尚硅谷大数据项目之实时项目

Prometheus&Grafana监控

(作者：尚硅谷大数据研发部)

版本：V2.0

# Prometheus入门

Prometheus受启发于Google的Brogmon监控系统（相似的Kubernetes是从Google的Brog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。

2016年5月继Kubernetes之后成为第二个正式加入CNCF基金会的项目，同年6月正式发布1.0版本。2017年底发布了基于全新存储层的2.0版本，能更好地与容器平台、云平台配合。

Prometheus作为新一代的云原生监控系统，目前已经有超过650+位贡献者参与到Prometheus的研发工作上，并且超过120+项的第三方集成。

## Prometheus的特点

Prometheus是一个开源的完整监控解决方案，其对传统监控系统的测试和告警模型进行了彻底的颠覆，形成了基于中央化的规则计算、统一分析和告警的新模型。 相比于传统监控系统，Prometheus具有以下优点：

### 易于管理

* 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。
* Prometheus基于Pull模型的架构方式，可以在任何地方（本地电脑，开发环境，测试环境）搭建我们的监控系统。
* 对于一些复杂的情况，还可以使用Prometheus服务发现(Service Discovery)的能力动态管理监控目标。

### 监控服务的内部运行状态

Pometheus鼓励用户监控服务的内部状态，基于Prometheus丰富的Client库，用户可以轻松的在应用程序中添加对Prometheus的支持，从而让用户可以获取服务和应用内部真正的运行状态。

### 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。

* http\_request\_status：指标名称(Metrics Name)
* {code='200',content\_path='/api/path',environment='produment'}：表示维度的标签，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。
* [value1@timestamp1,value2@timestamp2...]：按照时间的先后顺序 存储的样本值。

### 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。

通过PromQL可以轻松回答类似于以下问题：

* 在过去一段时间中95%应用延迟时间的分布范围？
* 预测在4小时后，磁盘空间占用大致会是什么情况？
* CPU占用率前5位的服务有哪些？(过滤)

### 高效

对于监控系统而言，大量的监控任务必然导致有大量的数据产生。而Prometheus可以高效地处理这些数据，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

### 可扩展

可以在每个数据中心、每个团队运行独立的Prometheus Sevrer。Prometheus对于联邦集群的支持，可以让多个Prometheus实例产生一个逻辑集群，当单实例Prometheus Server处理的任务量过大时，通过使用功能分区(sharding)+联邦集群(federation)可以对其进行扩展。

### 易于集成

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。

同时这些客户端收集的监控数据，不仅仅支持 Prometheus，还能支持Graphite这些其他的监控工具。

同时Prometheus还支持与其他的监控系统进行集成：Graphite， Statsd， Collected， Scollector， muini， Nagios等。 Prometheus社区还提供了大量第三方实现的监控数据采集支持：JMX，CloudWatch，EC2，MySQL，PostgresSQL，Haskell，Bash，SNMP，Consul，Haproxy，Mesos，Bind，CouchDB，Django，Memcached，RabbitMQ，Redis，RethinkDB，Rsyslog等等。

### 可视化

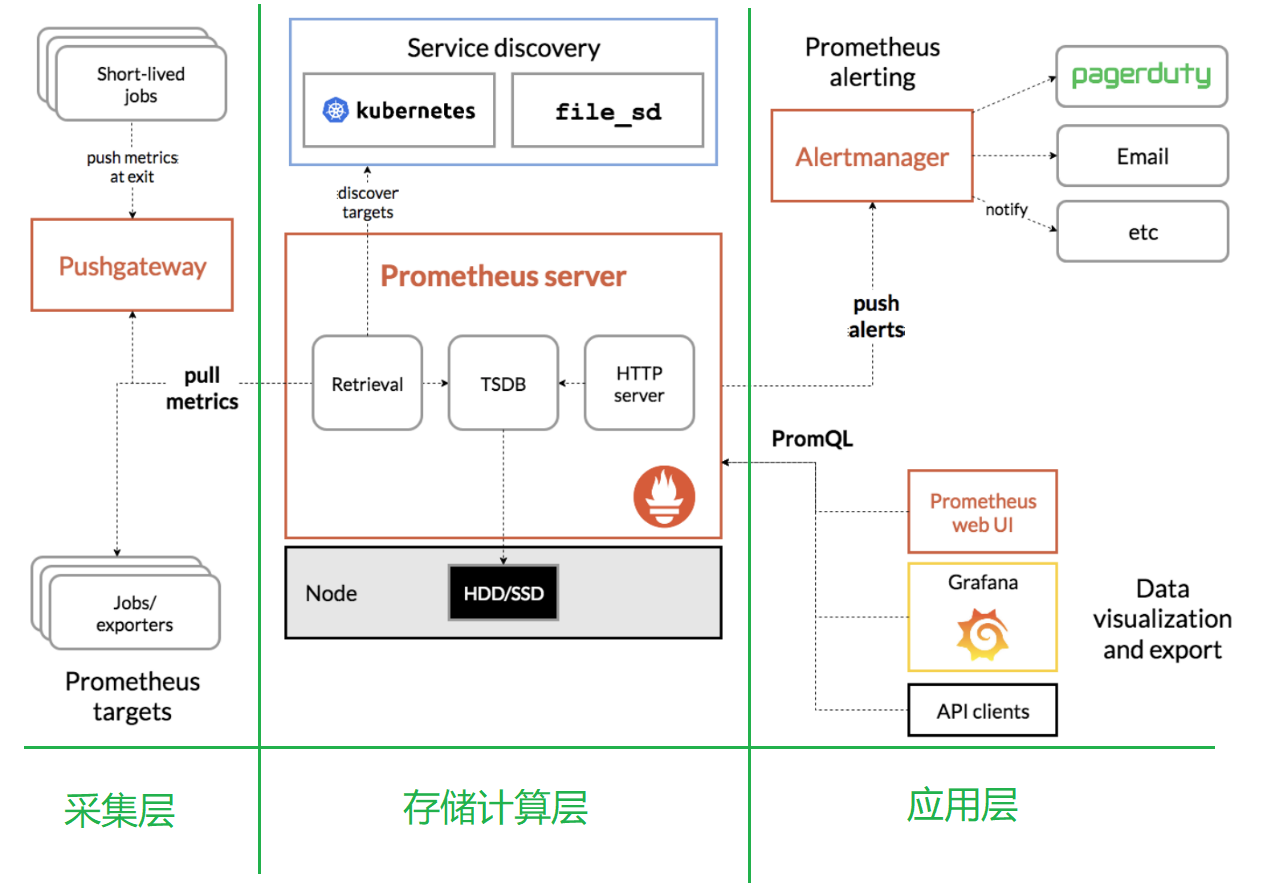
* Prometheus Server中自带的Prometheus UI，可以方便地直接对数据进行查询，并且支持直接以图形化的形式展示数据。同时Prometheus还提供了一个独立的基于Ruby On Rails的Dashboard解决方案 Promdash。
* 最新的Grafana可视化工具也已经提供了完整的Prometheus支持，基于Grafana可以创建更加精美的监控图标。
* 基于Prometheus提供的API还可以实现自己的监控可视化UI。

### 开放性

通常来说当我们需要监控一个应用程序时，一般需要该应用程序提供对相应监控系统协议的支持，因此应用程序会与所选择的监控系统进行绑定。为了减少这种绑定所带来的限制，对于决策者而言要么你就直接在应用中集成该监控系统的支持，要么就在外部创建单独的服务来适配不同的监控系统。

而对于Prometheus来说，使用Prometheus的client library的输出格式不止支持Prometheus的格式化数据，也可以输出支持其它监控系统的格式化数据，比如Graphite。 因此你甚至可以在不使用Prometheus的情况下，采用Prometheus的client library来让你的应用程序支持监控数据采集。

## Prometheus的架构



### Prometheus生态圈组件

* Prometheus Server：主服务器，负责收集和存储时间序列数据
* client libraies：应用程序代码插桩，将监控指标嵌入到被监控应用程序中
* Pushgateway：推送网关，为支持short-lived作业提供一个推送网关
* exporter：专门为一些应用开发的数据摄取组件—exporter，例如：HAProxy、StatsD、Graphite等等。
* Alertmanager：专门用于处理alert的组件

### 架构理解

Prometheus既然设计为一个维度存储模型，可以把它理解为一个OLAP系统。

**1、存储计算层**

* Prometheus Server，里面包含了存储引擎和计算引擎。
* Retrieval组件为取数组件，它会主动从Pushgateway或者Exporter拉取指标数据。
* Service discovery，可以动态发现要监控的目标。
* TSDB，数据核心存储与查询。
* HTTP server，对外提供HTTP服务。

**2、采集层**

采集层分为两类，一类是生命周期较短的作业，还有一类是生命周期较长的作业。

* 短作业：直接通过API，在退出时间指标推送给Pushgateway。
* 长作业：Retrieval组件直接从Job或者Exporter拉取数据。

**3、应用层**

应用层主要分为两种，一种是AlertManager，另一种是数据可视化。

* AlertManager

对接Pagerduty，是一套付费的监控报警系统。可实现短信报警、5分钟无人ack打电话通知、仍然无人ack，通知值班人员Manager...

Emial，发送邮件

... ...

* 数据可视化

Prometheus build-in WebUI

Grafana

其他基于API开发的客户端

# Prometheus的安装

官网：<https://prometheus.io/>

下载地址：<https://prometheus.io/download/>

## 安装Prometheus Server

Prometheus基于Golang编写，编译后的软件包，不依赖于任何的第三方依赖。只需要下载对应平台的二进制包，解压并且添加基本的配置即可正常启动Prometheus Server。

### 上传安装包

上传prometheus-2.26.0.linux-amd64.tar.gz到虚拟机的/opt/software目录

### 解压安装包

* 解压到/opt/module目录下

[atguigu@hadoop1 software]$ tar -zxvf prometheus-2.26.0.linux-amd64.tar.gz -C /opt/module

* 修改目录名

[atguigu@hadoop1 ~] cd /opt/module

[atguigu@hadoop1 module] mv prometheus-2.26.0.linux-amd64 prometheus-2.26.0

### 修改配置文件 prometheus.yml

[atguigu@hadoop1 prometheus-2.26.0]$ vim prometheus.yml

在scrape\_configs配置项下添加配置：

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['hadoop1:9090']

# 添加 PushGateway 监控配置

- job\_name: 'pushgateway'

static\_configs:

- targets: ['hadoop1:9091']

labels:

instance: pushgateway

# 添加 Node Exporter 监控配置

- job\_name: 'node exporter'

static\_configs:

- targets: ['hadoop1:9100', 'hadoop2:9100', 'hadoop3:9100']

labels:

instance: node exporter

配置说明：

**1、global配置块**：控制Prometheus服务器的全局配置

* scrape\_interval：配置拉取数据的时间间隔，默认为1分钟。
* evaluation\_interval：规则验证（生成alert）的时间间隔，默认为1分钟。

**2、rule\_files配置块**：规则配置文件

**3、scrape\_configs配置块**：配置采集目标相关， prometheus监视的目标。Prometheus自身的运行信息可以通过HTTP访问，所以Prometheus可以监控自己的运行数据。

* job\_name：监控作业的名称
* static\_configs：表示静态目标配置，就是固定从某个target拉取数据
* targets：指定监控的目标，其实就是从哪儿拉取数据。Prometheus会从http://hadoop1:9090/metrics上拉取数据。

*Prometheus是可以在运行时自动加载配置的。启动时需要添加：--web.enable-lifecycle*

## 安装Pushgateway

Prometheus在正常情况下是采用拉模式从产生metric的作业或者exporter（比如专门监控主机的NodeExporter）拉取监控数据。但是我们要监控的是Flink on YARN作业，想要让Prometheus自动发现作业的提交、结束以及自动拉取数据显然是比较困难的。PushGateway就是一个中转组件，通过配置Flink on YARN作业将metric推到PushGateway，Prometheus再从PushGateway拉取就可以了。

### 上传安装包

上传pushgateway-1.4.0.linux-amd64.tar.gz到虚拟机的/opt/software目录

### 解压安装包

* 解压到/opt/module目录下

[atguigu@hadoop1 software]$ tar -zxvf pushgateway-1.4.0.linux-amd64.tar.gz -C /opt/module

* 修改目录名

[atguigu@hadoop1 ~] cd /opt/module

[atguigu@hadoop1 module] mv pushgateway-1.4.0.linux-amd64 pushgateway-1.4.0

## 安装Node Exporter（选择性安装）

在Prometheus的架构设计中，Prometheus Server并不直接服务监控特定的目标，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。因此为了能够能够监控到某些东西，如主机的CPU使用率，我们需要使用到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。

Exporter可以是一个相对开放的概念，其可以是一个独立运行的程序独立于监控目标以外，也可以是直接内置在监控目标中。只要能够向Prometheus提供标准格式的监控样本数据即可。

为了能够采集到主机的运行指标如CPU, 内存，磁盘等信息。我们可以使用Node Exporter。Node Exporter同样采用Golang编写，并且不存在任何的第三方依赖，只需要下载，解压即可运行。可以从<https://prometheus.io/download/> 获取最新的node exporter版本的二进制包。

### 上传安装包

上传node\_exporter-1.1.2.linux-amd64.tar.gz到虚拟机的/opt/software目录

### 解压安装包

* 解压到/opt/module目录下

[atguigu@hadoop1 software]$ tar -zxvf node\_exporter-1.1.2.linux-amd64.tar.gz -C /opt/module

* 修改目录名

[atguigu@hadoop1 ~] cd /opt/module

[atguigu@hadoop1 module] mv node\_exporter-1.1.2.linux-amd64 node\_exporter-1.1.2

* 启动并页面查看是否成功

执行./node\_exporter

浏览器输入：<http://hadoop1:9100/metrics>，可以看到当前node exporter获取到的当前主机的所有监控数据。

### 节点分发

* 将解压后的目录分发到要监控的节点

[atguigu@hadoop1 module] xsync node\_exporter-1.1.2

* 修改Prometheus配置文件prometheus.yml，在2.1.3的时候已经添加过配置

### 设置为开机自启

* 创建service文件

[atguigu@hadoop1 module] sudo vim /usr/lib/systemd/system/node\_exporter.service

[Unit]

Description=node\_export

Documentation=https://github.com/prometheus/node\_exporter

After=network.target

[Service]

Type=simple

User=atguigu

ExecStart= /opt/module/node\_exporter-1.1.2/node\_exporter

Restart=on-failure

[Install]

WantedBy=multi-user.target

* 分发文件

[atguigu@hadoop1 module] sudo /home/atguigu/bin/xsync /usr/lib/systemd/system/node\_exporter.service

