets/*.json
6         refresh_interval: 60m
```

- 重新启动服务

扩展：这是基于文件发现，还有基于consul基于dns的服务发现，这个自行扩展。

PromQL

Prometheus提供了一种功能强大的表达式语言PromQL（Prometheus Query Language）。Prometheus允许用户实时选择和汇聚时间序列数据，是Prometheus自己开发的数据查询语言，使用这个查询语言能够进行各种聚合、分析和计算，使管理员能够根据指标更好地了解系统性能。

时序数据库

首先Prometheus是一款时序数据库TSDB，它结合生态系统内的其他组件例如Pushgateway、Alertmanager等可构成一个完整的IT监控系统。

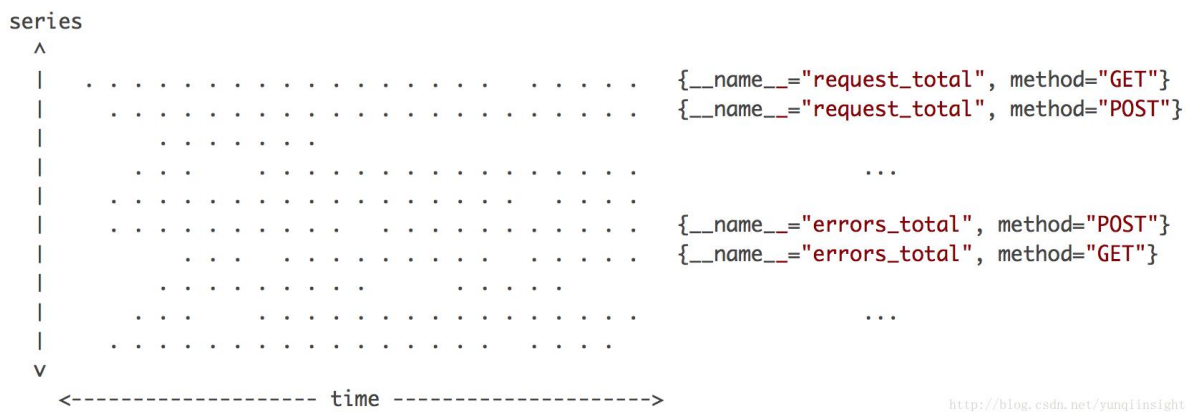
时序数据库的特点如下：

- 数据写入特点——写入平稳、持续、高并发高吞吐；写多读少，在写操作上数据上能达到95%以上；无更新时写入最近生成的数据
- 数据查询特点——按时间范围读取一段时间的数据；对最近生成的数据读取概率高，对历史数据查询概率低；按照数据点的不同密集度实现多精度查询
- 数据存储特点——数据存储量比较大；具有时效性，数据通常会有一个保存周期，多精度数据存储

对时序数据库的基本要求如下：

- 能够支持高并发、高吞吐的写入
- 交互及的聚合查询，能够达到低查询延迟
- 依据场景设计可以支持海量数据存储
- 在线应用服务场景中，需要高可用架构支持
- 针对写入和存储量的要求，应用环境底层需要分布式架构支持

时序数据



时间序列数据：按照时间顺序记录系统、设备状态变化的数据，每个数据称为一个样本

数据采集以特定的时间周期进行，随着时间的流逝，将这些样本数据记录下来，将生成一个离散的样本数据序列

将该序列称作为**向量**，而将多个序列放在同一个坐标系内（以时间为横轴，以序列为纵轴，将形成一个有数据点组成的矩阵）

- **即时向量**：特定或全部的时间序列上的集合，具有相同时间戳的一组样本称之为即时向量
- **范围向量**：特定或全部的时间序列上的集合，在指定的同一时间范围内的所有样本值称之为范围向量

时间序列选择器

即时向量选择器

即时向量选择器由两部分组成

- **指标名称**：用于限定特定指标下的时间序列，即负责过滤指标，可选
- **匹配器**：或称为标签选择器，用于过滤时间序列上的标签，定义在{}中
- 常见使用举例
 - prometheus_http_requests_total, 仅给定指标名称
 - {job="node_exporter"}, 仅给定匹配器
 - up{job="node_exporter"}, 指标名称和匹配器的组合

匹配器用于定义标签过滤的条件，目前支持如下四种

- **=**：相等匹配模式，用来指定返回的时间序列与指定的标签完全匹配
- **!=**：反向匹配模式，即不等于模式
- **=~**：正则表达式匹配模式
- **!~**：反向正则表达式

范围向量选择器

同即时向量选择器唯一不同的地方在于，范围向量选择器需要在表达式后紧跟一个方括号[]来表达需要在时序上返回的样本所处的时间范围

时间范围：以当前时间为基准点，指向过去一个特定的时间长度，例如[5m]，便是指过去5分钟之内

可用的时间单位有：

- ms
- s
- m
- h

- d
- w
- y

必须使用整数时间，例如1h30m，但不允许使用1.5h

需要注意的是，范围向量选择器返回的是一定时间范围内的数据样本，虽然不同时间序列的数据抓点时间点相同，但特们的时间戳并不会严格对齐。多个target上的数据抓取需要分散在抓取时间点的周围，他们的时间戳并不会严格对齐，目的是为了均衡Prometheus的负载

因而，Prometheus在趋势上准确，但并非绝对精准

偏移量修改器

默认情况下，即时向量选择器和范围向量选择器都以当前时间为基准，而偏移量修改器能够修改该基准

例如：

`up{job="node_exporter"}[5m]`表示的是获取过去5分钟的即时样本

`up{job="node_exporter"}[5m] offset 1d`表示的是获取过去1天的即时样本

PromQL操作符

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)

布尔运算

目前，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)

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，交集)
- `or` (或者，并集)
- `unless` (，差集)

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

操作符优先级

在PromQL操作符中优先级由高到低依次为：

1. `^`
2. `*`, `/`, `%`
3. `+`, `-`
4. `==`, `!=`, `<=`, `<`, `>=`, `>`
5. `and`, `unless`
6. `or`

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：

