那么我们如何 既能永久保存这些 监控图 ， 又可以让它更美观呢?
请关注 我们后面的篇章（Grafana + promethues）

大米運維課堂

最前沿開源監控 Prometheus專題講座

第十讲：企业级监控数据采集脚本开发实践

第十讲内容:

- pushgateway 的介绍
- pushgateway 的安装和运行和配置
- 自定义编写脚本的方法 发送pushgateway 采集
- 使用pushgateway的优缺点

（一） pushgateway 的介绍

pushgateway 是另一种采用被动推送的方式（而不是exporter主动获取）获取监控数据的prometheus 插件

在上篇中 我们对pushgateway已经做过介绍

它是可以单独运行在 任何节点上的插件（并不一定要在被监控客户端）

然后 通过用户自定义开发脚本 把需要监控的数据 发送给pushgateway
然后pushgateway 再把数据 推送给prometheus server

接下来咱们来实践

（二）pushgatway 的安装和运行和配置

pushgateway 跟 prometheus和 exporter 一样

下载地址
https://prometheus.io/download/#pushgateway

解压后 直接运行

github的官方地址

https://github.com/prometheus/pushgateway
又是以go语言开发的， 用go开发的程序 在安装运行时 就是这么得天独厚的 简单任性

同样 我们使用 daemonize 方式 讲pushgateway 放入后台运行

root 13980 13976 16 1月19 ? 4-19:53:51 /data/pushgateway/pushgateway
root 13981 13978 0 1月19 ? 04:15:08 /data/pushgateway/pushgateway1 -web.listen-address 0.0.0.0:9092

关于pushgateway的配置 主要指的是 在prometheus sever端的配置

我们来看下

```
- job_name: 'pushgateway'

  static_configs:
    - targets: ['localhost:9091','localhost:9092']
```

在prometheus.yml 配置文件中， 单独定义一个job
然后 target 指向到 pushgateway运行所在的 机器名和 pushgateway运行的端口 即可

这里我们发现了没有， localhost:9091/9092 大米老师企业中使用的 pushgateway开启了两个

在prometheus_server 本机上

为什么开启两个 我最后一个段落给大家解释

（三）自定义编写脚本的方法 发送pushgateway 采集

接下来 进入重头戏了
pushgateway 本身是没有任何抓取监控数据的功能的 它只是被动的等待推送过来

所以 让我们来学习 pushgateway 编程脚本的写法

如下是一段 生产环境中 使用shell 编写的 pushgateway脚本
用于抓取 TCP waiting_connection 瞬时数量

cat /usr/local/node_exporter/**node_exporter_shell.sh**

```
#!/bin/bash

instance_name=`hostname -f | cut -d'.' -f1`      #本机机器名 变量 用于之后的 标签

if [ $instance_name == "localhost" ];then # 要求机器名 不能是 localhost 不然标签就没有区分了
echo "Must FQDN hostname"
exit 1
fi

# For waitting connections
```

label="count_netstat_wait_connections"　　# 定一个新的 **key**
count_netstat_wait_connections=`netstat -an | grep -i wait | wc -l`
#定义一个新的数值 netstat中 wait 的数量

echo "$label : $count_netstat_wait_connections"

echo "$label $count_netstat_wait_connections" | curl --data-binary @- http://prometheus.server.com:9091/metrics/job/pushgateway1/instance/$instance_name

#最后 把 key & value 推送给 pushgatway

curl —data-binary

将HTTP POST请求中的数据发送给**HTTP服务器(pushgateway)**,
与用户提交HTML表单时浏览器的行为完全一样。
HTTP POST请求中的数据为纯二进制数据

从上面的 短短的十几行的脚本来看

其实最重要的 是两个标红的地方

count_netstat_wait_connections=`netstat -an | grep -i wait | wc -l`

后面红色部分 是我们通过Linux命令行 就简单的获取到了 我们需要监控的数据 TCP_WAIT数

http://prometheus.server.com:9091/metrics/job/pushgateway1/instance/$instance_name

最后这里 用POST 方式 把 key & value 推送给 pushgatway的URL地址

这个URL地址中 分成如下三个部分

http://prometheus.server.com:9091/metrics/job/pushgateway1
这里是 URL的主location

job/pushgateway1
这里是 第二部分 第一个标签： 推送到 哪一个prometheus.yml 定义的 job里

{instance=“server01"}
instance/$instance_name
这里是 第二个标签 推送后 显示的 机器名是什么

其实 pushgateway的脚本 并不是很难写

但是通过这样的脚本编程方式

我们可以很快速的 自定义 我们需要的任何 监控数据(Linux 命令方式)