* 设为开机自启动（所有机器都执行）

[atguigu@hadoop1 module] sudo systemctl enable node\_exporter.service

* 启动服务（所有机器都执行）

[atguigu@hadoop1 module] sudo systemctl start node\_exporter.service

## 启动Prometheus Server 和 Pushgateway

### Prometheus Server目录下执行启动命令

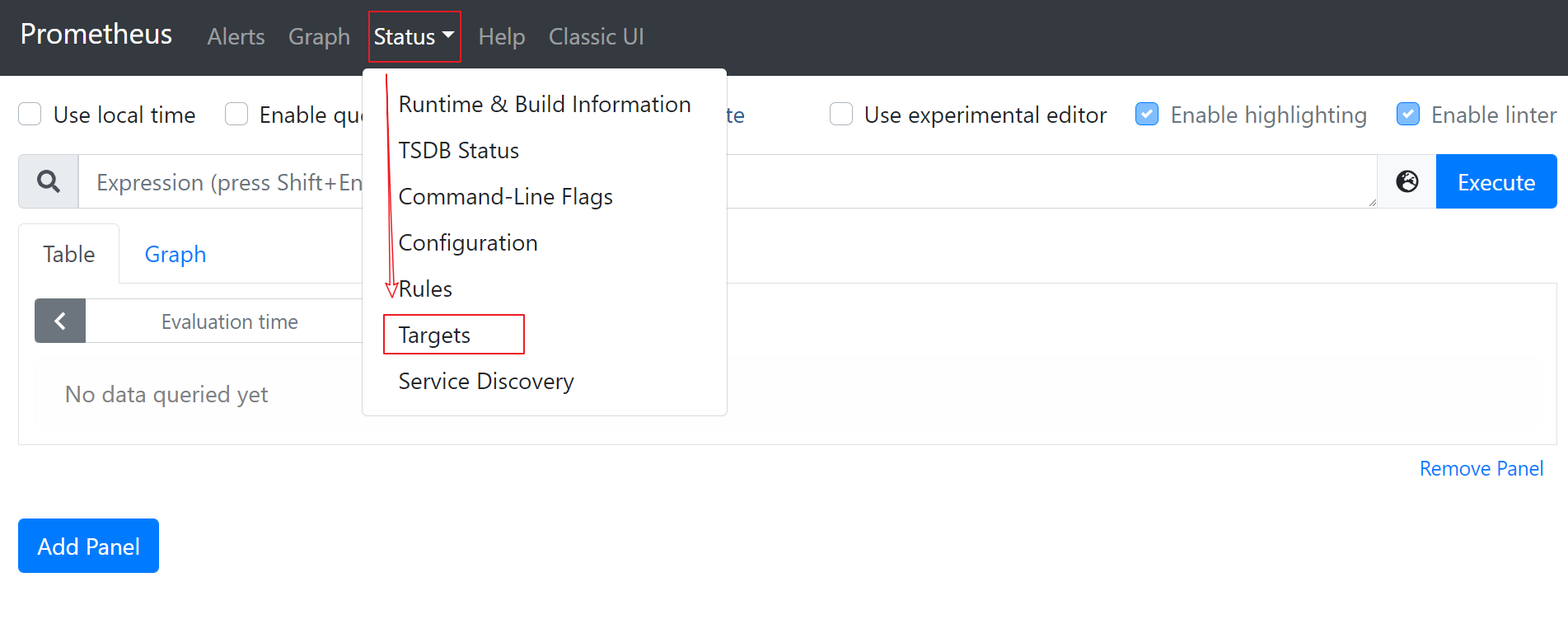
[atguigu@hadoop1 prometheus-2.26.0]$ nohup ./prometheus --config.file=prometheus.yml > ./prometheus.log 2>&1 &

### Pushgateway目录下执行启动命令

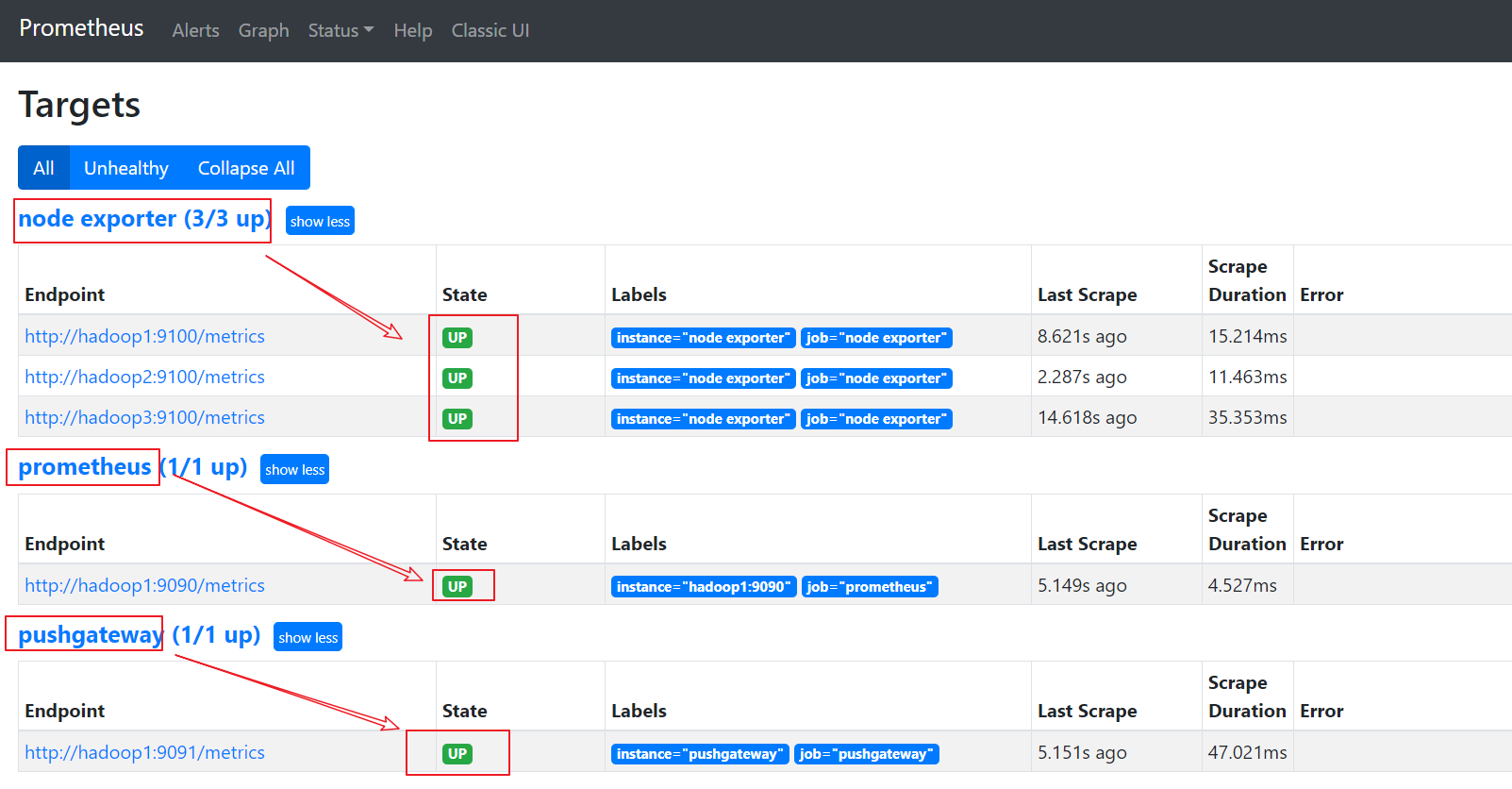
[atguigu@hadoop1 pushgateway-1.4.0]$ nohup ./pushgateway --web.listen-address :9091 > ./pushgateway.log 2>&1 &

### 打开web页面查看

* 浏览器输入：<http://hadoop1:9090/>
* 点击 Status，选中Targets：



* prometheus、pushgateway和node exporter都是up状态，表示安装启动成功：



# PromQL介绍

Prometheus通过指标名称（metrics name）以及对应的一组标签（labelset）唯一定义一条时间序列。指标名称反映了监控样本的基本标识，而label则在这个基本特征上为采集到的数据提供了多种特征维度。用户可以基于这些特征维度过滤，聚合，统计从而产生新的计算后的一条时间序列。PromQL是Prometheus内置的数据查询语言，其提供对时间序列数据丰富的查询，聚合以及逻辑运算能力的支持。并且被广泛应用在Prometheus的日常应用当中，包括对数据查询、可视化、告警处理当中。可以这么说，PromQL是Prometheus所有应用场景的基础，理解和掌握PromQL是Prometheus入门的第一课。

## 基本用法

### 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：

prometheus\_http\_requests\_total{code="200",handler="alerts",instance="localhost:9090",job="prometheus",method="get"}= (20889@1518096812.326)

prometheus\_http\_requests\_total{code="200",handler="graph",instance="localhost:9090",job="prometheus",method="get"}= ([21287@1518096812.326](mailto:21287@1518096812.326))

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；

例如，如果我们只需要查询所有prometheus\_http\_requests\_total时间序列中满足标签instance为localhost:9090的时间 序列，则可以使用如下表达式：

prometheus\_http\_requests\_total{instance="localhost:9090"}

反之使用 instance!="localhost:9090" 则可以排除这些时间序列：

prometheus\_http\_requests\_total{instance!="localhost:9090"}

* **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=~regx 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!~regx 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"}

### 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\_request\_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：

prometheus\_http\_requests\_total{code="200",handler="alerts",instance="localhost:9090",job="prometheus",method="get"}=[

1@1518096812.326

1@1518096817.326

[1@1518096822.326](mailto:1@1518096822.326)

[1@1518096827.326](mailto:1@1518096827.326)

[1@1518096832.326](mailto:1@1518096832.326)

[1@1518096837.325](mailto:1@1518096837.325)

] 9. prometheus\_http\_requests\_total{code="200",handler="graph",instance="localhost:9090",job="prometheus",method="get"}=[

4@1518096812.326

[4@1518096817.326](mailto:4@1518096817.326)

[4@1518096822.326](mailto:4@1518096822.326)

[4@1518096827.326](mailto:4@1518096827.326)

[4@1518096832.326](mailto:4@1518096832.326)

4@1518096837.325

]

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

### 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：

http\_request\_total{} # 瞬时向量表达式，选择当前最新的数据

http\_request\_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\_request\_total{} **offset** 5m

prometheus\_http\_request\_total{}[1d] **offset** 1d

### 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\_request\_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)

### 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`

### 合法的PromQL表达式

所有的PromQL表达式都**必须至少包含一个指标名称**(例如http\_request\_total)，或者一个**不会匹配到空字符串**的标签过滤器(例如{code=”200”})。

因此以下两种方式，均为合法的表达式：

prometheus\_http\_request\_total # 合法

prometheus\_http\_request\_total{} # 合法

{method="get"} # 合法

而如下表达式，则不合法：

{job=~".\*"} # 不合法

同时，除了使用 {label=value} 的形式以外，我们还可以使用内置的 \_\_name\_\_ 标签来指定监控指标名称：

{\_\_name\_\_=~"prometheus\_http\_request\_total"} # 合法

{\_\_name\_\_=~"node\_disk\_bytes\_read|node\_disk\_bytes\_written"} # 合法

## 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。

### 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

* **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)
* **使用bool修饰符改变布尔运算符的行为**

布尔运算符的默认行为是对时序数据进行**过滤**。而在其它的情况下我们可能需要的是真正的布尔结果。例如，只需要 知道当前模块的HTTP请求量是否>=1000，如果大于等于1000则返回1（true）否则返回0（false）。这时可以使 用bool修饰符改变布尔运算的默认行为。 例如：

prometheus\_http\_requests\_total > bool 1000

使用bool修改符后，布尔运算不会对时间序列进行过滤，而是直接依次瞬时向量中的各个样本数据与标量的比较结果 0或者1。从而形成一条新的时间序列。

prometheus\_http\_requests\_total{code="200",handler="query",instance="localhost:9090",job="prometheus",method="get"} 1

prometheus\_http\_requests\_total{code="200",handler="query\_range",instance="localhost:9090",job="prometheus",method="get"} 0

同时需要注意的是，如果是在两个标量之间使用布尔运算，则必须使用bool修饰符

2 == bool 2 # 结果为1

### 使用集合运算符

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。

### 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

### 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。

例如：

sum(prometheus\_http\_requests\_total) without (instance)

等价于

sum(prometheus\_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：

sum(prometheus\_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：

count\_values("count", prometheus\_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：

topk(5, prometheus\_http\_requests\_total)

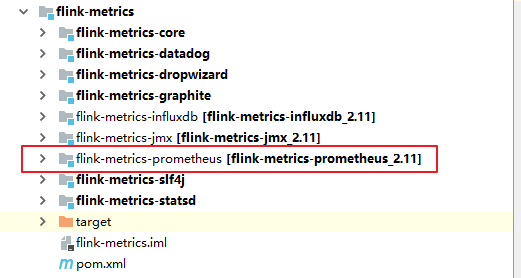
quantile用于计算当前样本数据值的分布情况quantile(φ, express)其中0 ≤ φ ≤ 1。

例如，当φ为0.5时，即表示找到当前样本数据中的中位数：

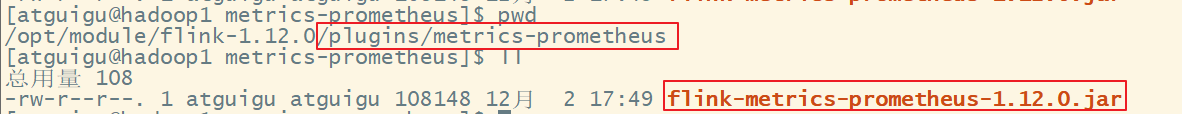
quantile(0.5, prometheus\_http\_requests\_total)

# Flink的集成

Flink 提供的 Metrics 可以在 Flink 内部收集一些指标，通过这些指标让开发人员更好地理解作业或集群的状态。由于集群运行后很难发现内部的实际状况，跑得慢或快，是否异常等，开发人员无法实时查看所有的 Task 日志。比如作业很大或者有很多作业的情况下，该如何处理？此时 Metrics 可以很好的帮助开发人员了解作业的当前状况。



从Flink的源码结构我们可以看到，Flink官方支持Prometheus，并且提供了对接Prometheus的jar包，很方便就可以集成。



## 拷贝jar包

* 将flink-metrics-prometheus-1.12.0.jar拷贝到 <flink\_home>/lib目录下

[atguigu@hadoop1 metrics-prometheus]$ cp /opt/module/flink-1.12.0/plugins/metrics-prometheus/flink-metrics-prometheus-1.12.0.jar /opt/module/flink-1.12.0/lib/