```
1 | <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。

例如：

```
1 | sum(http_requests_total) without (instance)
```

等价于

```
1 | sum(http_requests_total) by (code,handler,job,method)
```

如果只需要计算整个应用的HTTP请求总量，可以直接使用表达式：

```
1 | sum(http_requests_total)
```

count_values用于时间序列中每一个样本值出现的次数。count_values会为每一个唯一的样本值输出一个时间序列，并且每一个时间序列包含一个额外的标签。

例如：

```
1 | count_values("count", http_requests_total)
```

topk和bottomk则用于对样本值进行排序，返回当前样本值前n位，或者后n位的时间序列。

获取HTTP请求数前5位的时序样本数据，可以使用表达式：

```
1 | topk(5, http_requests_total)
```

quantile用于计算当前样本数据值的分布情况quantile(ϕ , express)其中 $0 \leq \phi \leq 1$ 。

例如，当 ϕ 为0.5时，即表示找到当前样本数据中的中位数：

```
1 | quantile(0.5, http_requests_total)
```

向量匹配

one-to-one

一对一向量匹配模式，它从运算符的两侧表达式中获取**即时向量**，依次比较并找到一对唯一条目进行匹配，如果两个条目具有完全相同的标签和对应的值，则他们匹配。

在操作符两边表达式标签不一致的情况下，可以使用on(label list)或者ignoring(label list) 来修改便签的匹配行为。使用ignoring可以在匹配时忽略某些便签。而on则用于将匹配行为限定在某些便签之内。

```
1 | <vector expr> <bin-op> ignoring(<label list>) <vector expr>
2 | <vector expr> <bin-op> on(<label list>) <vector expr>
```

例如当存在样本：

```
1 | method_code:http_errors:rate5m{method="get", code="500"} 24
2 | method_code:http_errors:rate5m{method="get", code="404"} 30
3 | method_code:http_errors:rate5m{method="put", code="501"} 3
4 | method_code:http_errors:rate5m{method="post", code="500"} 6
5 | method_code:http_errors:rate5m{method="post", code="404"} 21
6 |
7 | method:http_requests:rate5m{method="get"} 600
8 | method:http_requests:rate5m{method="del"} 34
9 | method:http_requests:rate5m{method="post"} 120
```

使用PromQL表达式：

```
1 | method_code:http_errors:rate5m{code="500"} / ignoring(code)
   method:http_requests:rate5m
```

该表达式会返回在过去5分钟内，HTTP请求状态码为500的在所有请求中的比例。如果没有使用ignoring(code)，操作符两边表达式返回的瞬时向量中将找不到任何一个标签完全相同的匹配项。

因此结果如下：

```
1 {method="get"} 0.04 // 24 / 600
2 {method="post"} 0.05 // 6 / 120
```

many-to-one和one-to-many

多对一和一对多的匹配模式，可以理解为向量元素中的一个样本数据匹配到了多个样本数据标签。在使用该匹配模式时，需要使用group_left或者group_right修饰符明确指定哪一个向量具有更高的基数，也就是说左或者右决定了哪边的向量具有较高的子集

```
1 <vector expr> <bin-op> ignoring(<label list>) group_left(<label list>)
  <vector expr>
2 <vector expr> <bin-op> ignoring(<label list>) group_right(<label list>)
  <vector expr>
3 <vector expr> <bin-op> on(<label list>) group_left(<label list>) <vector
  expr>
4 <vector expr> <bin-op> on(<label list>) group_right(<label list>) <vector
  expr>
```

多对一和一对多两种模式一定是出现在操作符两侧表达式返回的向量标签不一致的情况。因此需要使用ignoring和on修饰符来排除或者限定匹配的标签列表。

例如,使用表达式：

```
1 method_code:http_errors:rate5m / ignoring(code) group_left
  method:http_requests:rate5m
```

该表达式中，左向量 method_code:http_errors:rate5m 包含两个标签method和code。而右向量 method:http_requests:rate5m 中只包含一个标签method，因此匹配时需要使用ignoring限定匹配的标签为code。在限定匹配标签后，右向量中的元素可能匹配到多个左向量中的元素 因此该表达式的匹配模式为多对一，需要使用group修饰符group_left指定左向量具有更好的基数。

