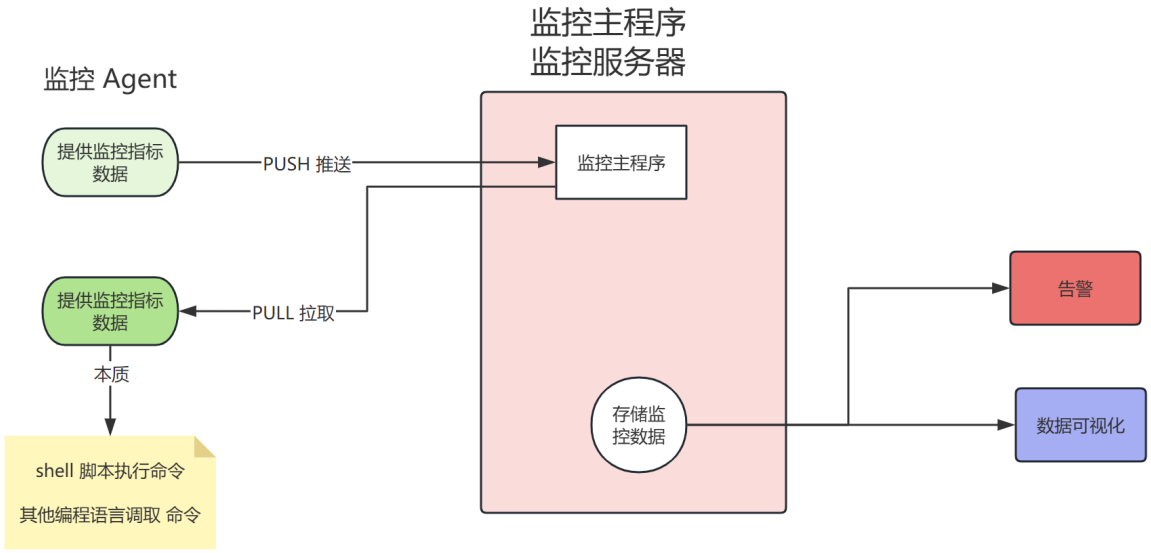
1. 认识 Prometheus

1.1. 介绍



Prometheus是一款开源的监控系统,是一个独立的开源项目，独立于任何公司进行维护。

- 成立于 2012年, 拥有非常活跃的开发人员和用户社区。
- 于2016年加入了云原生计算基金会(CNCF)，成为继Kubernetes之后的第二个托管项目。

Prometheus将其指标收集并存储为时间序列数据，即指标信息与记录它的时间戳一起存储。

1.2. 什么是指标

通俗地说，指标是用于记录用户想要获取的，对于某种应用或服务进行测量的数字结果。
这些数字结果因用户想要测量的具体内容因应用程序而异。

- 对于Web服务器，它可能是请求时间。
- 对于数据库，它可能是活动连接数或活动查询数等。

指标能够真实的反应出应用当时所处的运行状态。再结合其他的指标可以很好的反应出应用处于此运行状态的原因；通俗来说是运维和开发人员排查、处理问题的重要依据。假设您正在运行一个 Web 应用程序，并发现页面响应缓慢。您将需要一些信息来了解您的 web 应用程序发生了什么。例如，当请求数较高时，应用程序可能会变慢。如果您有请求计数指标，则可以发现原因并增加服务器数量以处理负载。

1.3. 时间序列

一组用连续且固定间隔的时间组成的时间。格式使用时间戳，比如：

```
1711277196
1711277316
1711277436
```

> 时间戳是指格林威治时间1970年01月01日00时00分00秒（北京时间1970年01月01日08时00分00秒）起至现在的总秒数

Prometeus 收集的每个指标数据都对应了一个时间戳，被一起保存在了Prometheus使用的数据库中。
比如，一组 HTTP 请求数的指标数据，可能是这样的：

```
109   1711277196
136   1711277316
168   1711277436
```

1.4. 特点

- 由指标名称和键/值对形式的标签（lable）构建的时间序列数据的多维数据模型。
- PromQL，一种灵活的查询语言，用于查询各种维度的指标数据。
- 不依赖分布式存储;可以存储在自我管理的单个服务器节点上的TSDB(时间序列数据库)。
- 服务端主动收集指标(pull 抓取): 指标数据的收集是通过服务端发送 HTTP 请求进行的。
- 也可以通过其他自定义的程序推送到一个中间网关服务，暂存之后再由服务端统一从这个中间网关抓取指标数据。
- 通过服务发现或从静态配置文件发现需要被监控的目标对象。
- 数据的展示支持多种模式的绘图和仪表板。