*Flink 的 Classpath 位于lib目录下，所以插件的jar包需要放到该目录下*

## 修改Flink配置

进入到Flink的conf目录，修改flink-conf.yaml

[atguigu@hadoop1 conf]$ vim flink-conf.yaml

添加如下配置：

##### 与Prometheus集成配置 #####

metrics.reporter.promgateway.class: org.apache.flink.metrics.prometheus.PrometheusPushGatewayReporter

# PushGateway的主机名与端口号

metrics.reporter.promgateway.host: hadoop1

metrics.reporter.promgateway.port: 9091

# Flink metric在前端展示的标签（前缀）与随机后缀

metrics.reporter.promgateway.jobName: flink-metrics-ppg

metrics.reporter.promgateway.randomJobNameSuffix: true

metrics.reporter.promgateway.deleteOnShutdown: false

metrics.reporter.promgateway.interval: 30 SECONDS

# Grafana安装

下载地址：<https://grafana.com/grafana/download>

## 上传并解压

* 将grafana-7.5.2.linux-amd64.tar.gz上传至/opt/software/目录下，解压：

[atguigu@hadoop1 software]$ tar -zxvf grafana-7.5.2.linux-amd64.tar.gz -C /opt/module/

* 更改名字：

[atguigu@hadoop1 module]$ mv grafana-7.5.2.linux-amd64 grafana-7.5.2

## 启动Grafana

[atguigu@hadoop1 grafana-7.5.2]$ nohup ./bin/grafana-server web > ./grafana.log 2>&1 &

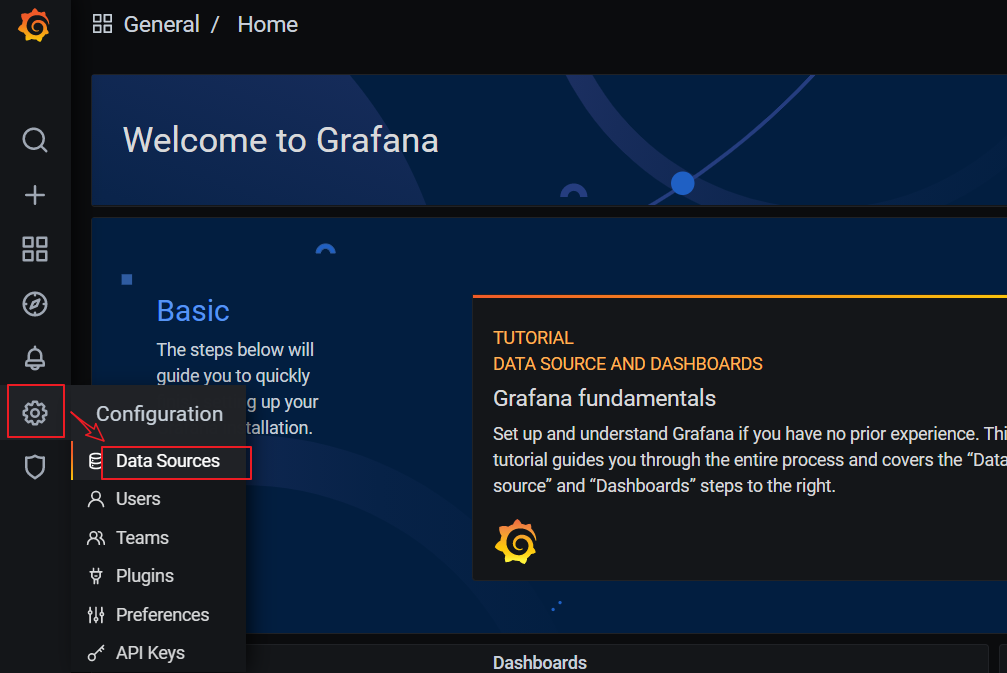
* 打开web：<http://hadoop1:3000>,默认用户名和密码：admin

# Grafana集成Prometheus

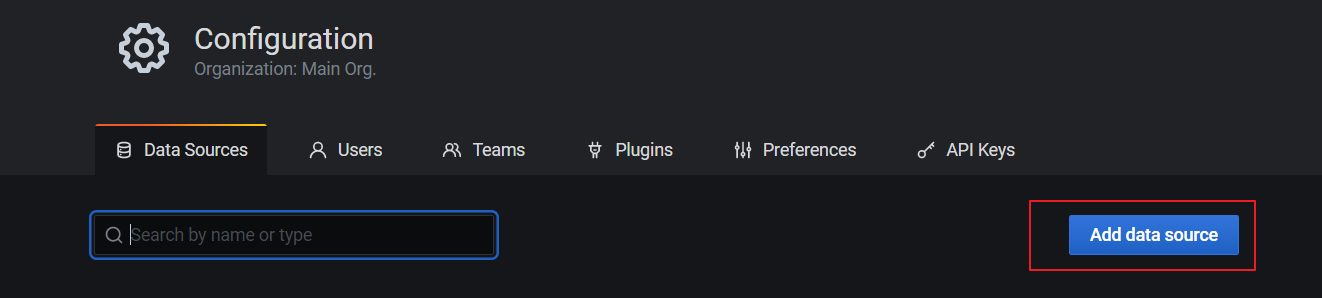
基本上来说传统关系型数据库（以MySQL为例）的SQL语句，ClickHouse基本都支持，这里不会从头讲解SQL语法只介绍ClickHouse与标准SQL（MySQL）不一致的地方。

## 添加数据源Prometheus

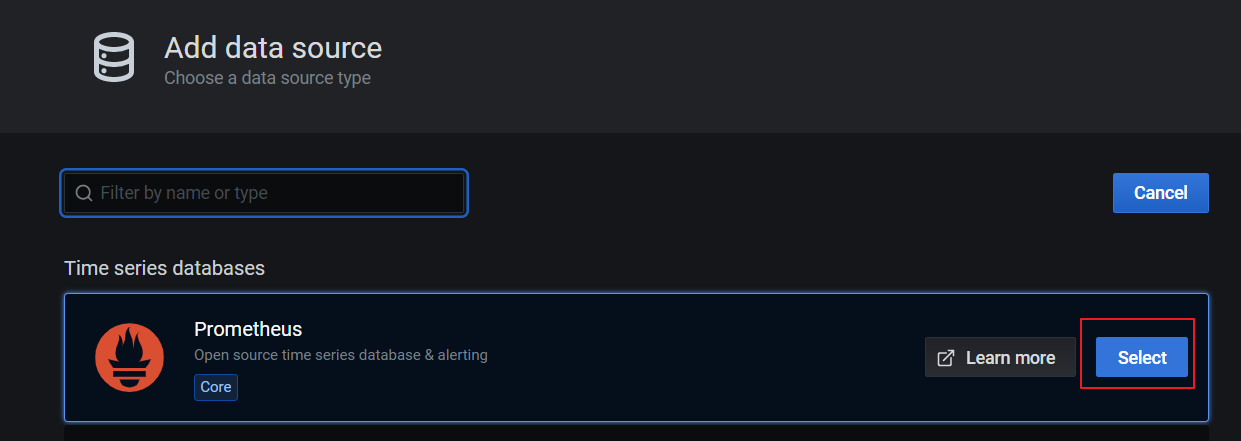
点击配置，点击Data Sources：



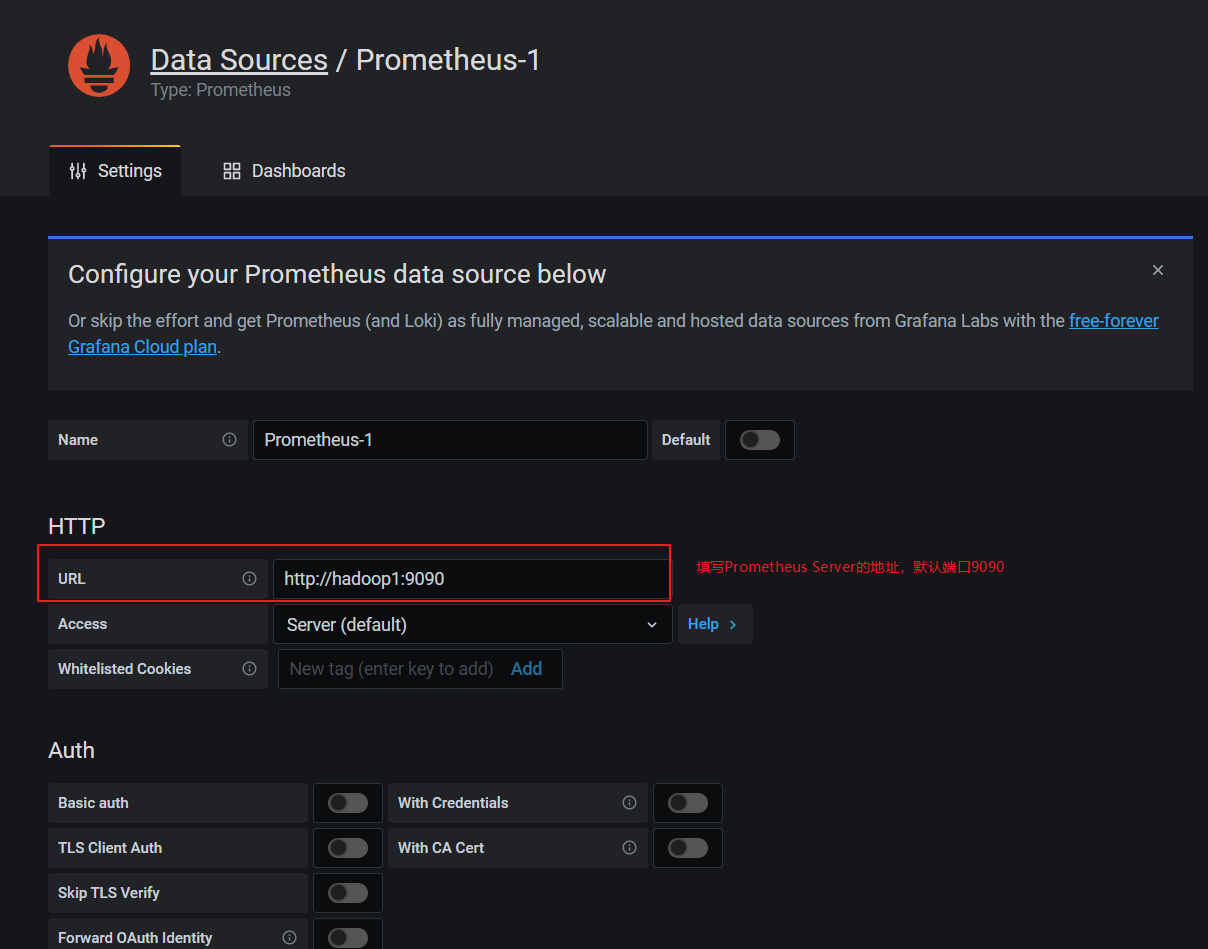
点击添加按钮：



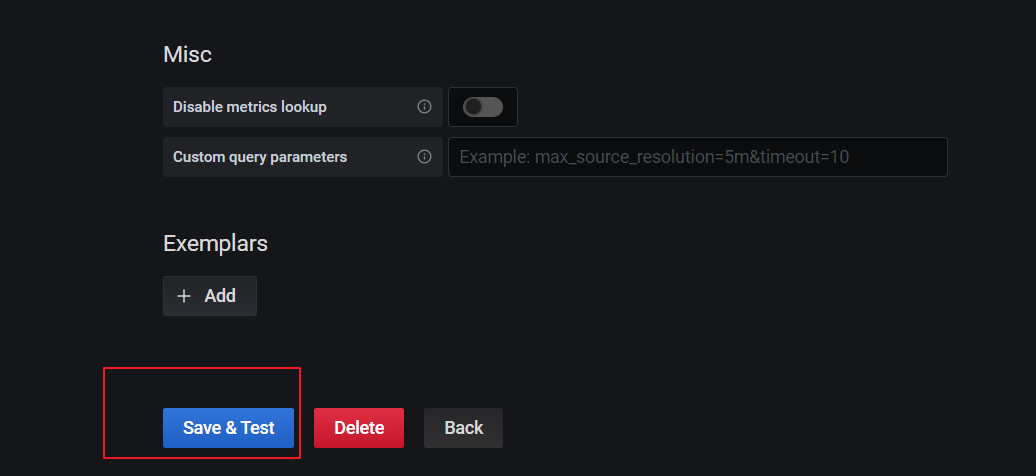
找到Prometheus，点击Select



配置Prometheus Server地址：



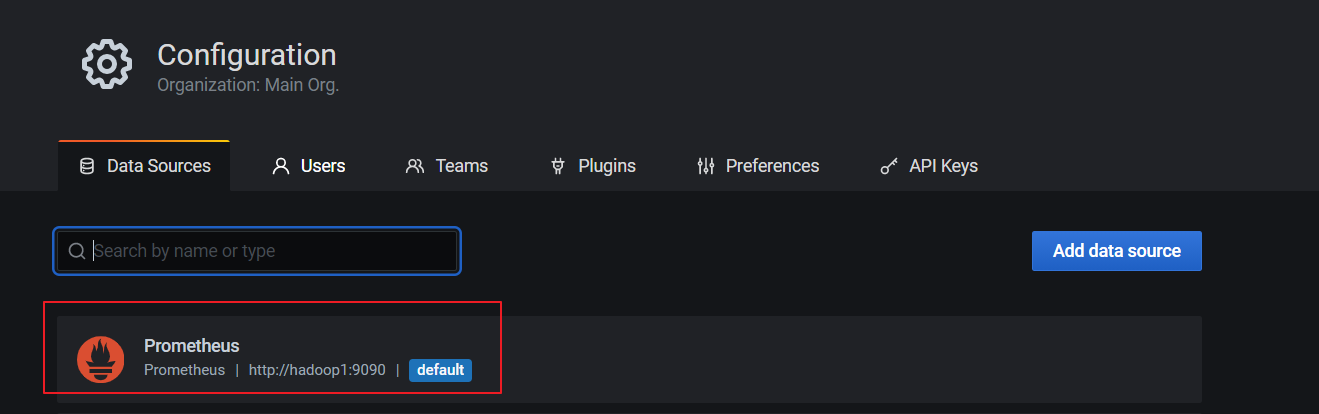
点击下方的Save&Test：



出现绿色的提示框，表示与Prometheus正常联通：

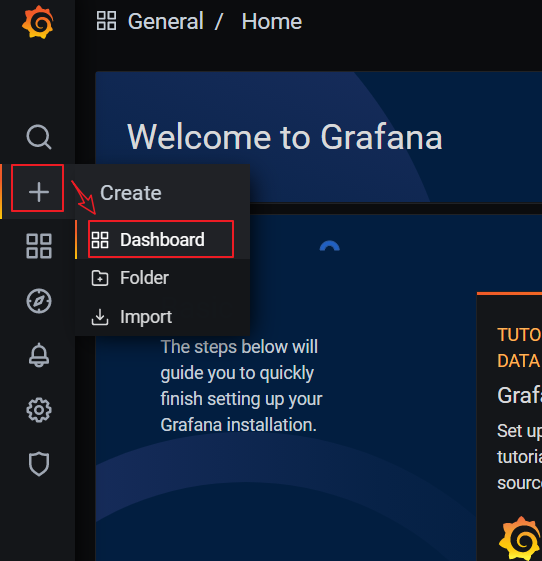


点击Back返回即可，可以看到Data Sources页面，出现了添加的Prometheus:

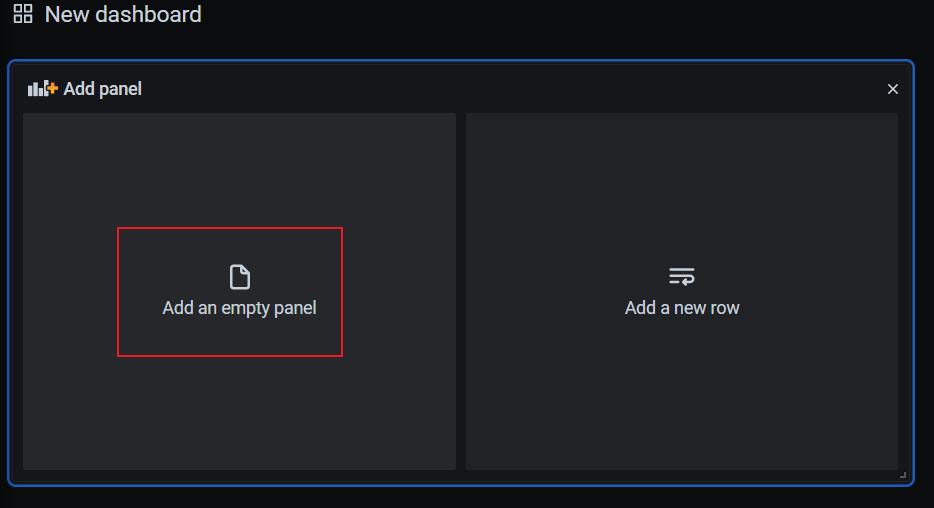


## 手动创建仪表盘Dashboard

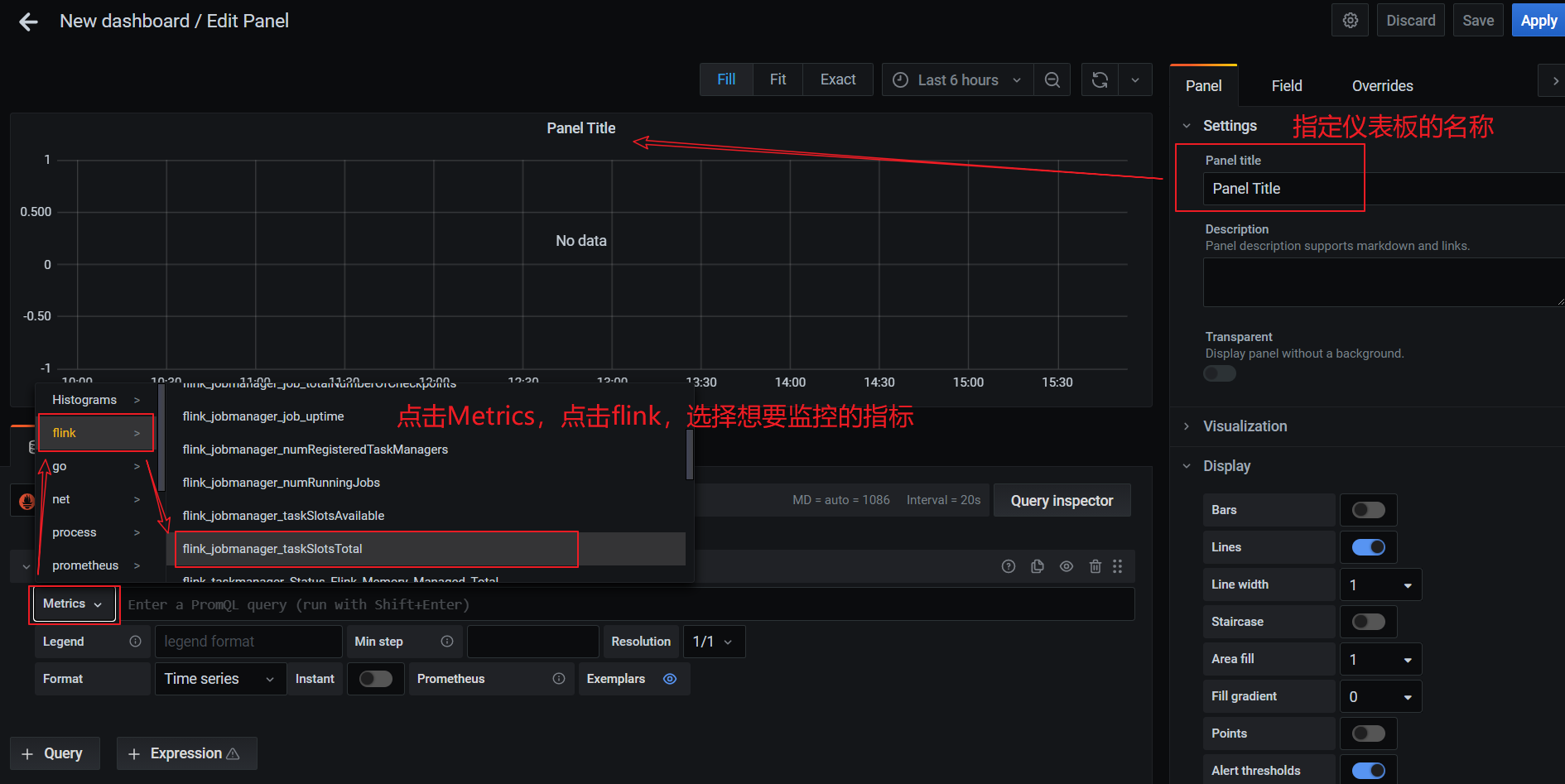
点击左边栏的 “+”号，选择Dashboard:



添加新的仪表板，点击Add an empty panel：



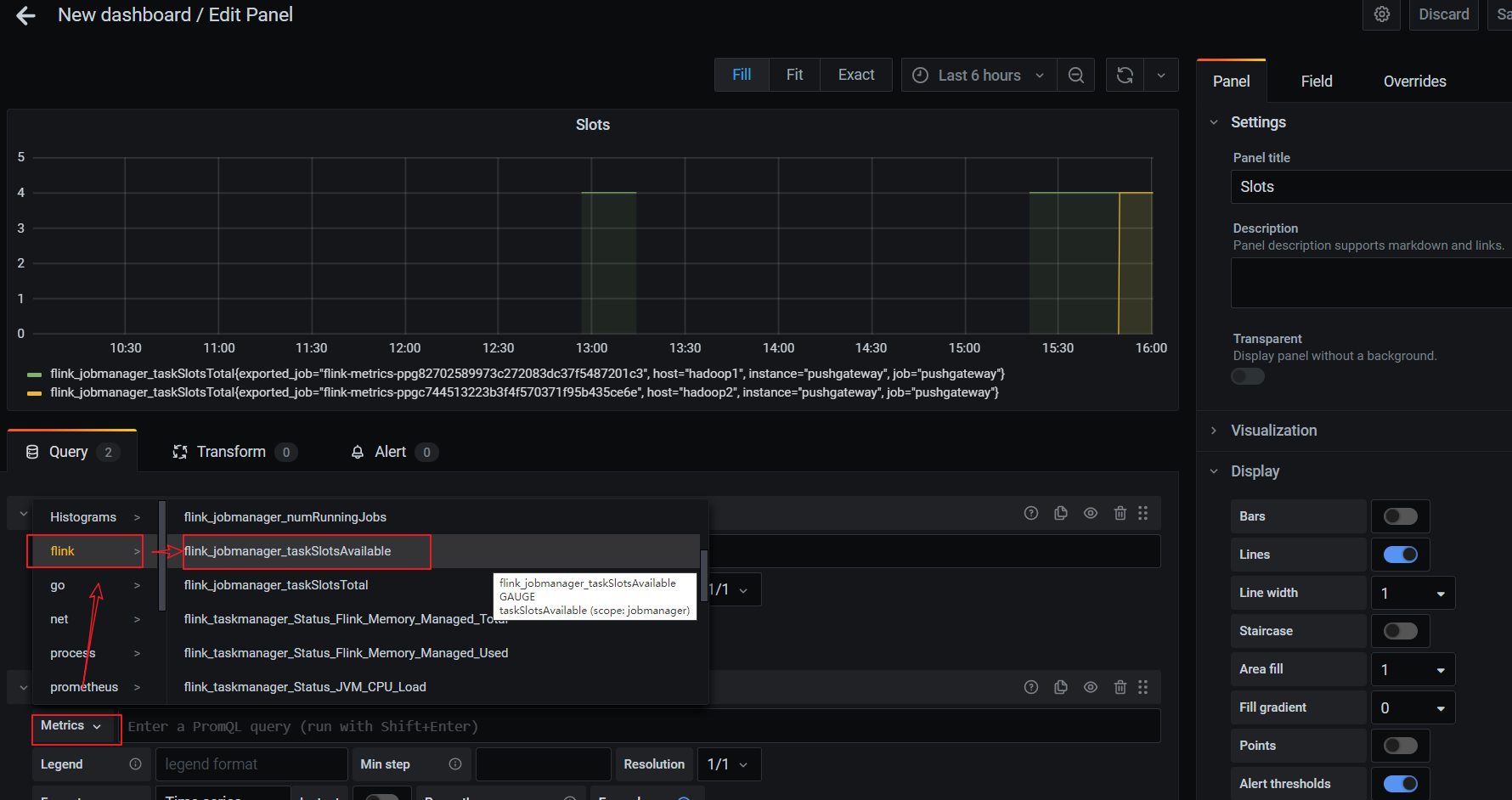
配置仪表板监控项：



一个仪表板可以配置多个监控项，添加其他监控项：



配置新的监控项：



### 任务失败监控

这一个指标监控主要是基于flink\_jobmanager\_job\_uptime 这个指标进行了监控。原理是在job任务存活时，会按照配置metrics.reporter.promgateway.interval上报频率递增。基于这个特点，当任务失败后这个数值就不会改变，就能监控到任务失败。