最终的运算结果如下：

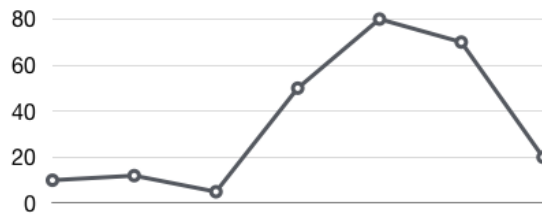
```
1 {method="get", code="500"} 0.04 // 24 / 600
2 {method="get", code="404"} 0.05 // 30 / 600
3 {method="post", code="500"} 0.05 // 6 / 120
4 {method="post", code="404"} 0.175 // 21 / 120
```

内置函数

计算Counter指标增长率

我们知道Counter类型的监控指标其特点是只增不减，在没有发生重置（如服务器重启，应用重启）的情况下其样本值应该是不断增大的。为了能够更直观的表达样本数据的变化剧烈情况，需要计算样本的增长速率。

如下图所示，样本增长率反映出了样本变化的剧烈程度：



通过增长率表示样本的变化情况

`increase(v range-vector)`函数是PromQL中提供的众多内置函数之一。其中参数`v`是一个区间向量，`increase`函数获取区间向量中的第一个后最后一个样本并返回其增长量。因此，可以通过以下表达式Counter类型指标的增长率：

```
1 | increase(node_cpu[2m]) / 120
```

这里通过`node_cpu[2m]`获取时间序列最近两分钟的所有样本，**increase**计算出最近两分钟的增长量，最后除以时间120秒得到**node_cpu**样本在最近两分钟的平均增长率。并且这个值也近似于主机节点最近两分钟内的平均CPU使用率。

除了使用`increase`函数以外，PromQL中还直接内置了`rate(v range-vector)`函数，`rate`函数可以直接计算区间向量`v`在时间窗口内平均增长速率。因此，通过以下表达式可以得到与`increase`函数相同的结果：

```
1 | rate(node_cpu[2m])
```

需要注意的是使用`rate`或者`increase`函数去计算样本的平均增长速率，容易陷入“长尾问题”当中，其无法反应在时间窗口内样本数据的突发变化。例如，对于主机而言在2分钟的时间窗口内，可能在某一个由于访问量或者其它问题导致CPU占用100%的情况，但是通过计算在时间窗口内的平均增长率却无法反应出该问题。

为了解决该问题，PromQL提供了另外一个灵敏度更高的函数`irate(v range-vector)`。`irate`同样用于计算区间向量的计算率，但是其反应出的是**瞬时增长率**。`irate`函数是通过区间向量中最后两个两本数据来计算区间向量的增长速率。这种方式可以避免在时间窗口范围内的“长尾问题”，并且体现出更好的灵敏度，通过`irate`函数绘制的图标能够更好的反应样本数据的瞬时变化状态。

```
1 | irate(node_cpu[2m])
```

`irate`函数相比于`rate`函数提供了更高的灵敏度，不过当需要分析长期趋势或者在告警规则中，`irate`的这种灵敏度反而容易造成干扰。因此在长期趋势分析或者告警中**更推荐使用rate函数**。

预测Gauge指标变化趋势

在一般情况下，系统管理员为了确保业务的持续可用运行，会针对服务器的资源设置相应的告警阈值。例如，当磁盘空间只剩512MB时向相关人员发送告警通知。这种基于阈值的告警模式对于当资源用量是平滑增长的情况下是能够有效的工作的。但是如果资源不是平滑变化的呢？比如有些某些业务增长，存储空间的增长速率提升了高几倍。这时，如果基于原有阈值去触发告警，当系统管理员接收到告警以后可能还没来得及去处理问题，系统就已经不可用了。因此阈值通常来说不是固定的，需要定期进行调整才能保证该告警阈值能够发挥去作用。那么还有没有更好的方法吗？

PromQL中内置的`predict_linear(v range-vector, t scalar)`函数可以帮助系统管理员更好的处理此类情况，`predict_linear`函数可以预测时间序列`v`在`t`秒后的值。它基于简单线性回归的方式，对时间窗口内的样本数据进行统计，从而可以对时间序列的变化趋势做出预测。例如，基于2小时的样本数据，来预测主机可用磁盘空间的是否在4个小时被占满，可以使用如下表达式：

```
1 | predict_linear(node_filesystem_free{job="node"}[2h], 4 * 3600) < 0
```

统计Histogram指标的分位数

在本章的第2小节中，我们介绍了Prometheus的四种监控指标类型，其中Histogram和Summary都可以同于统计和分析数据的分布情况。区别在于Summary是直接在客户端计算了数据分布的分位数情况。而Histogram的分位数计算需要通过`histogram_quantile(ϕ float, b instant-vector)`函数进行计算。其中 ϕ ($0 < \phi < 1$) 表示需要计算的分位数，如果需要计算中位数 ϕ 取值为0.5，以此类推即可。