1.5. 组件

普罗米修斯生态系统由多个组件组成，其中许多组件是可选的：

- **Prometheus server**

  主要组件，用于抓取和存储指标数据

- **Exporter**

  实际提供监控指标数据的服务，Prometheus server 对这些 Exporter 主动发起 HTTP 请求，抓取指标数据并保存起来。监控不同的目标有不同的 exporter，比如监控服务器的 `node-exporter`，监控 mysql 的 `mysqld-erporter` 等。

- **Pushgateway**

  被动收集、暂存监控指标数据。
  通常用于被监控的对象不能和 Prometheus Server 直接通信的场景中。可以通过一个程序定期向 Pushgateway 推送指标数据，Promethes Server 会定期抓取 Pushgateway 的指标数据。

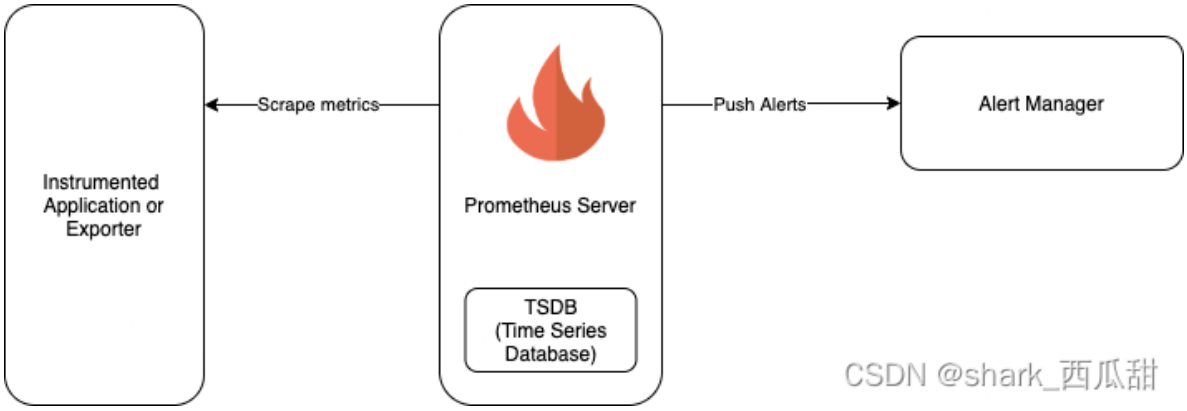
- **alertmanager**

  用于接收、汇总、去重、发送告警信息的服务。

大多数 Prometheus 组件都是用Go编写的，这使得它们易于构建和部署为静态二进制文件。

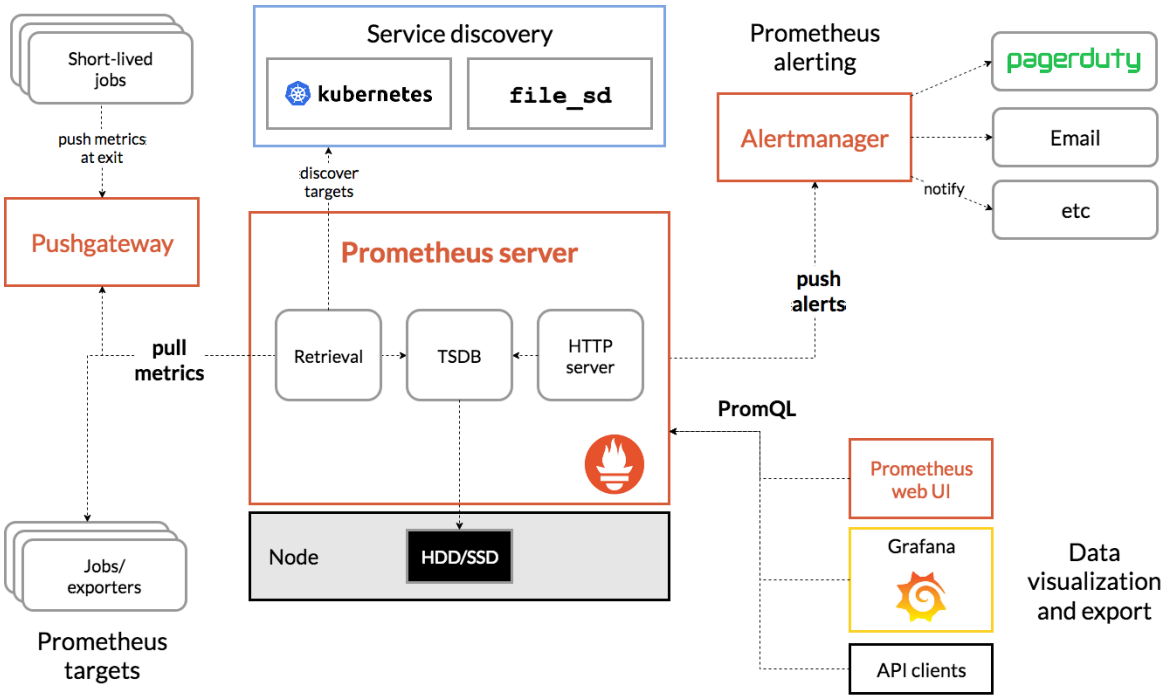
2. 架构

2.1. 基础架构



Prometheus Server 从应用程序或者 Exporter 抓取监控指标，存放在自己管理的一个 TSDB 中。在 Prometheus Server 中会定义一些规则，如果某个监控指标数据符合了某个规则，就会触发一条告警信息，此时 Prometheus Server 会根据具体的配置，把此条告警信息推送给 Alert Manager。