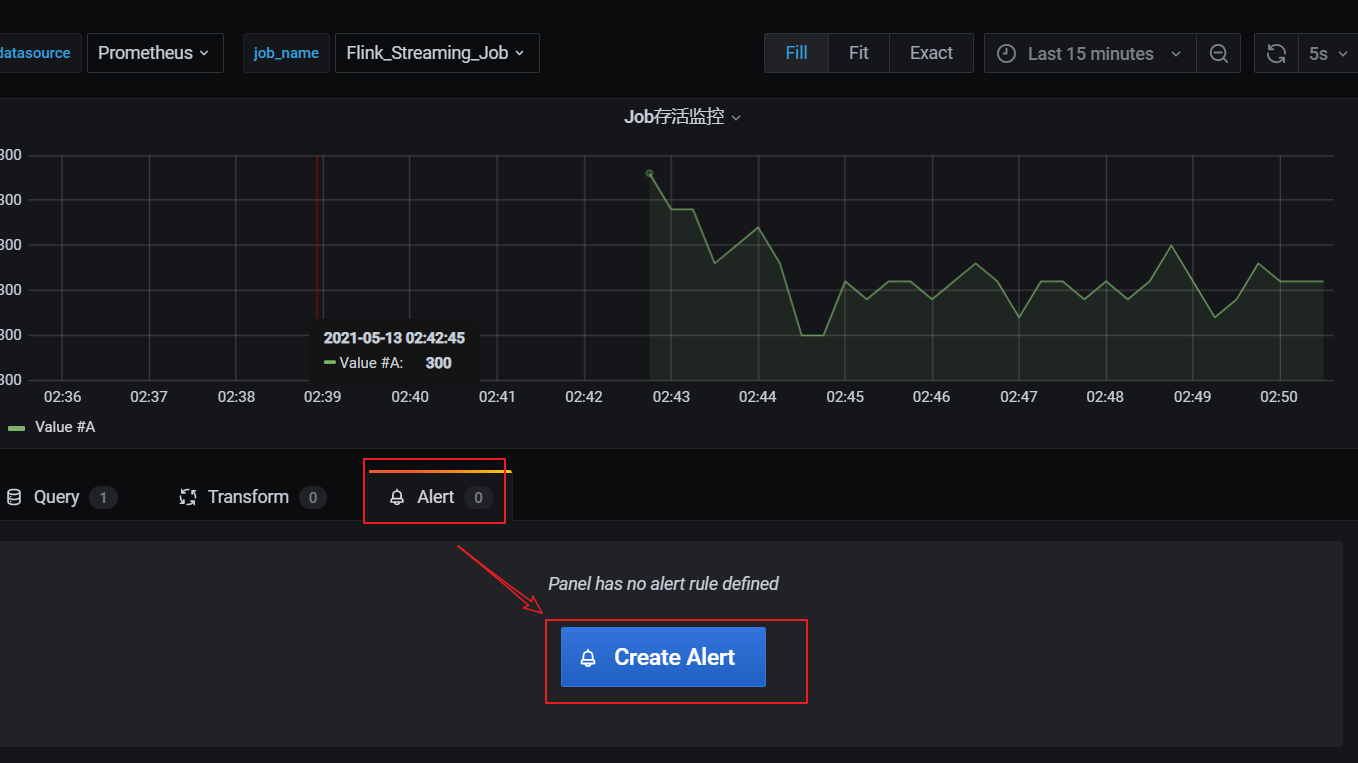
* 添加监控项：

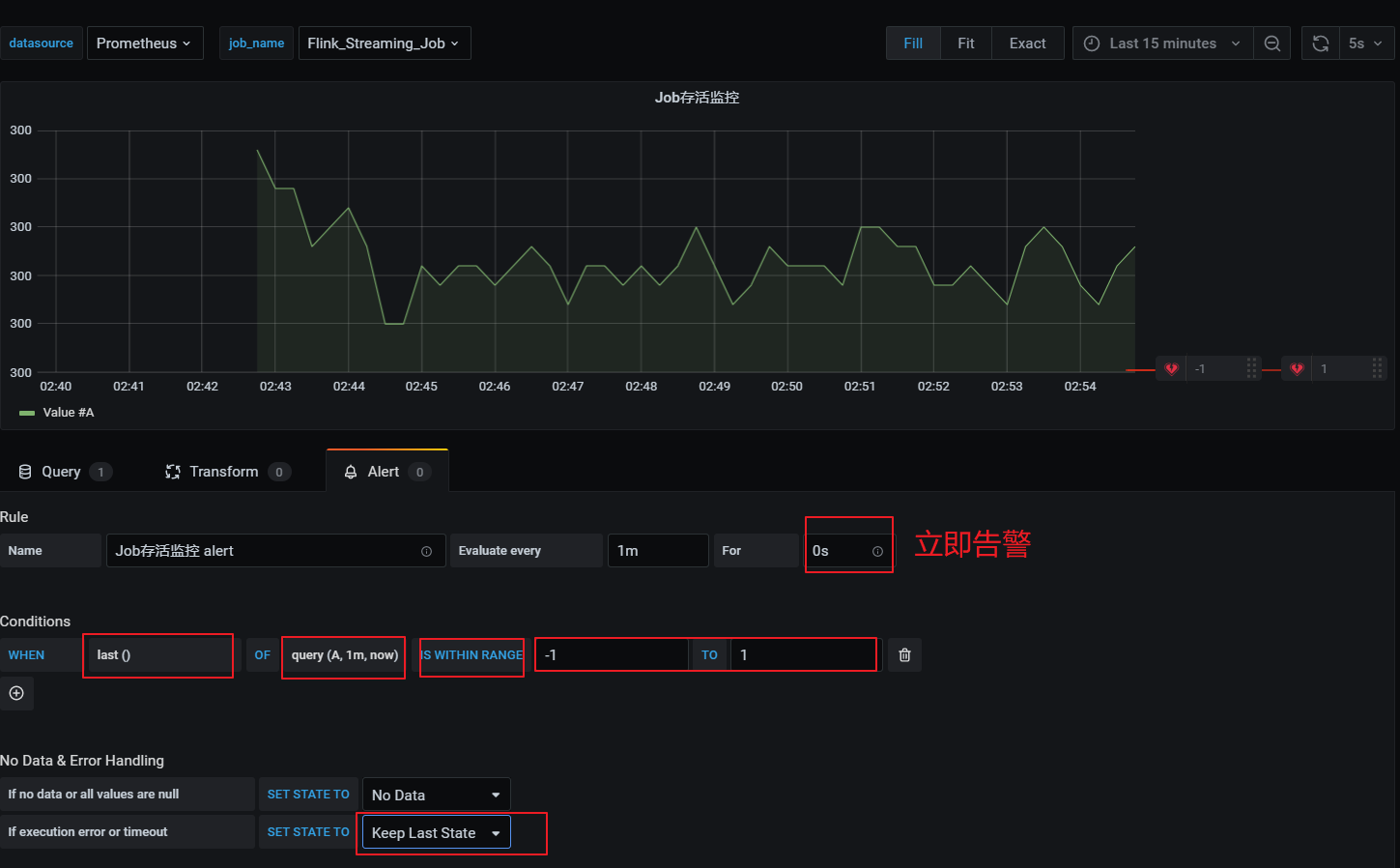


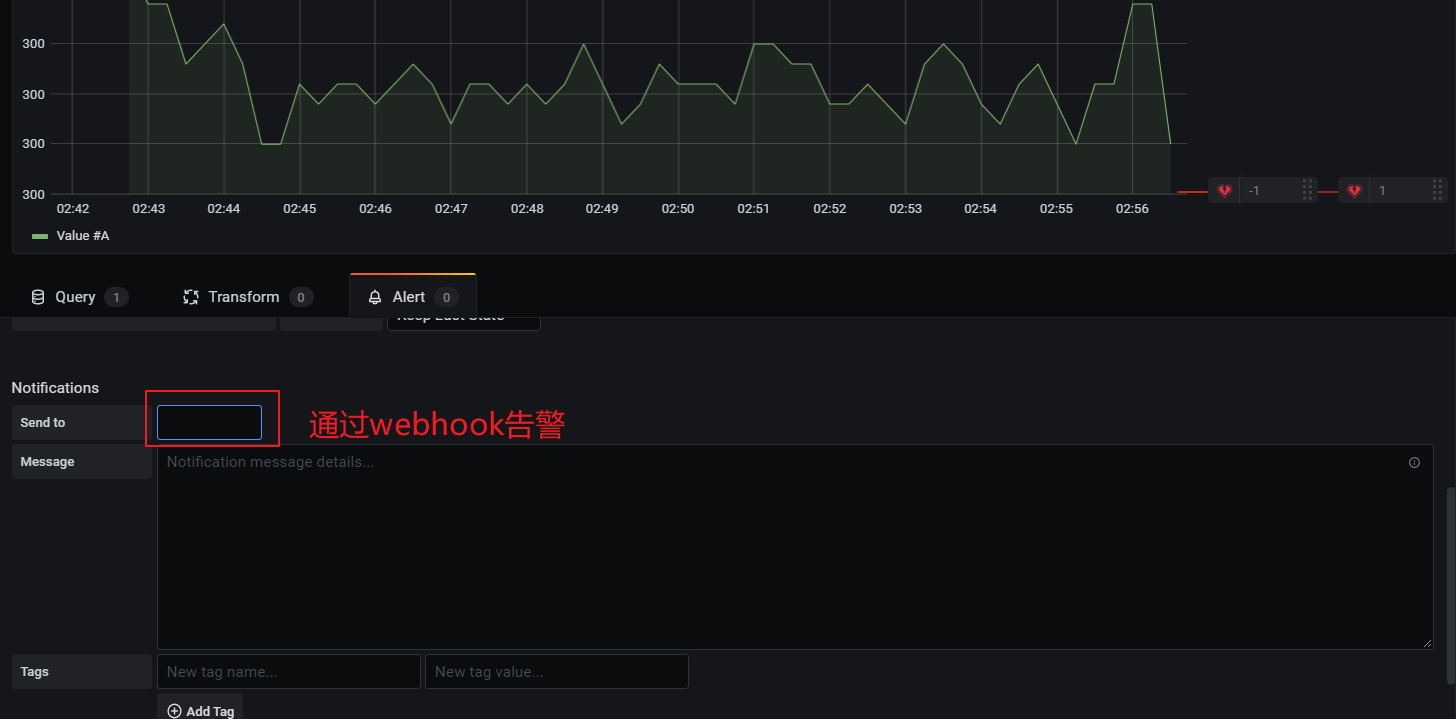


30秒为数据上报到 promgateway 频率，除以100为了数据好看，当job任务失败后数 flink上报的promgateway 的 flink\_jobmanager\_job\_uptime指标值不会变化。((flink\_jobmanager\_job\_uptime)-(flink\_jobmanager\_job\_uptime offset 30s))/100 值就会是0，可以配置告警。

* 配置告警







在告警通知中可以邮件和webhook,webhook可以调用相关接口，执行一些动作。webhook需要提前配置，在这里配置告警时就可以直接引入。

### 网络延时或任务重启监控

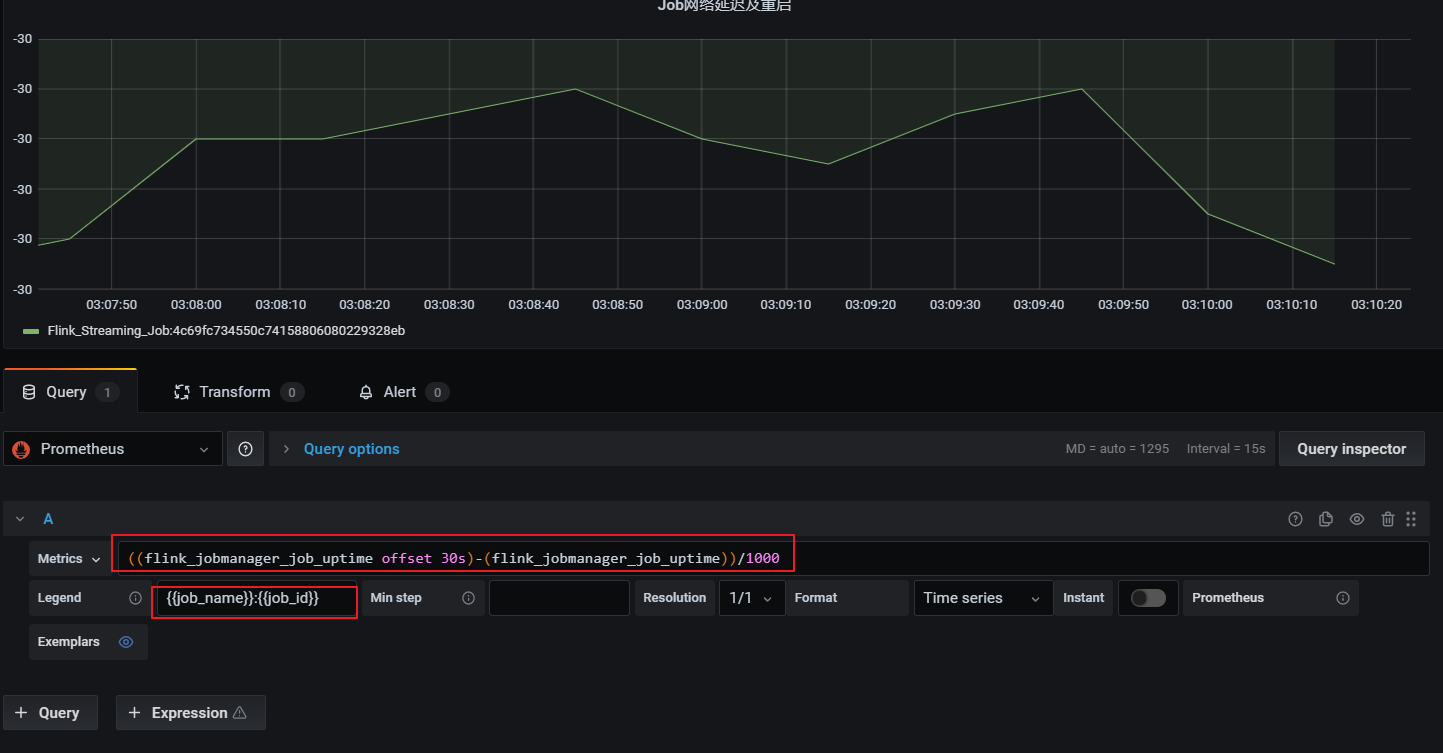
这个告警也是基于flink\_jobmanager\_job\_uptime 指标，在出现网络延时或者重启后进行监控通知，监控指标如下：

((flink\_jobmanager\_job\_uptime offset 30s)-(flink\_jobmanager\_job\_uptime))/1000

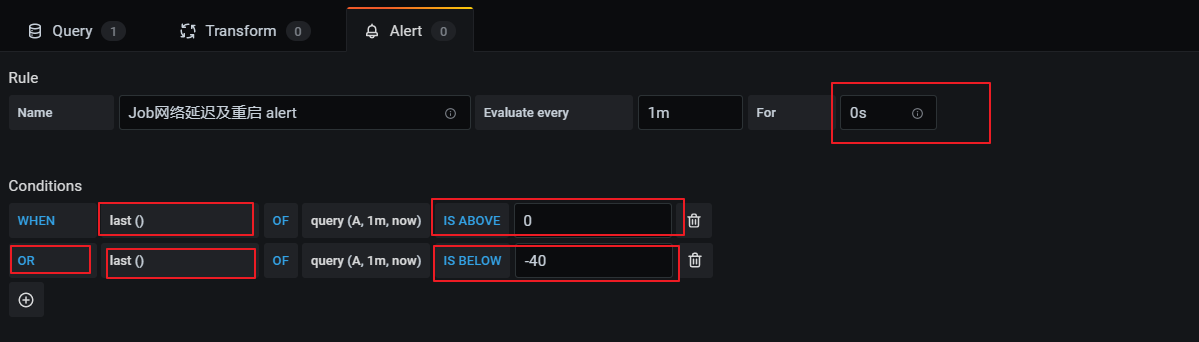
1）延时会导致值突然小于-30（正常情况为-30）

2）重启会导致flink\_jobmanager\_job\_uptime指标清零从新从0值上报，导致查询公式值突然大于0（正常情况为-30）

* 添加监控项

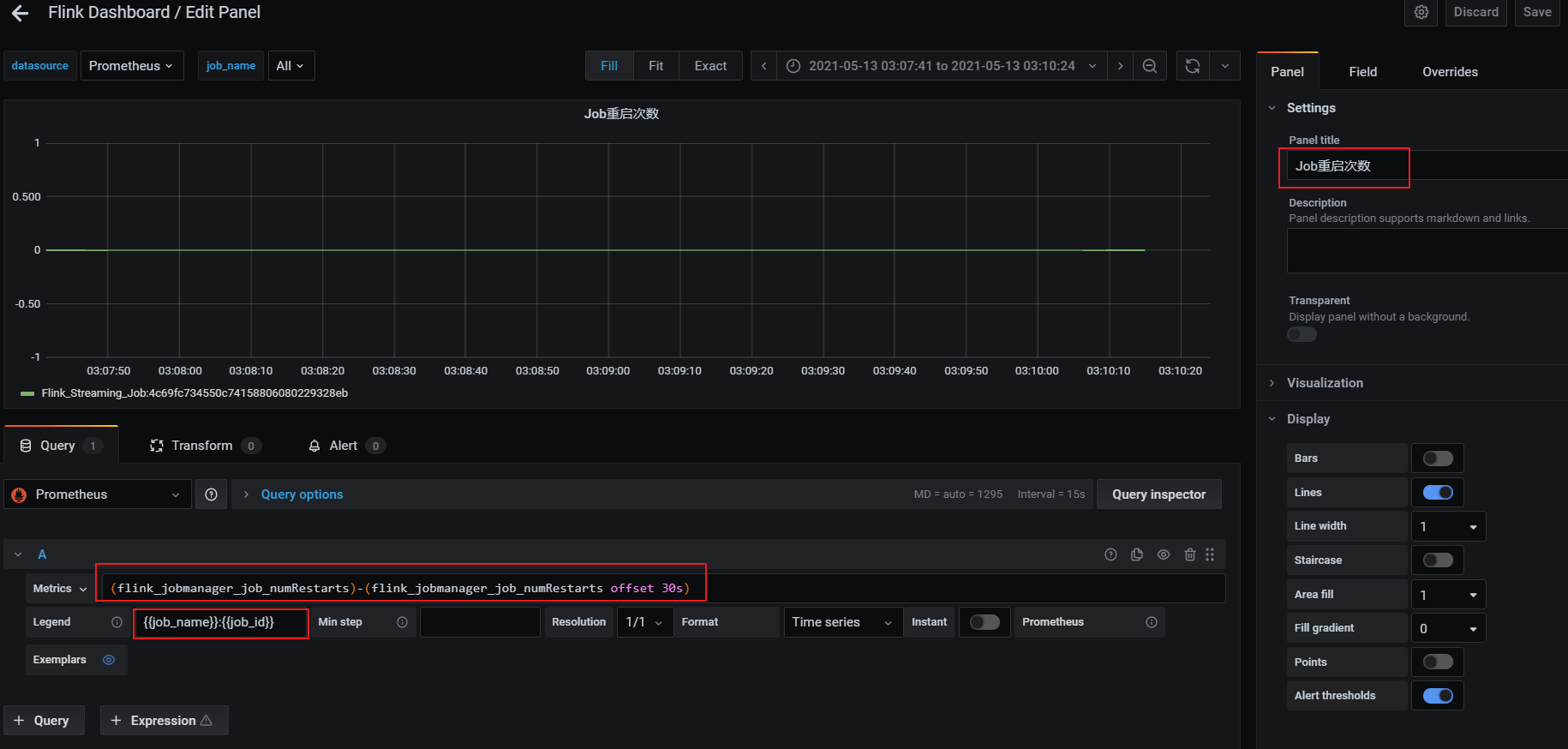


* 配置告警规则：



### 重启次数

基于flink\_jobmanager\_job\_numRestarts 指标，表示flink job的重启次数。一般设置重启策略后，在任务异常重启后这个数值会递增+1。可以单纯的监控重启次数，也可以每次重启都进行告警（差值）。