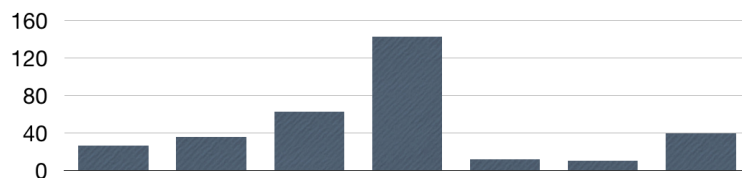
以指标`http_request_duration_seconds_bucket`为例：

```
1 # HELP http_request_duration_seconds request duration histogram
2 # TYPE http_request_duration_seconds histogram
3 http_request_duration_seconds_bucket{le="0.5"} 0
4 http_request_duration_seconds_bucket{le="1"} 1
5 http_request_duration_seconds_bucket{le="2"} 2
6 http_request_duration_seconds_bucket{le="3"} 3
7 http_request_duration_seconds_bucket{le="5"} 3
8 http_request_duration_seconds_bucket{le="+Inf"} 3
9 http_request_duration_seconds_sum 6
10 http_request_duration_seconds_count 3
```

当计算9分位数时，使用如下表达式：

```
1 | histogram_quantile(0.5, http_request_duration_seconds_bucket)
```

通过对Histogram类型的监控指标，用户可以轻松获取样本数据的分布情况。同时分位数的计算，也可以非常方便的用于评判当前监控指标的服务水平。



获取分布直方图的中位数

需要注意的是通过`histogram_quantile`计算的分位数，**并非为精确值**，而是通过`http_request_duration_seconds_bucket`和`http_request_duration_seconds_sum`**近似计算**的结果。

Grafana使用

grafana是一款图形显示非常优秀的工具，支持图表模板导入，支持出Prometheus之外多种数据源（包括MySQL、zabbix、elasticsearch等等）

- 下载安装rpm包，下载网址：<https://grafana.com/grafana/download>

```
1 [root@grafana ~]# wget https://dl.grafana.com/oss/release/grafana-7.5.3-1.x86_64.rpm
2 [root@grafana ~]# yum install grafana-7.5.3-1.x86_64.rpm -y
```

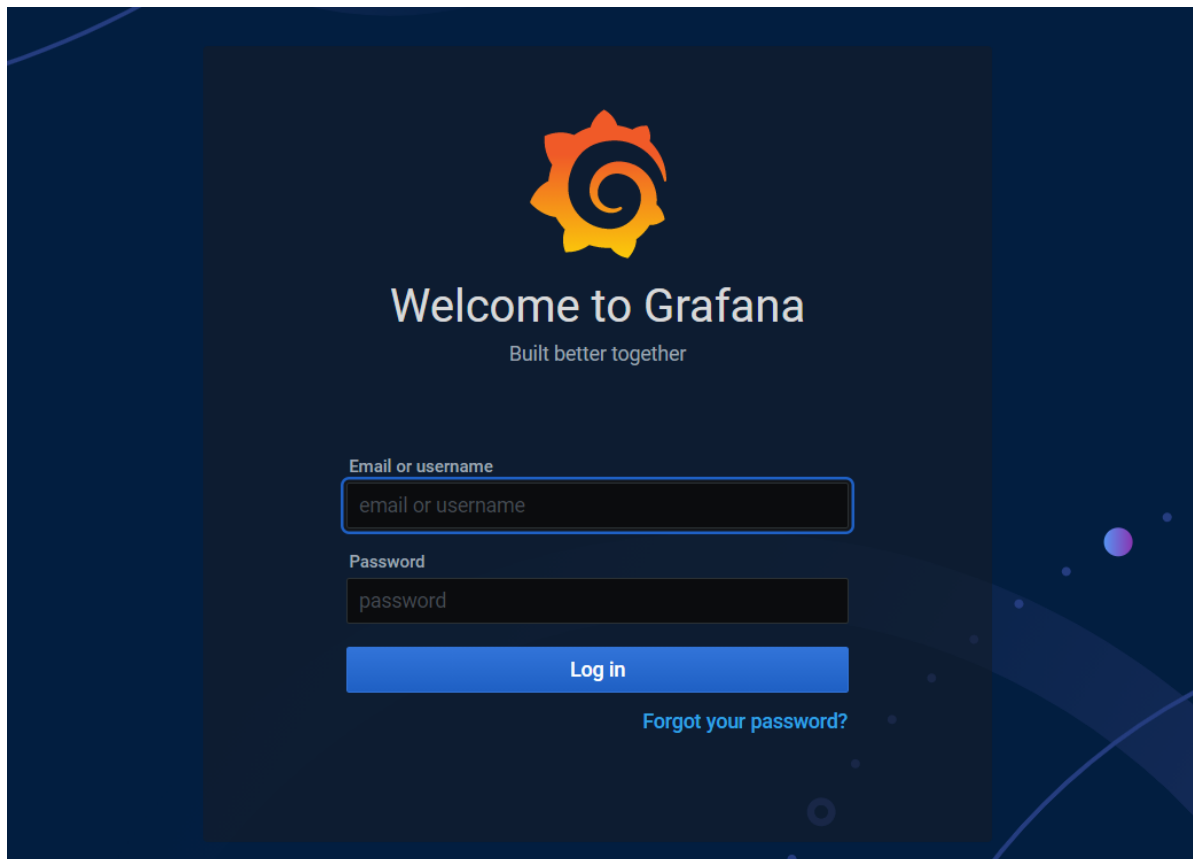
- 启动grafana

```
1 [root@grafana ~]# systemctl start grafana-server.service
2 [root@grafana ~]# systemctl enable grafana-server.service
3 Created symlink from /etc/systemd/system/multi-user.target.wants/grafana-server.service to /usr/lib/systemd/system/grafana-server.service.
```