2.2. 完整的生态系统架构



> 图片来自官方文档

从上图中可以看出，Prometheus完整的生态系统中多出了几个重要的组件和服务。

Pushgateway 承担了中间网关的角色，其他程序可以主动把自己所监控对象的指标推送给 Pushgateway。

Server discover Prometheus Server 通过它发现 kubernetes 集群核心组件（Pod,Svervice等）的指标。

PromQL Prometheus 自己的查询语言，用于查询TSDB 中的指标数据。

Prometheus web UI Prometheus 的 web 页面，可以查看查询指标，查看监控对象的简易图表。

Grafana 它通过 PromQL 查询不同维度的指标，用非常华丽合适的图表展示出来。

3. 部署

3.1. 下载

点击官方下载链接，下载符合自己系统的版本

Prometheus 组件大部分是使用 GO 语言编写，部署起来没有什么依赖，只需要区分 CPU 大家架构，和选择自己喜欢的版本即可。

此文档采用已经编辑好的静态二进制的版本部署，截止写稿时(2024年3月38日)最近版本是 **prometheus-2.45.4.linux-amd64.tar.gz**

下载到服务器

```
wget
https://github.com/prometheus/prometheus/releases/download/v2.45.4/prometheus-2.45.4.linux-amd64.tar.gz
```

3.2. 解压并创建软链接

```
tar -xf prometheus-2.45.4.linux-amd64.tar.gz -C /usr/local/
ln -s /usr/local/prometheus-2.45.4.linux-amd64 /usr/local/prometheus
```

3.3. 创建进程管理文件

`/etc/systemd/system/prometheus.service`

```
[Unit]
Description=Prometheus 监控服务
After=network-online.target remote-fs.target nss-lookup.target
Wants=network-online.target

[Service]

# 2. EnvironmentFile=-/etc/default/prometheus

# 3. 下面的内容是启动命令和启动参数，是一行
ExecStart=//usr/local/prometheus/prometheus --
storage.tsdb.path=/usr/local/prometheus/data --web.enable-lifecycle --web.read-
timeout=5m --storage.tsdb.retention=15d --web.max-connections=512 --
query.timeout=2m --query.max-concurrency=20  --
config.file=/usr/local/prometheus/prometheus.yml

KillSignal=SIGQUIT

Restart=always

RestartPreventExitStatus=1 6 SIGABRT

TimeoutStopSec=5
KillMode=process
PrivateTmp=true
LimitNOFILE=1048576
```

```
LimitNPROC=1048576

[Install]
WantedBy=multi-user.target
```

启动服务

```
systemctl daemon-reload

# 4. 授权开机自启的同时启动服务
systemctl enable --now prometheus

systemctl status prometheus
```