利用当前值减去30秒前的值，如果等于1证明重启了一次。

* 添加告警规则：

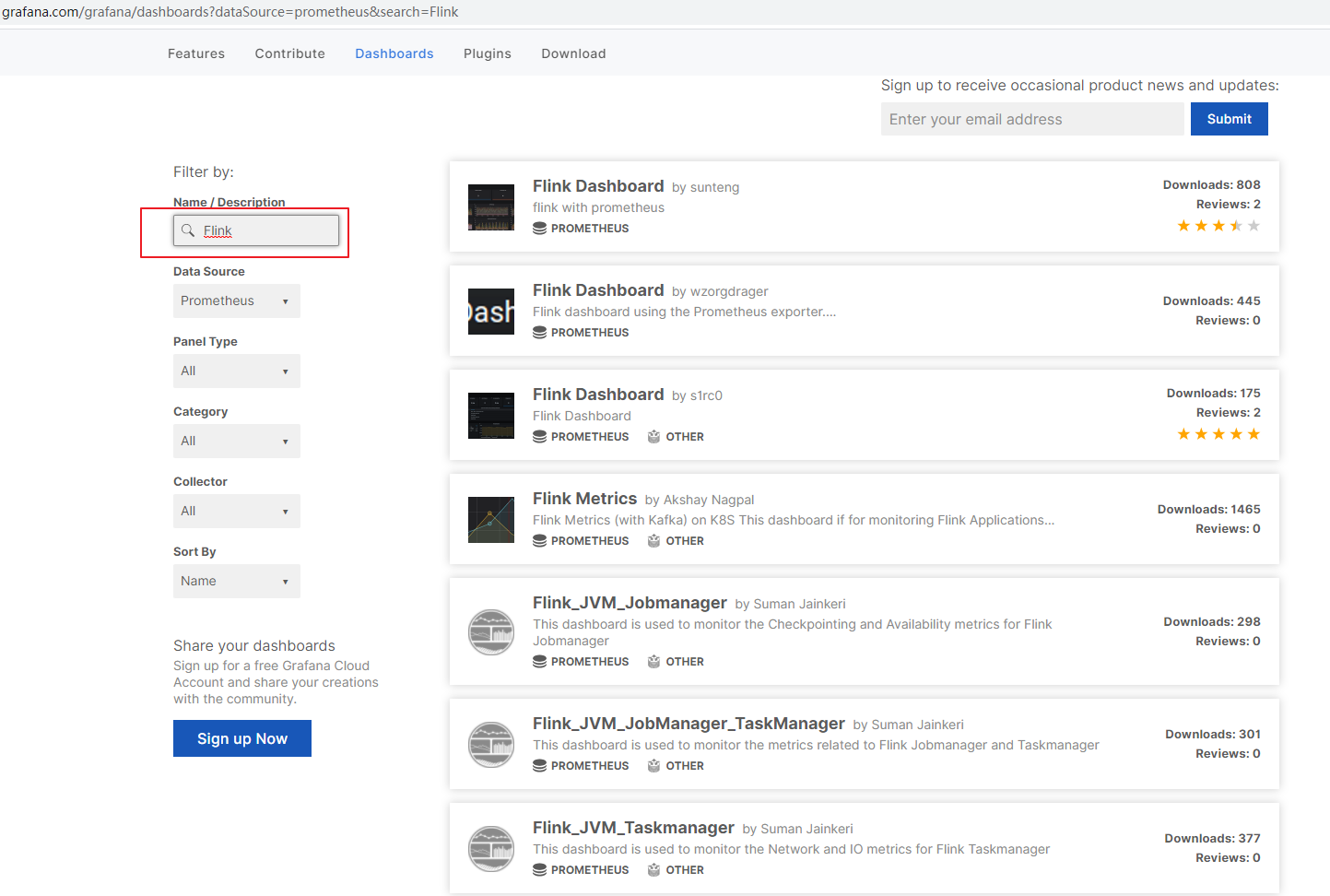


## 直接添加Flink模板

手动一个个添加Dashboard比较繁琐，Grafana社区鼓励用户分享Dashboard，通过<https://grafana.com/dashboards>网站，可以找到大量可直接使用的Dashboard模板。

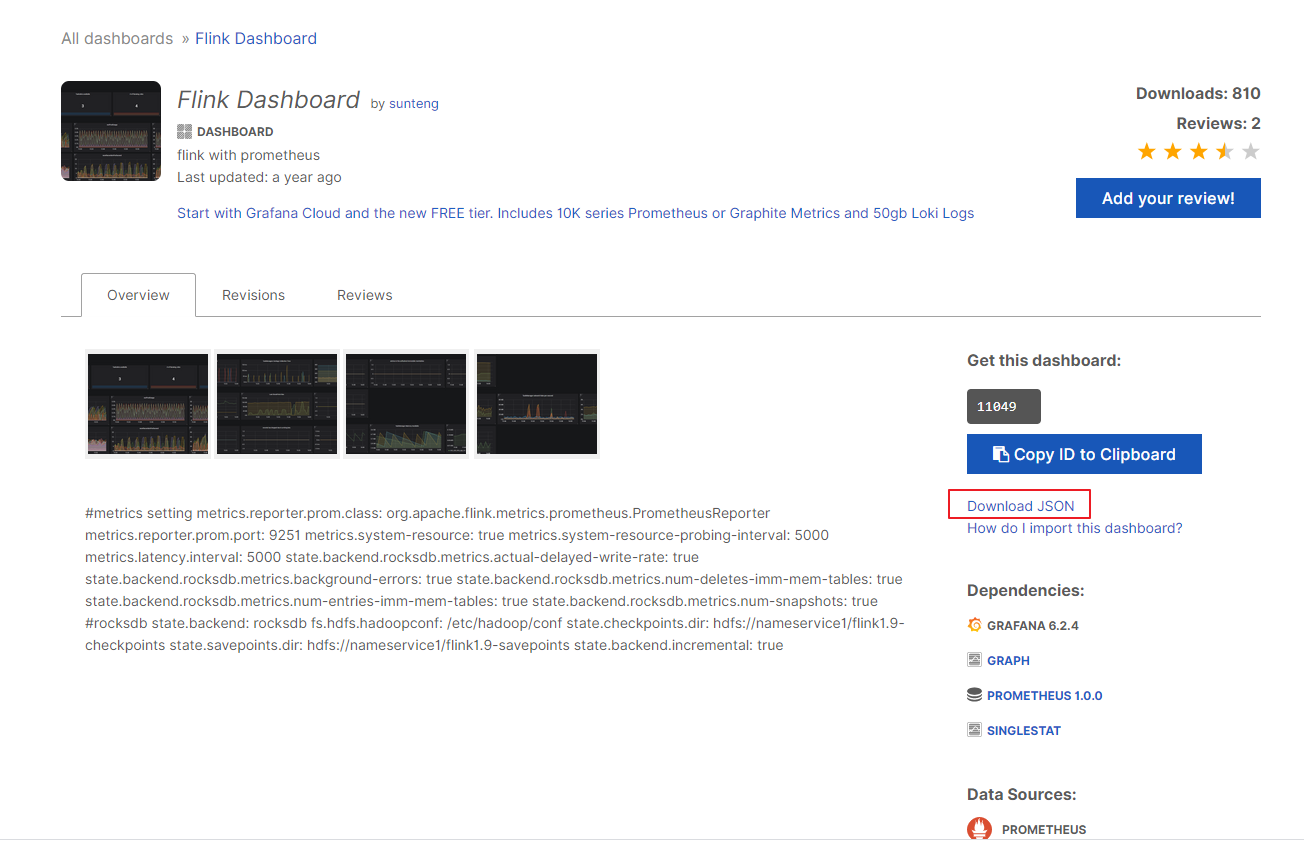
Grafana中所有的Dashboard通过JSON进行共享，下载并且导入这些JSON文件，就可以直接使用这些已经定义好的Dashboard：