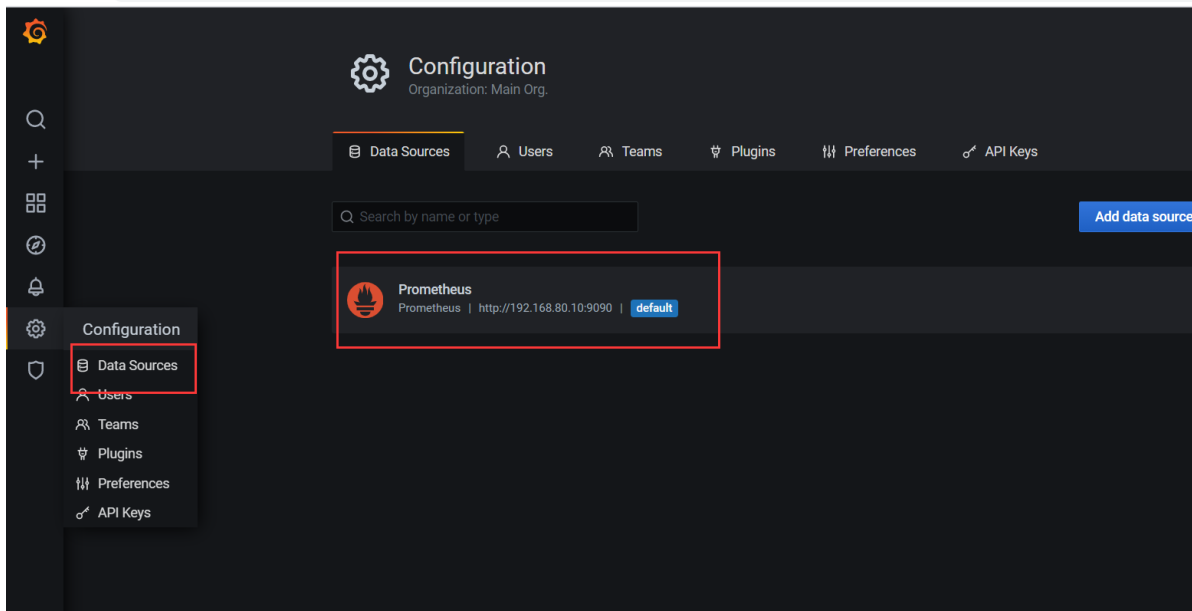
- 检查端口号

```
1 [root@grafana ~]# ss -tanlp | grep 3000
2 LISTEN      0      128      :::3000      :::*
   users:((("grafana-server",pid=1780,fd=12))
```