默认监听端口 **9090**

```
ss -ntal |grep 9090
```

3.4. 程序启动参数

使用如下命令可以获取到所有的启动参数

```
prometheus --help
```

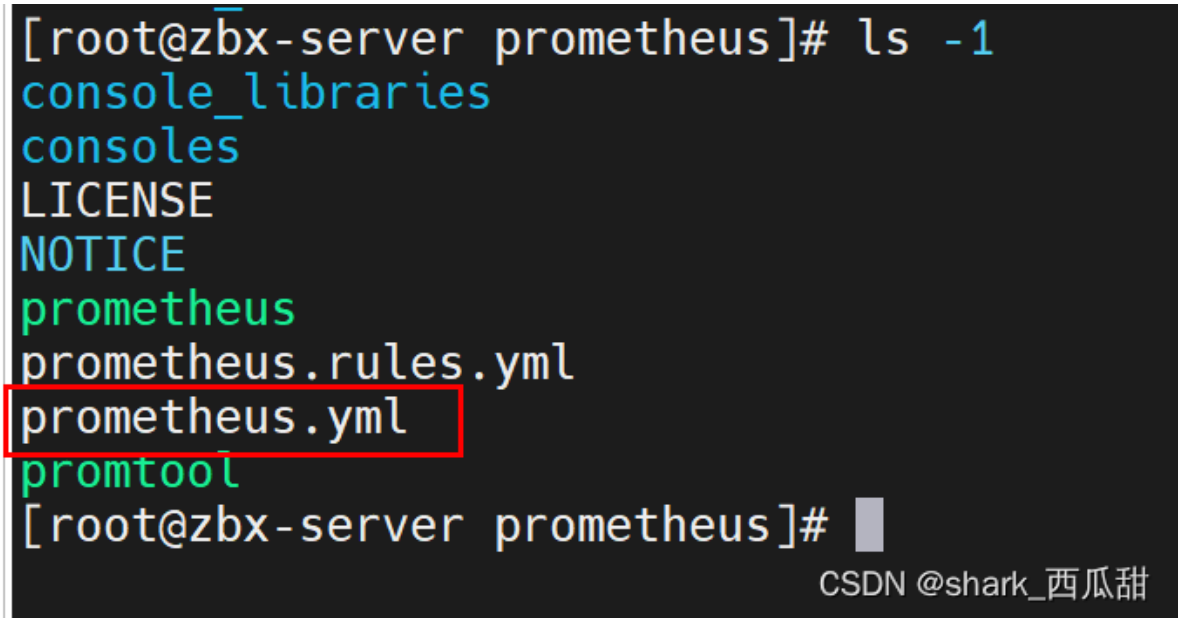
如下是常有的部分启动参数

```
--web.listen-address="0.0.0.0:9090"
配置监听地址和端口

--config.file=/etc/prometheus/prometheus.yml
指定配置文件

--storage.tsdb.path=/path/to/data
指定数据保存的目录，最后一级的目录(data)会自动创建

--web.enable-lifecycle
热加载配置，就是修改配置文件后，重新加载生效，而无需重启服务。

--web.read-timeout=5m
请求连接持续等待时长

--web.max-connections=512
最大并发连接数

--storage.tsdb.retention=15d
采集的监控数据保留在内存或者磁盘中的最长时间
15-30天为宜

--storage.tsdb.path="/data"
存储数据的路径

--query.timeout=2m
单次查询等待时长，超时中断本次查询
```

```
--query.max-concurrency=20
接受的最大并发查询数
```

4. 配置文件介绍

二级制方式下载的压缩包中附带了一个名 `prometheus.yml` 的示例配置。



4.1. 配置文件布局

配置文件主要有四部分组成：

```yaml
# 1 全局配置
global:
  ...

# 2 告警规则配置，从哪里加载告警规则
rule_files:
  - ...

# 3 告警管理器配置
alerting:
  ...

# 4 配置被监控对象
scrape_configs:
  - ...
```

4.2. 全局配置

在全局配置中，可以配置 Prometheus 抓取指标的间隔时间等。

```
global:
  scrape_interval: 30s # 将抓取间隔时间设置为每 30 秒一次。默认值为每1分钟一次。

  evaluation_interval: 5s # 规则重启评估的间隔时间设置为每隔15秒评一次。默认值为每1分钟一次。
```

4.3. 告警规则配置

可以指定一个具体的文件，或者使用通配符匹配的文件路径，作为加载告警规则的路径。

是一个列表，可以指定多个。

生产环境中通常会用不同的目录名称区分不同的监控对象类型。

Prometheus 第一次启动加载后评估一次规则，之后每隔 `evaluation_interval` 指定的间隔时间对规则进行一次评估。

```
rule_files:
  # - "/usr/local/prometheus/rules/*.rules.yml"
  # - "first_rules.yml"
  # - "second_rules.yml"
```

4.4. 告警管理器配置

可以配置多个告警管理器

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          # - alertmanager的IP或者主机名:9093
```

4.5. 配置自动发现

4.5.1. 从 Prometheus 自身配置文件获取被监控对象

这里的配置就是 Prometheus 如何获取被监控对象，或者被监控对象列表。

```
scrape_configs:
    # Prometheus 自己
  - job_name: "prometheus"
    static_configs:
      - targets:
          - "localhost:9090"

    # 下面的是示例配置
  - job_name: "beijing"
    static_configs:
      - targets:
          - "192。168.146.138:9111"
        labels:
          group: 'beijing'
```

使用一个一个的 Job 方式进行管理，Job 使用 `scrape_configs` 下面的 `job_name` 指定一个 Job 名称进行配置。会以标签 `job_name=<job name>` 的方式被自动添加到其下面的所有被监控对象的每一个指标数据中。

通常通过 `static_configs` 配置项下的 `targets` 中的列表，称为静态文件的方式发现被监控对象。

`labels` 可以配置向抓取到的每条指标数据中添加的自定义的标签。

对被监控对象抓取指标时，默认使用 **http** 协议，URL 使用 **'/metrics'**

检查配置文件语法
在部署的主目录下有个 `promtoll` 程序，可以检查你配置文件的语法。