进入官网，搜索Flink模板：

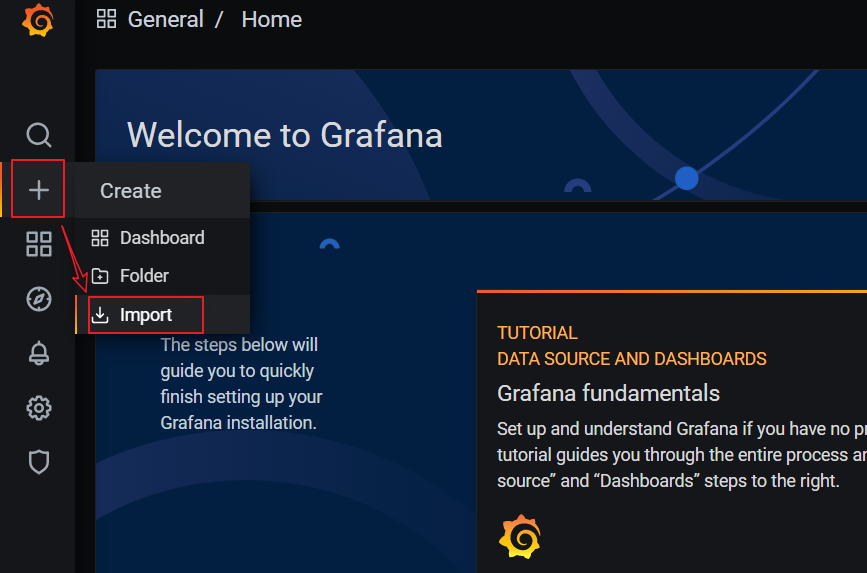


选择自己喜欢的模板（800+下载的这个模板相对指标较多）

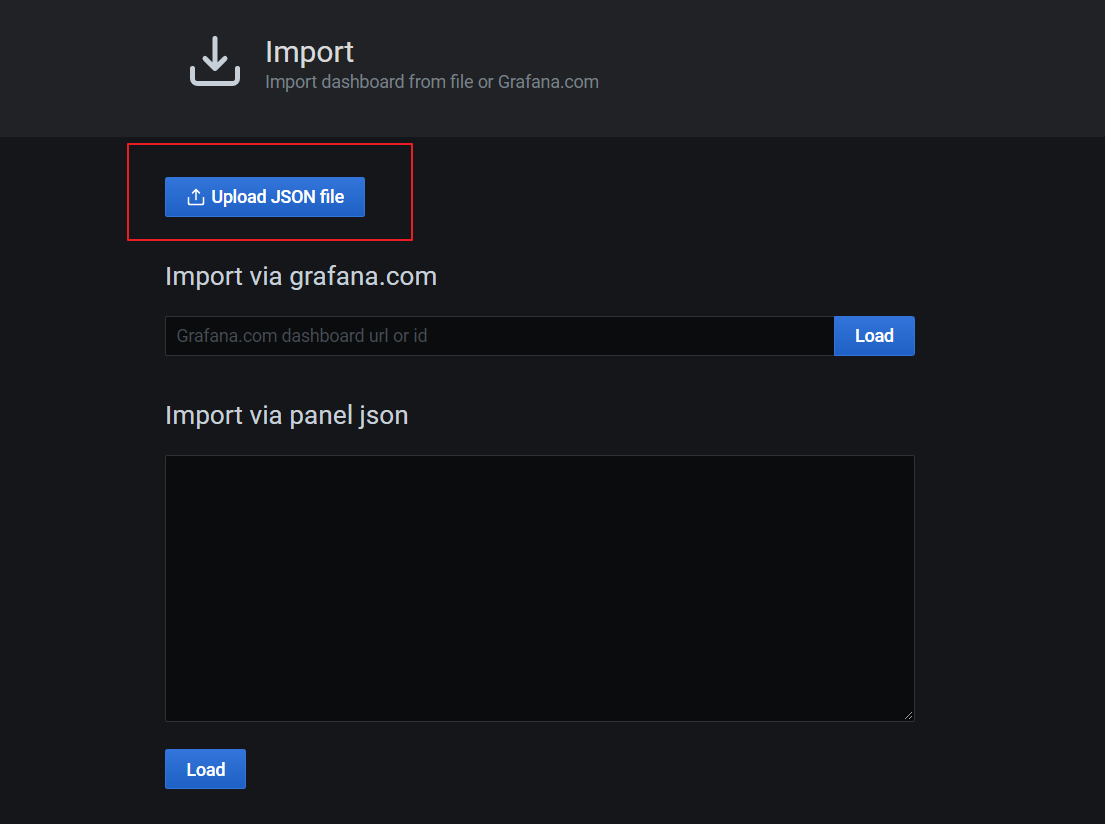
选中跳转页面后，点击 Download JSON：



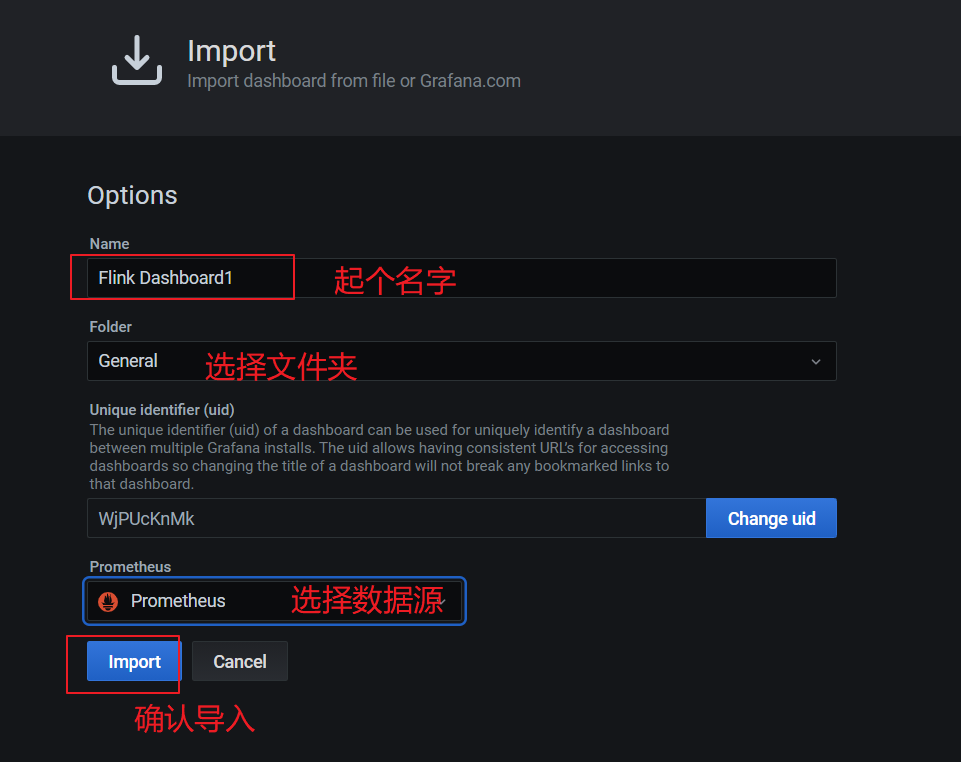
点击左侧 ”+”号，选择import：



上传JSON文件：



配置模板信息：



导入完，在首页即可看见添加的仪表盘，点击进去查看：



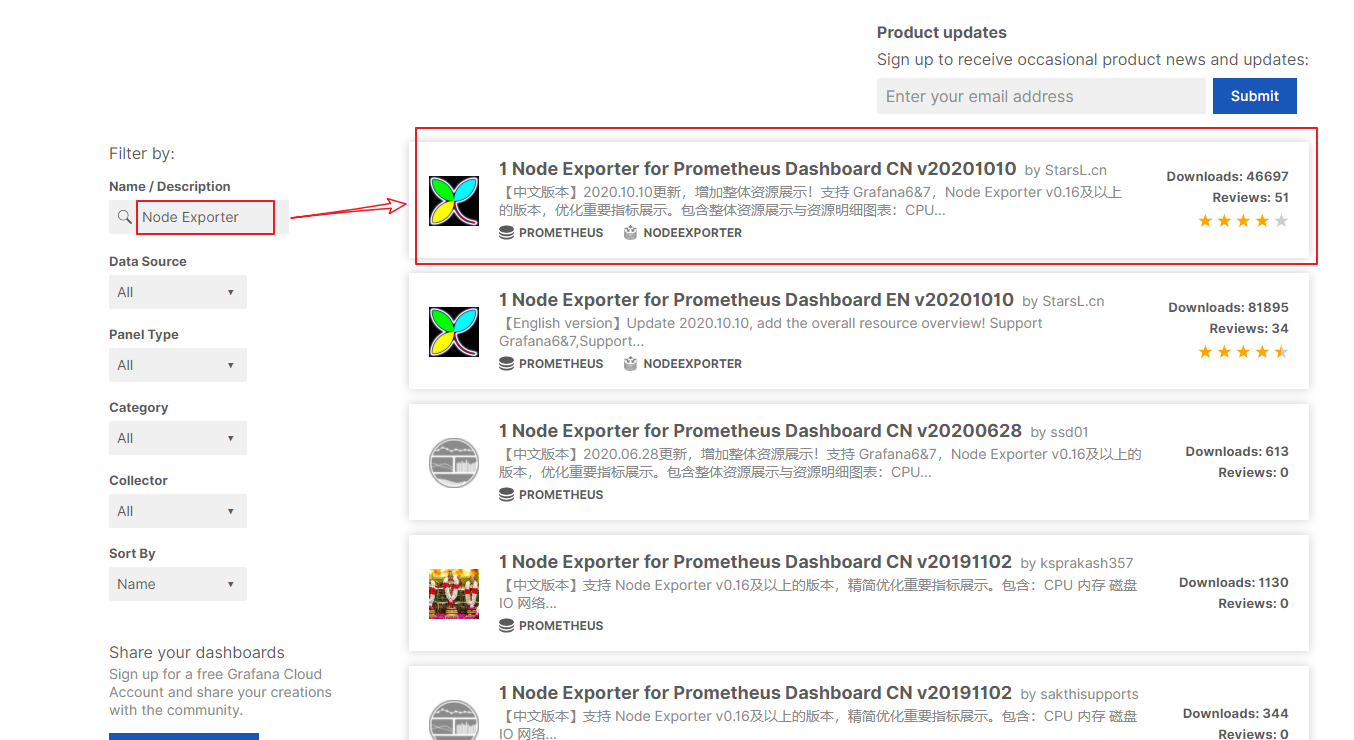
正常提交job，即可在grafana看到相关监控项的情况。

注意：代码里env.execute(“作业名”),最好指定不同的作业名用于区分，不指定会使用默认的作业名：Flink Streaming Job，在Grafana页面就无法区分不同job！！！