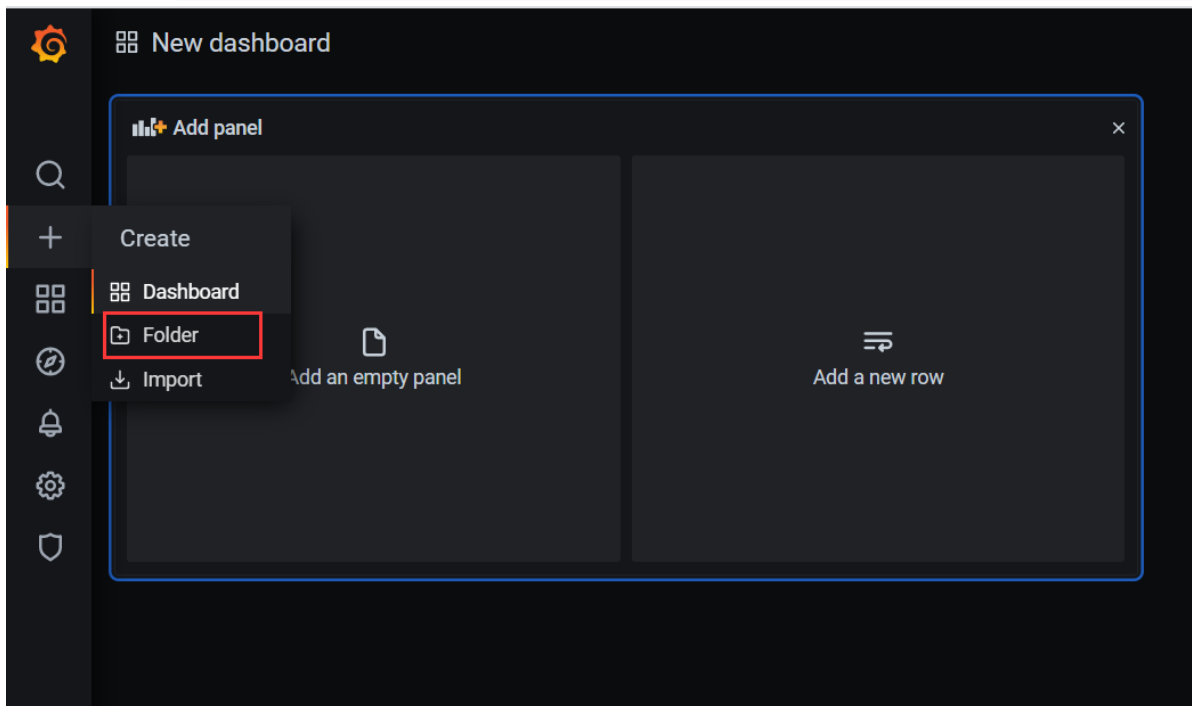
- 首次登陆用户名密码为admin admin



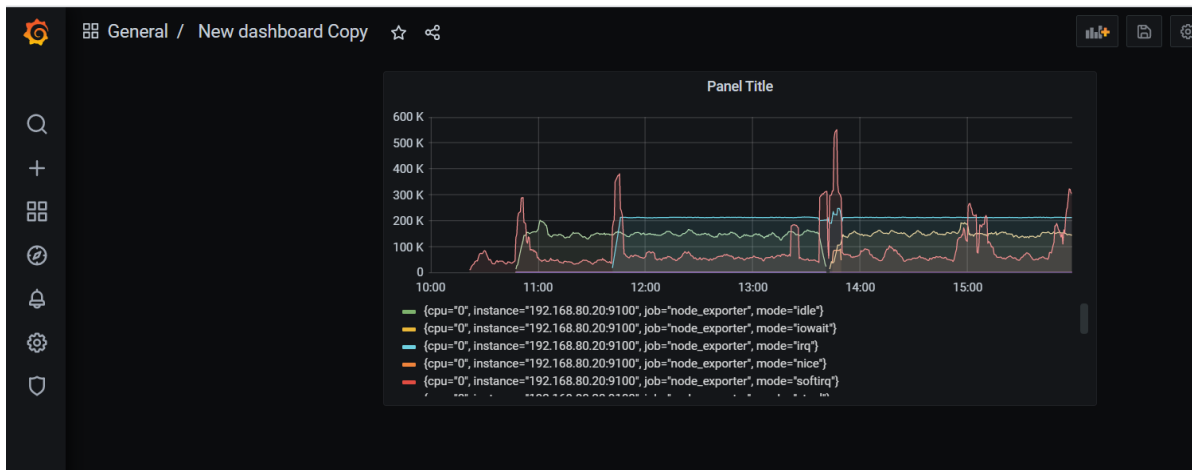
- 添加数据源



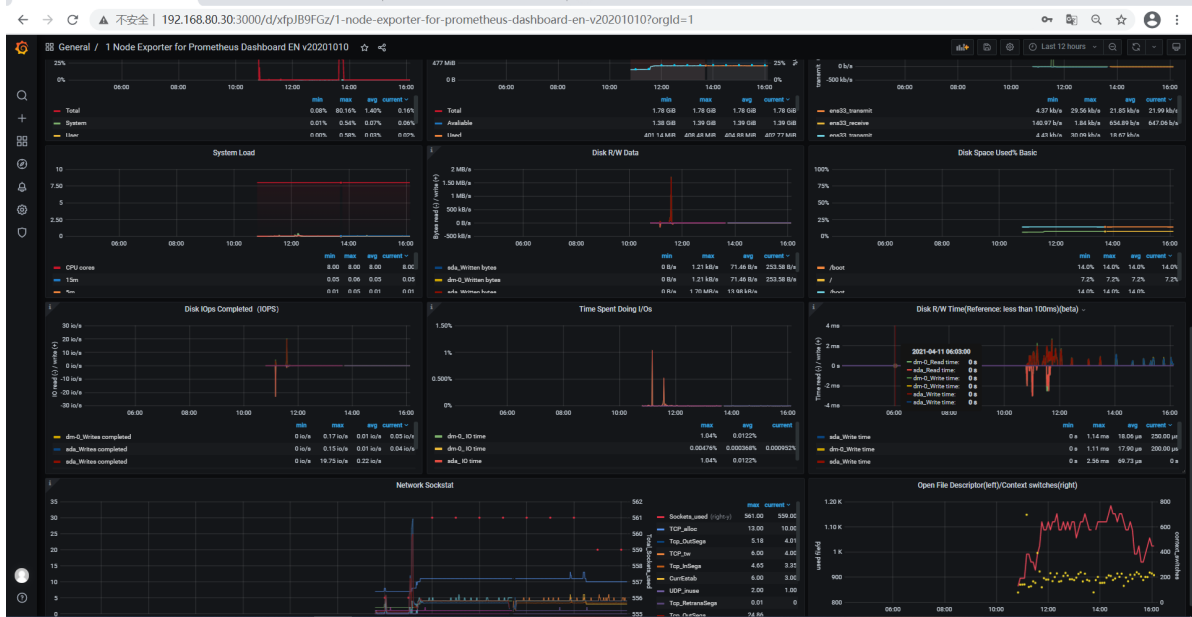
- 添加dashboard



- 添加成功后可以在首页下方看到自己的图



- 人工创建图表比较繁琐，可以直接导入已经存在的模板
- 参考网址：<https://www.cnblogs.com/xuliuzai/p/11134714.html>