```
[root@prome prometheus]# ./promtool check config prometheus.yml
Checking prometheus.yml
 SUCCESS: prometheus.yml is valid prometheus config file syntax
```

4.5.2. 从一个个的独立文件中获取被监控对象

`file_sd_config` 配置项，用于文件的服务发现提供了一种更通用的方式来配置静态目标，并作为插入自定义服务发现机制的接口。

指定的文件的更新会被Prometheus server 监控，文件的变化会在配置的时间点更新到服务中。

配置示例：

```
scrape_configs:
    # Prometheus 自己
  - job_name: "prometheus"
    file_sd_config:
      - files:
          - ./path/to/*sd.yml
```

`path/to/some-sd.yml`

```
- targets:
    - "localhost:9090"
```

4.6. 热更新配置文件

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

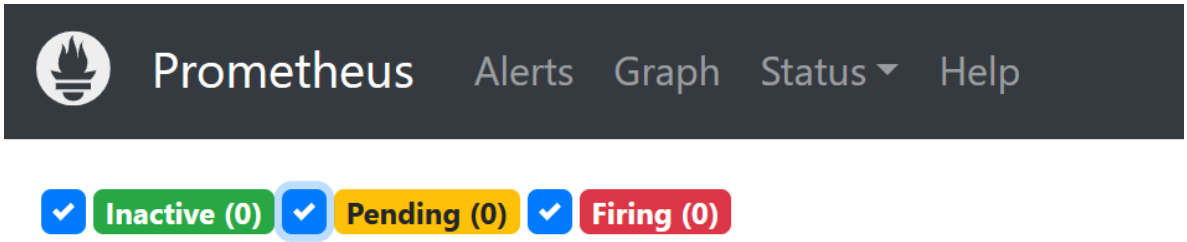
如果使用 容器部署，并且配置文件使用的是绑定挂载方式，需要使用一般目录绑定挂载，这样宿主机中目录中的内容，才能实时更新到容器内。否则配置文件不会得到更新。