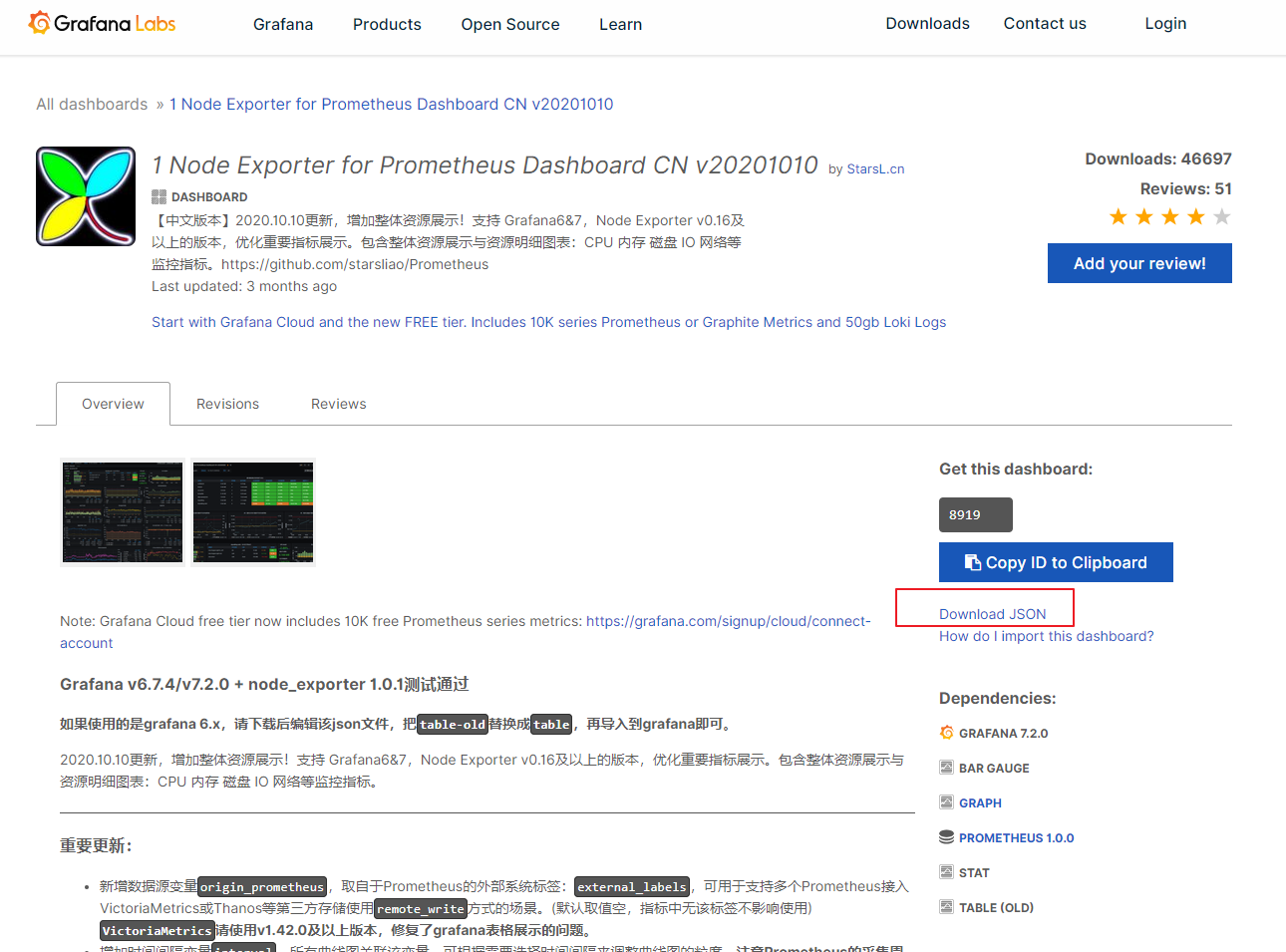
## 添加Node Exporter模板

同5.3，进入https://grafana.com/dashboards 页面，

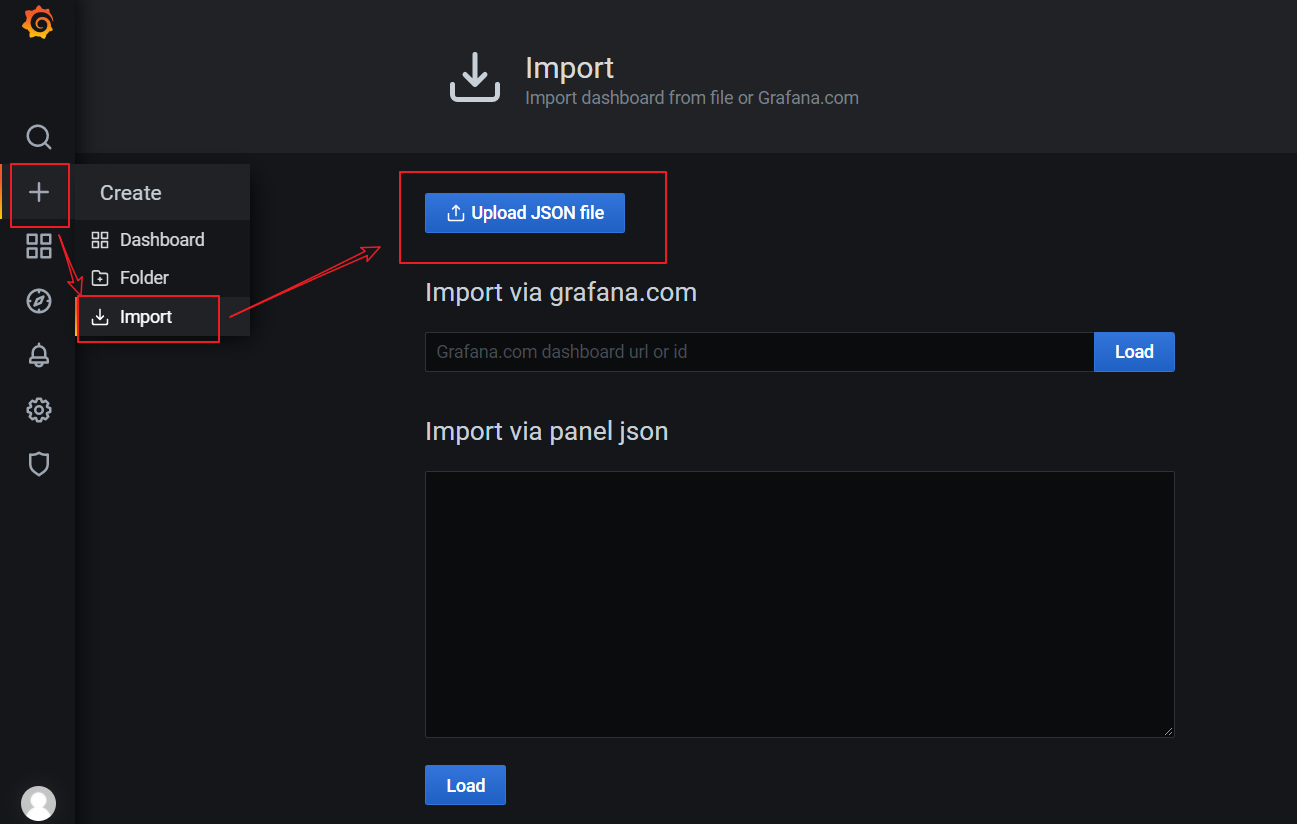
* 搜索Node Exporter，选择下载量最高的中文版本：

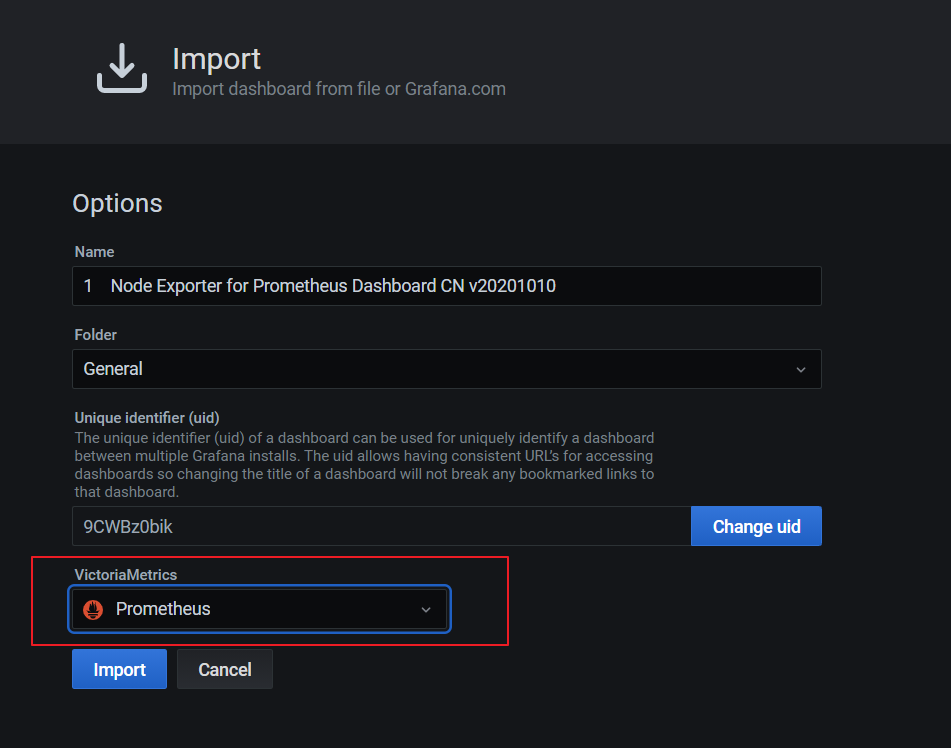


* 下载模板json文件



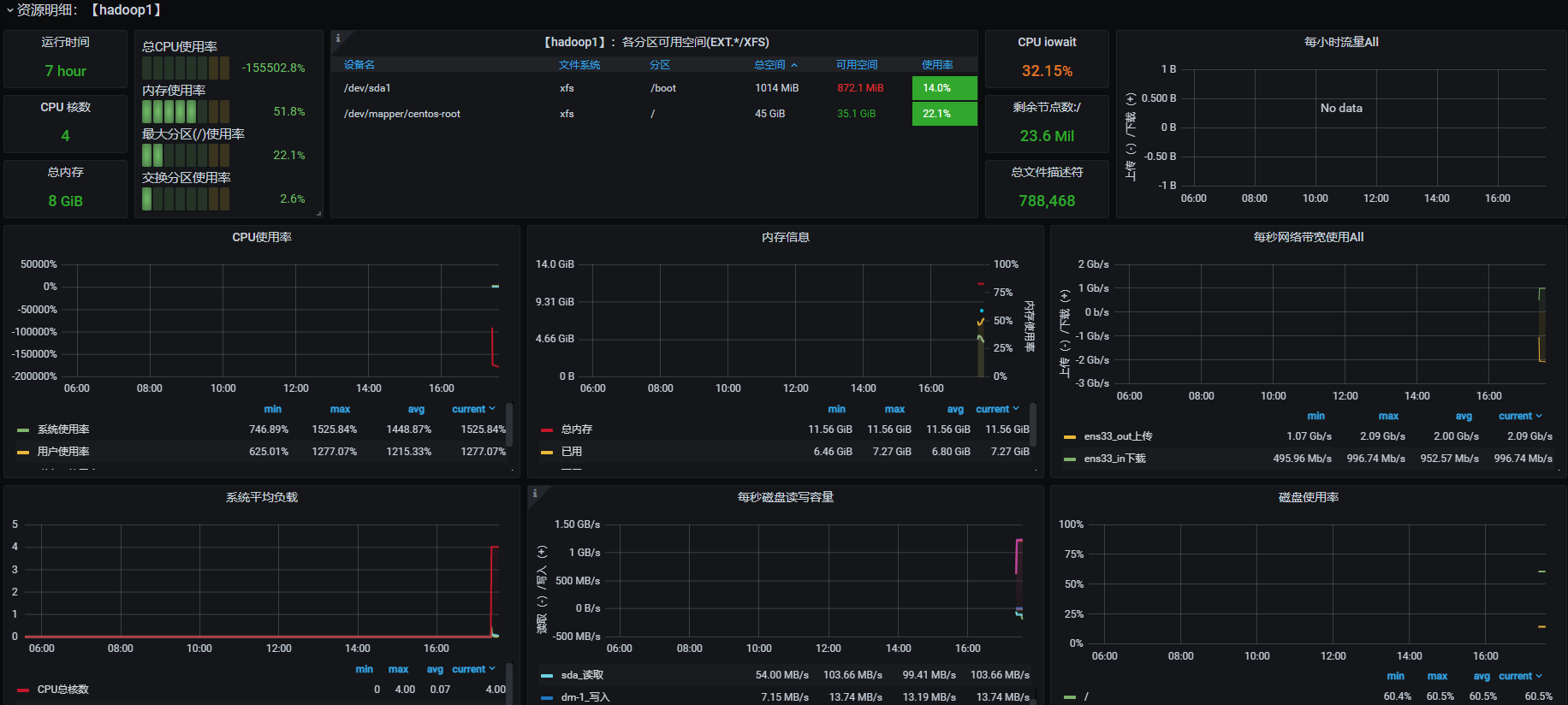
* 在Grafana中导入模板：

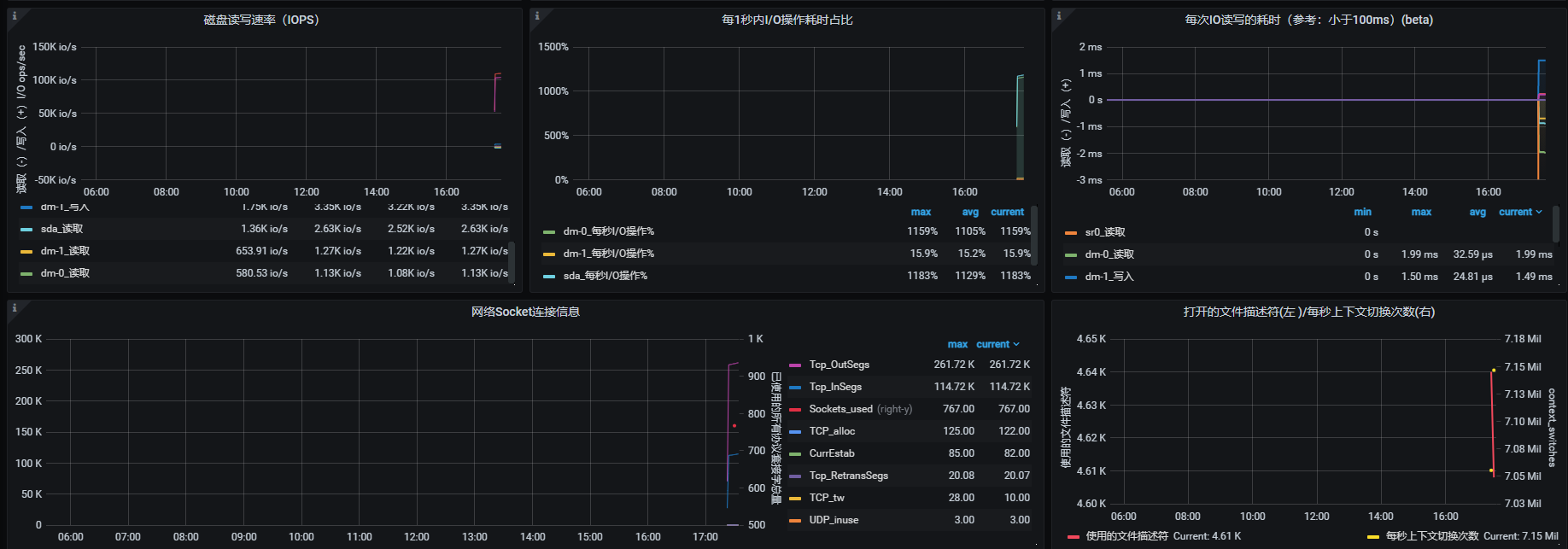




* 欣赏酷炫又详细的监控页：







## 组件启停脚本

* 进入到/home/atguigu/bin目录下，创建脚本flink-monitor.sh

#!/bin/bash

case $1 in

"start"){

echo '----- 启动 prometheus -----'

nohup /opt/module/prometheus-2.26.0/prometheus --web.enable-admin-api --config.file=/opt/module/prometheus-2.26.0/prometheus.yml > /opt/module/prometheus-2.26.0/prometheus.log 2>&1 &

echo '----- 启动 pushgateway -----'

nohup /opt/module/pushgateway-1.4.0/pushgateway --web.listen-address :9091 > /opt/module/pushgateway-1.4.0/pushgateway.log 2>&1 &

echo '----- 启动 grafana -----'

nohup /opt/module/grafana-7.5.2/bin/grafana-server --homepath /opt/module/grafana-7.5.2 web > /opt/module/grafana-7.5.2/grafana.log 2>&1 &

};;

"stop"){

echo '----- 停止 grafana -----'

pgrep -f grafana | xargs kill

echo '----- 停止 pushgateway -----'

pgrep -f pushgateway | xargs kill

echo '----- 停止 prometheus -----'

pgrep -f prometheus | xargs kill

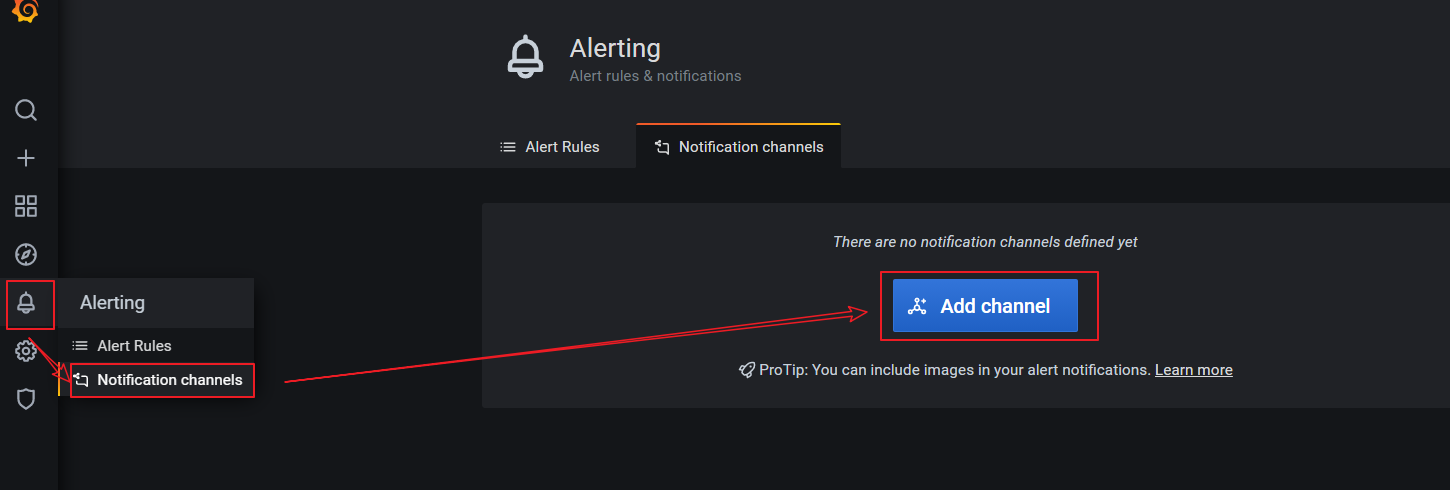
};;

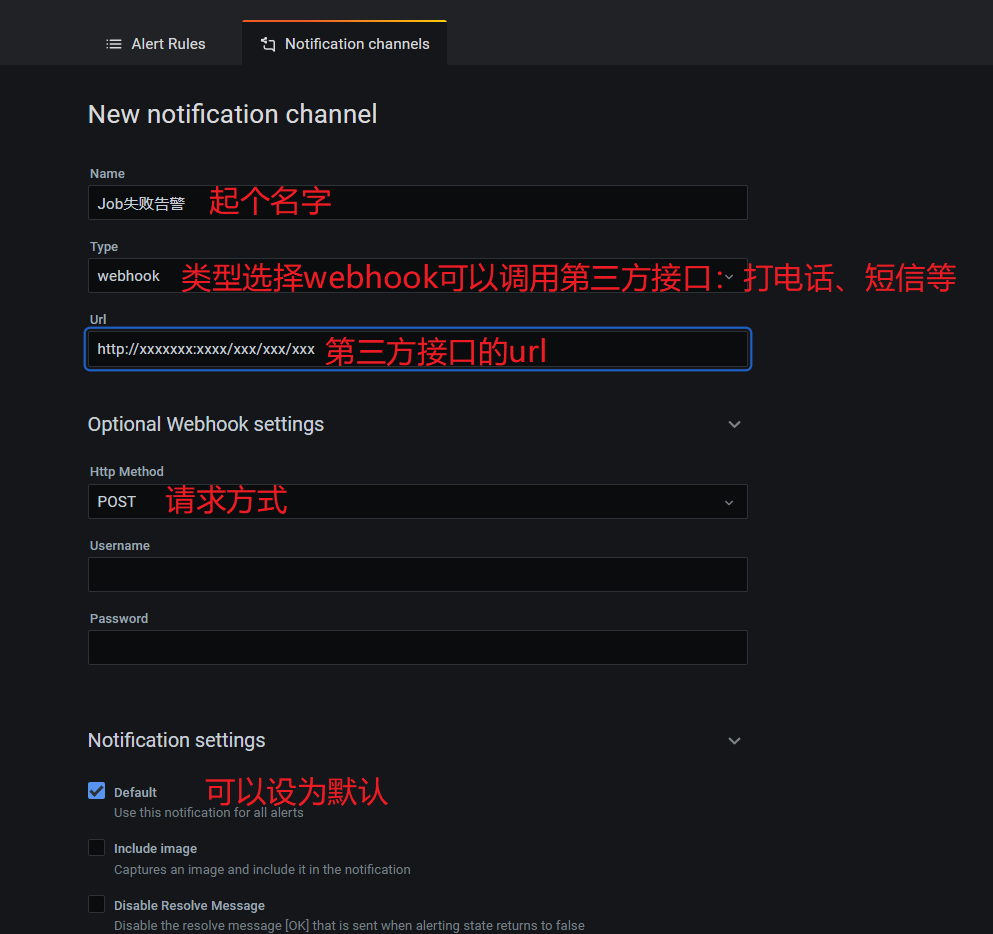
esac

* 脚本添加执行权限

[atguigu@hadoop1 bin]$ chmod +x flink-monitor.sh

# 告警通知

告警规则可以基于Prometheus的AlterManager组件（需要安装），也可以基于Grafana实现（如第5章）。下面配置告警规则触发后的通知方式：



比如接口是post请求，请求体body在配置告警规则时传入，比如5.2.1中：