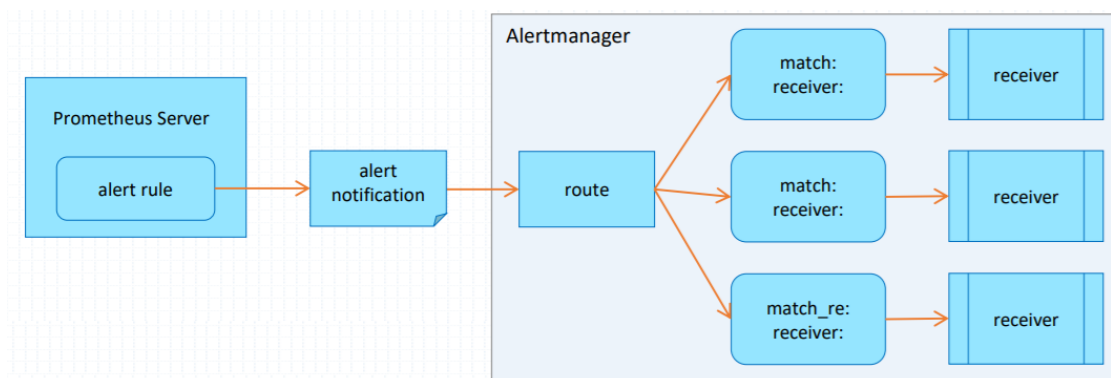


Alertmanager告警

概述

- Prometheus对指标的收集、存储同告警能力分属于Prometheus Server和AlertManager两个独立的组件组成，前者仅仅负责基于告警规则生成告警通知，具体的告警操作则由后者完成；
- AlertManager负责处理由客户端发来的告警通知
 - 客户端通常是Prometheus Server，但也支持来自其他工具的告警
 - AlertManager对告警通知进行分组、去重后根据路由规则将其路由到不同的receiver，如email、企业微信、钉钉等
- 告警逻辑
 - 在AlertManager上定义receiver，他们能够基于某个媒介接收告警信息的特定用户；

- 在Alertmanager上定义路由规则（route），以便将收到的告警通知按需分别进行处理
- 在Prometheus上定义告警规则生成告警通知，发送给Alertmanager



Alertmanager机制

除了基本的告警通知能力外，Alertmanager还支持对告警进行去重分组抑制静默和路由等功能

分组机制

将相似告警合并为单个告警通知的机制，在系统因大面积故障而触发告警是，分组机制能避免用户被大量的告警噪声淹没，进而导致关键信息的隐没

抑制机制

系统中某个组件或服务故障而触发告警通知后，那些依赖于该组件或服务或其他组件或服务也会因此而触发告警，抑制是避免类似的级联告警的一种特性，从而让用户的经历集中于真正的故障所在

静默机制

在一个特定的时间窗口内，便接收到告警通知，Alertmanager也不会真正向用户发送告警行为；通常，在系统例行维护期间，需要激活告警系统的静默特性

部署Alertmanger

- 下载二进制包

```
1 [root@server1 ~]# wget
  https://github.com/prometheus/alertmanager/releases/download/v0.21.0/alertman
  ager-0.21.0.linux-amd64.tar.gz
2 [root@server1 ~]# sha256sum alertmanager-0.21.0.linux-amd64.tar.gz
3 9ccd863937436fd6bfe650e22521a7f2e6a727540988eef515dde208f9aef232
  alertmanager-0.21.0.linux-amd64.tar.gz
```

- 解压缩

```
1 [root@server1 ~]# tar -zxvf alertmanager-0.21.0.linux-amd64.tar.gz -C /data/
2 [root@server1 ~]# cd /data/
3 [root@server1 data]# ln -sv alertmanager-0.21.0.linux-amd64 alertmanager
4 [root@server1 data]# chown -R root:root alertmanager-0.21.0.linux-amd64
```