5. 管理页面介绍

5.1. 菜单栏

Alerts

这里可以看到到已经配置并加载成功的告警规则和规则触发的告警。



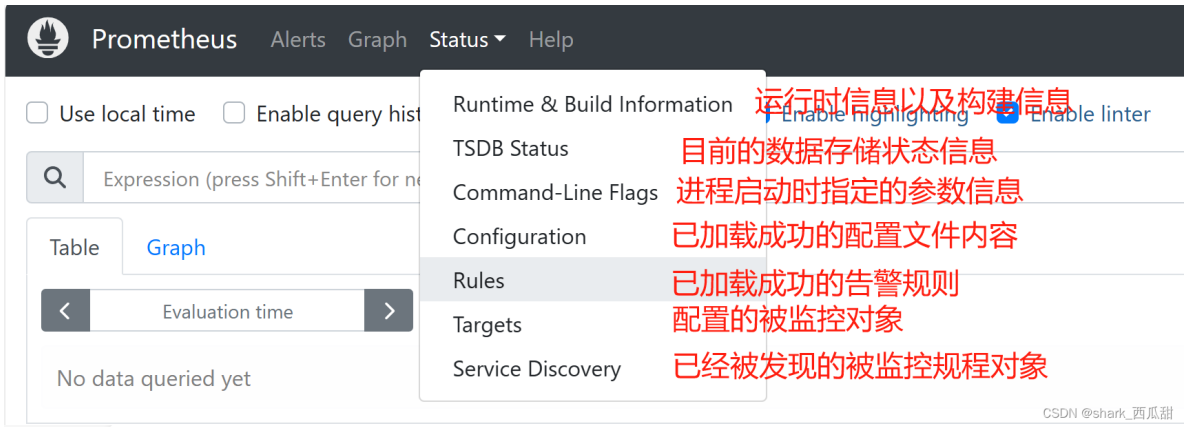
后面讲解告警章节的时候在详细介绍。

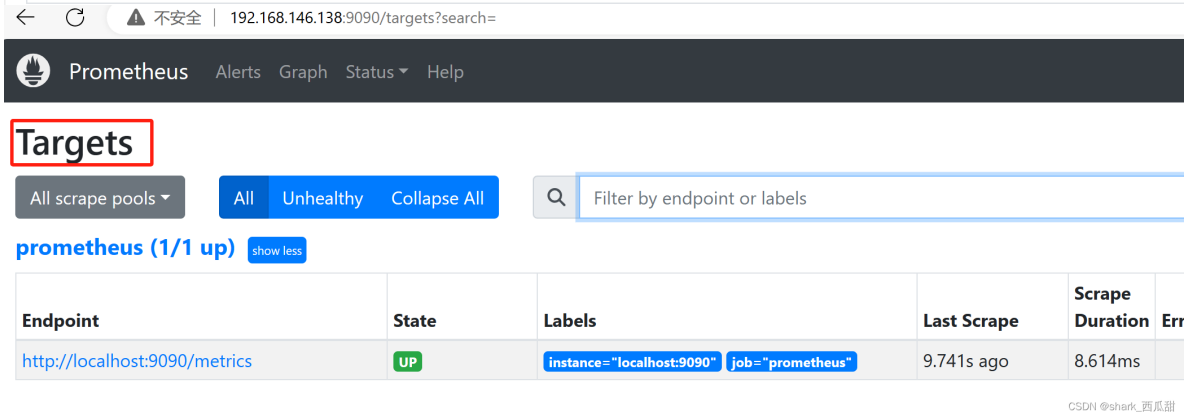
Help

可以跳转到官方的文档页面。

Status

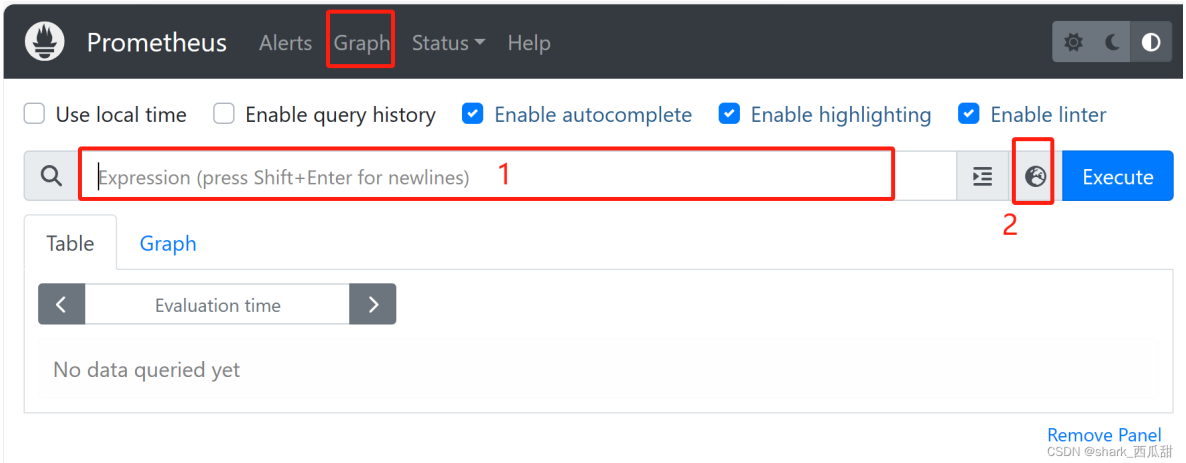
可以看到 Prometheus 的各种信息。





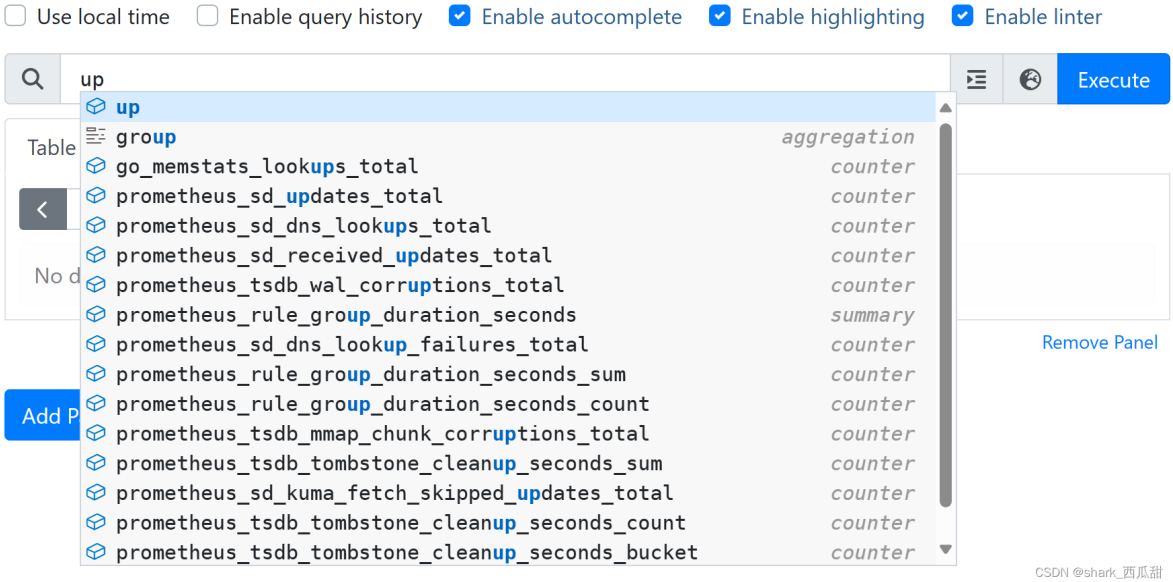
Graph

在这里可以输入需要查询的指标。

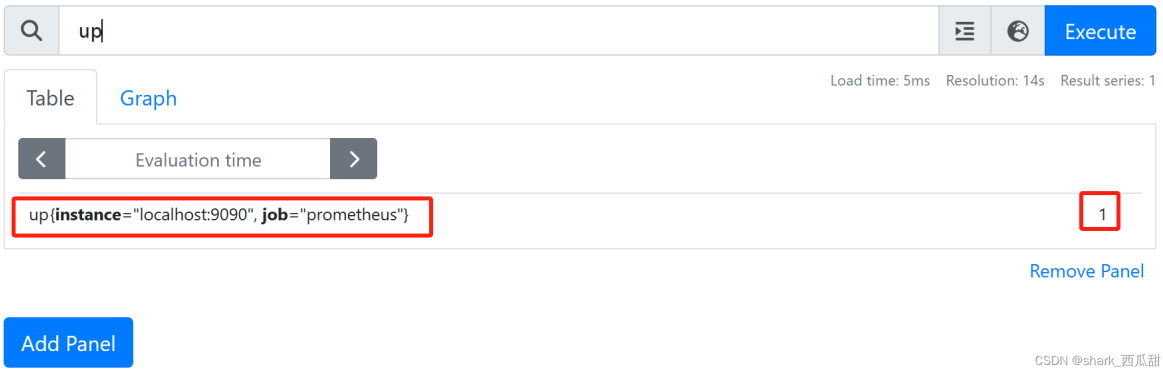


上图中 1 是输入框，需要输出的是 PromQL 的表达式。
上图中 2 点击后，可以查看目前所有的指标。

默认 Prometheus 自身是被监控的，其中有一个指标 `up`



可以看到输入框是支持自动模糊匹配提示的，输入指标名称后，回车即可触发查询，结果默认会以表格的方式展现在下方。



上图中左侧就是指标的完整表示形式，右侧就是这个指标的值。

6. 指标详解

6.1. 指标表示方式

指标由 指标名称和标签组成

在 Prometheus 中，通常使用以下表示法进行标识：

`指标名称{<标签名>=<标签值>, ...}`

例如:

```
up{instance="localhost:9090", job="prometheus"}
```

> up 指标名
>
> instance 标签名， 对应的标签值是 localhost:9090

6.2. 指标名称规范

必须是 ASCII 字母和数字，以及下划线和冒号的任意组合。不能以数字开头，双下划线开头的是保留给内部使用。
示例：

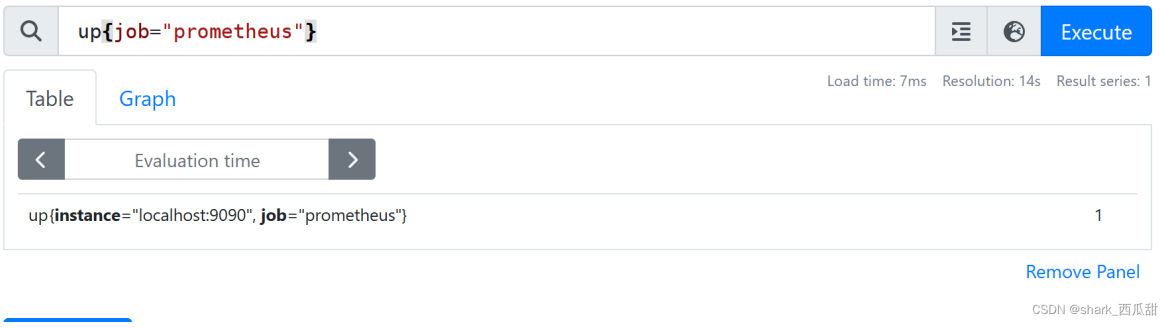
```
http_requests_total_139
```

6.3. 标签 lable

标签用于对某个指标数据的进一步标识，便于对此指标数据的进一步描述，或者对这个指标数据的特性的标记，可以用于查询此指标数据的过滤条件。

可以使用指定的标签，对同一指标名称的监控数据进行特定维度的查询。例如，想查看某些被监控的服务器的状态是否在线，可以使用如下表示方法:

```
up{job="prometheus"}
```



> 值若是字符类型，必须用双引号包裹起来。

这里是通过 Job 的名称作为条件的查询，当然也可以根据已经配置的其他标签作为条件，比如某个地方的机房名称，数据库等，当然这都需要先进行配置。

标签名称规范和指标名称规范一致。

示例： `method` , `host_name`

标签使用 `name=values` 的方式存储，例如: `host=web1`, 标签名称是 `host`, 其值是 `web1`

一个指标数据可以拥有多个标签,一个标签也可以对应多个相同或者不同的数据，就是多对多的关系。

为了帮助你理解标签的作用，你可以简单的理解标签就像是MySQL 查询是给的条件，标签名好比字段名，标签值好比字段对应的某个具体的数据。

标签： `host="web1"`

转换为 MySQL中查询的条件: `where host="web1"`

后面章节讲到查询的时候，我会具体说明如何使用标签。

6.4. Prometheus 自动生成的标签和指标数据

当 Prometheus 抓取目标时，它会自动将一些标签附加到抓取的指标数据中，用于识别抓取的目标。

这些自动添加的标签是：

- job：配置文件中给被监控对象已配置作业名称。
- instance：配置文件中 target 列表中的 `<host>:<port>` 部分。

对于抓取的每个实例，Prometheus 会添加如下的指标数据：

- `up{job="<job-name>", instance="<instance-id>"}: 1` 如果实例运行正常，即可访问，其值为 `1` ,抓取失败则为 `0`。**对于实例可用性的监控非常有用**
- `scrape_duration_seconds{job="<job-name>", instance="<instance-id>"}:` 抓取一个指标的的持续时间，就是响应时间。
- `scrape_samples_scraped{job="<job-name>", instance="<instance-id>"}`：被监控对象暴露出的指标总个数。

7. 基本认证

7.1. 配置文件

创建一个 YAML 文件，并使用程序启动参数 `--web.config.file` 指定，就能实现基本的用户名密码方式的认证。

YAML 文件示例

```
basic_auth_users:
  # 格式 用户名:加密过的密码
  shark: $2y$10$GOpTzQuug7Wmai873xzm6esDImUQUJzsW/WCuH2IBM4k7JelGRMFq
```

7.2. 加密密码的生成

密码需要使用 bcrypt 加密。
生成加密密码的工具：

7.2.1. 可以使用 htpasswd 命令：

这个需要你安装 **httpd-tools** 软件包
执行如下名称获取加密的密码：

```
htpasswd -nBC 10 "" | tr -d ':'
```

```
[root@prometheus ~]# htpasswd -nBC 10 "" |tr -d ':'
New password:
Re-type new password:
$2y$10$GOpTzQuug7Wmai873xzm6esDImUQUJzsW/WCuH2IBM4k7JelGRMFq

[root@prometheus ~]#
```

7.2.2. 可以使用 python 中的模块

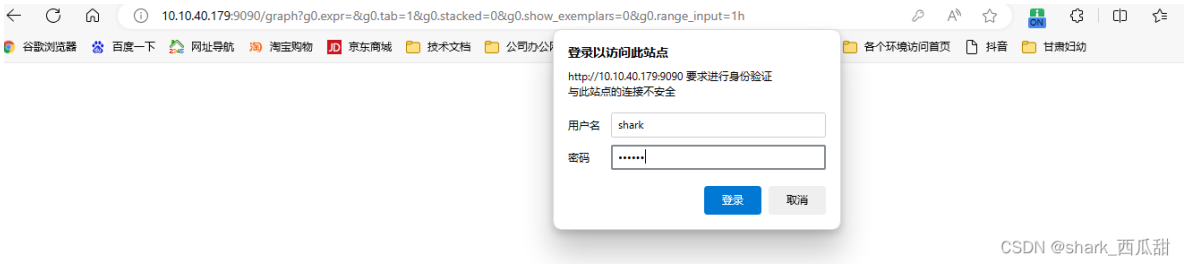
安装 python 包

```
pip install bcrypt==3.1.4
或者
pip3 install bcrypt==3.1.4
```

在 python 解释器中输入如下代码，生成密码

```
import  bcrypt
salt = bcrypt.gensalt()
bcrypt.hashpw('123456'.encode('utf-8'), salt).decode()
```

7.3. 访问验证



7.4. 修改配置文件，添加认证信息

当你配置完 Web 页面的认证后，你会发现 Prometheus 自身的监控不在线了

此时你需要修改配置文件，并添加上认证信息

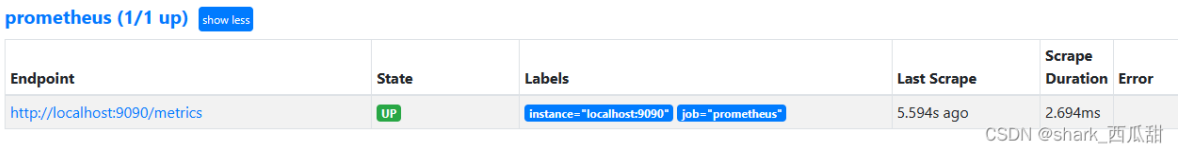
```
scrape_configs:
  - job_name: "prometheus"
    basic_auth:
      username: shark
      password: 123456
      #- password_file: "/usr/local/prometheus/web-auth.yml"
```

`password` 和 `password-file` 是互斥的，配置的值都需要是明文的密码。

7.5. 重新加载配置文件

```
curl -X POST --user 'shark:123456'  localhost:9090/-/reload
```

刷新浏览器



传送门

下一篇:2-云原生监控体系-使用node-exporter监控Linux服务器