- 查看配置文件

参考文档: <https://blog.csdn.net/aixiaoyang168/article/details/98474494>

- 默认配置文件

```
1 [root@server1 alertmanager]# cat alertmanager.yml
2 global:                                # 全局配置模块
3   resolve_timeout: 5m                  # 用于设置处理超时时间，默认是5分钟
4 route:                                # 路由配置模块
5   group_by: ['alertname']              # 告警分组
6   group_wait: 10s                      # 10s内收到的同组告警在同一条告警通知中发送出去
7   group_interval: 10s                  # 同组之间发送告警通知的时间间隔
8   repeat_interval: 1h                  # 相同告警信息发送重复告警的周期
9   receiver: 'web.hook'                 # 使用的接收器名称
10 receivers:                             # 接收器
11 - name: 'web.hook'                     # 接收器名称
12   webhook_configs:                     # 设置webhook地址
13 - url: 'http://127.0.0.1:5001/'
14 inhibit_rules:                          # 告警抑制功能模块
15 - source_match:
16     severity: 'critical'                # 当存在源标签告警触发时抑制含有目标标签的告警
17   target_match:
18     severity: 'warning'
19   equal: ['alertname', 'dev', 'instance'] # 保证该配置下标签内容相同才会被抑制
抑制
```

- 自定义alertmanager使用qq邮箱报警的配置文件

```
1 [root@server1 alertmanager]# cat alertmanager.yml
2 global:
3   resolve_timeout: 5m
4   smtp_from: 'bbj1030@foxmail.com'
5   smtp_smarthost: 'smtp.qq.com:465'
6   smtp_auth_username: 'bbj1030@foxmail.com'
7   smtp_auth_password: '*****'
8   smtp_require_tls: false
9   smtp_hello: 'qq.com'
10 route:
11   group_by: ['alertname']
12   group_wait: 5s
13   group_interval: 5s
14   repeat_interval: 5m
15   receiver: 'email'
16 receivers:
17 - name: 'email'
18   email_configs:
19 - to: '2177780569@qq.com'
20   send_resolved: true
21 inhibit_rules:
22 - source_match:
23     severity: 'critical'
24   target_match:
25     severity: 'warning'
26   equal: ['alertname', 'dev', 'instance']
```

- 启动Alertmanager

```
1 [root@server1 alertmanager]# ./alertmanager
```

- 组合Prometheus与alertmanager

```
1 [root@server1 prometheus]# cat prometheus.yml
2 alerting:
3   alertmanagers:
4     - static_configs:
5       - targets:
6         - 192.168.80.152:9093
7     .....
8 scrape_configs:
9   - job_name: 'Alertmanager'
10     static_configs:
11       - targets: ['192.168.80.152:9093']
```

- 告警规则

```
1 [root@server1 prometheus]# cat prometheus.yml
2 rule_files:
3   - "/data/prometheus/rules/*.yaml"
4
5 [root@server1 rules]# pwd
6 /data/prometheus/rules
7 [root@server1 rules]# cat rules.yml
8 groups:
9   - name: up
10     rules:
11       - alert: node
12         expr: up{job="node_exporter"} == 0
13         for: 1m
14         labels:
15           severity: critical
16         annotations:
17           description: "Node has been down for more than 1 minutes"
18           summary: "Node down"
19
20 [root@server1 prometheus]# ./promtool check rules rules/rules.yml
21 Checking rules/rules.yml
22 SUCCESS: 1 rules found
23
```

Prometheus Alerts Graph Status Help Classic UI			
Rules			
up	1h 29m 14s ago		0.947ms
Rule	State	Error	Evaluation Time
alert: node expr: up{job="node_exporter"} == 0 for: 3m labels: severity: critical annotations: description: Node has been down for more than 5 minutes summary: Node down	OK		0.928ms

Prometheus Alerts Graph Status Help Classic UI			
<div> Inactive (0) Pending (1) Firing (0) Show annota </div>			
/data/prometheus/rules/rules.yml > up pending (1)			
▼ node (1 active)			
name: node expr: up{job="node_exporter"} == 0 for: 3m labels: severity: critical annotations: description: Node has been down for more than 5 minutes summary: Node down			
Labels	State	Active Since	Value
alertname=node instance=192.168.80.152:9100 job=node_exporter severity=critical	PENDING	2021-03-10T05:34:09.129609539Z	0

- Interval没有满足触发条件，告警未激活状态
- pending，已满足触发条件，但未满足告警持续时间的状态，即未满足告警中for子句指定的持续时间
- firing，已满足触发条件且已经超过for子句中指定的持续时间时的状态

QQ邮箱

mail.qq.com

Redhat<2177780569@qq.com>

邮箱首页 | 设置 · 换肤

写信

收信

通讯录

收件箱 (13)

星标邮件

群邮件

草稿箱

已发送

已删除

垃圾箱

其他邮箱

日历 | 记事本

简历

在线文档

附件收藏

文件中转站

贺卡 | 明信片

返回

回复

回复全部

转发

删除

彻底删除

举报

拒收

标记为...

移动到...

[FIRING:1] node (192.168.80.20:9100 node_exporter 1)

发件人: bbj1030 <bbj1030@foxmail.com>

时间: 2021年8月1日 (星期日) 上午11:13

收件人: Redhat <2177780569@qq.com>

邮件可翻译为中文

立即翻译

1 alert for alertname=node

View In Alertmanager

[1] Firing

Labels

alertname = node
 instance = 192.168.80.20:9100
 job = node_exporter
 severity = 1

Annotations

description = Node has been down for more than 1 minutes
 summary = Node down

Source

自行扩展，alertmanager定义receiver，使用其他方式告警，自行研究可视化工具grafana的使用