最后 这个我们编写的监控bash脚本 是一次性执行的 bash script.sh
我们需要按时间段 反复执行

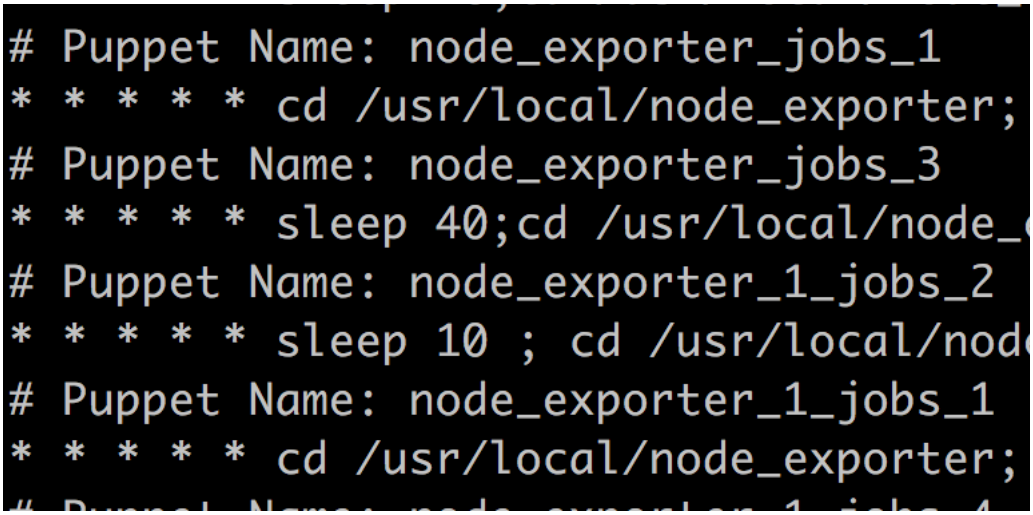
所以呢？ 自然就得结合 contab了

这里顺带提一句

crontab 默认 只能最短一分钟的间隔
如果希望 小于一分钟的间隔 15s

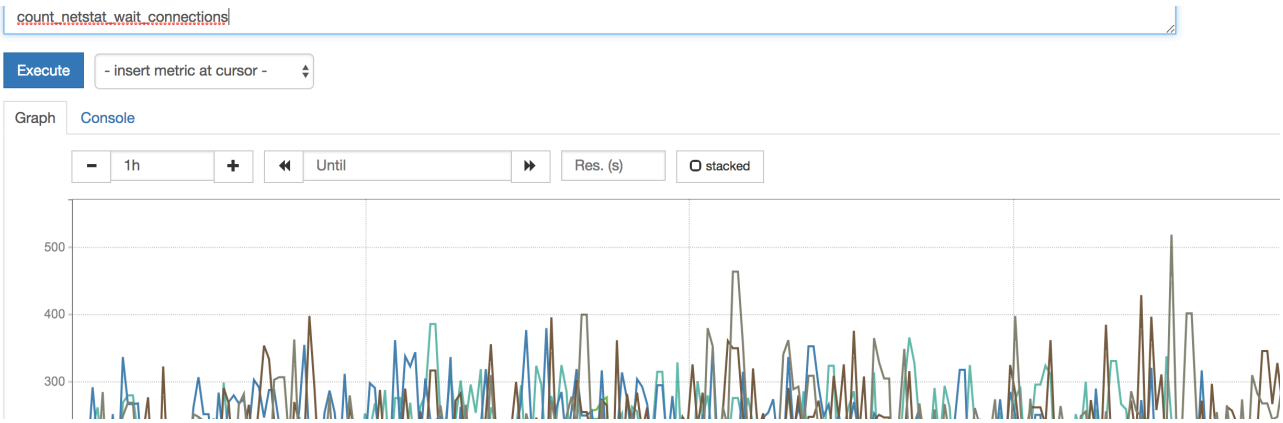
我们使用如下的方法

sleep 10
sleep 20
😀



之后 我们回到prometheus主界面 尝试输入 由我们自己定义的 new_key
看看结果

key的名字 就是这个 label="count_netstat_wait_connections"
定一个新的 key



嘿嘿 很愉快 对不?

其他种类的监控数据 我们都可以通过类似的形式 直接写脚本发送
使用python 也是很好的方式
不过我个人还是推荐 bash shell 因为速度超级的快~ （用python的话 还得include 很多库 大米老师有点懒 哈哈😄）

（四） 使用pushgateway的优缺点

大米之前就跟大家说过 pushgateway这种自定义的 采集方式 非常的快速 而且极其灵活 几乎不收到任何约束

其实我个人 还是非常希望 使用pushgateway来获取监控数据的

各类的exporters虽然玲琅满目 而且默认提供的数据很多了已经

一般情况下 我在企业中 只安装 node_exporter 和 DB_exporter 两个

其他种类的 监控数据 我倾向于 全部使用pushgateway的方式采集 （要的就是快速~ 灵活~）

不过言归正传 习惯并不代表正确性

pushgateway虽然灵活 但是 也是存在一些 短板的
接下来 我们来看看 prometheus官网是怎么说的

WHEN TO USE THE PUSHGATEWAY

The Pushgateway is an intermediary service which allows you to push metrics from jobs which cannot be scraped. For details, see Pushing metrics.

Should I be using the Pushgateway?

We only recommend using the Pushgateway in certain limited cases. There are several pitfalls when blindly using the Pushgateway instead of Prometheus's usual pull model for general metrics collection:

- When monitoring multiple instances through a single Pushgateway, the Pushgateway becomes both a single point of failure and a potential bottleneck.
- You lose Prometheus's automatic instance health monitoring via the `up` metric (generated on every scrape).
- The Pushgateway never forgets series pushed to it and will expose them to Prometheus forever unless those series are manually deleted via the Pushgateway's API.

The latter point is especially relevant when multiple instances of a job differentiate their metrics in the Pushgateway via an `instance` label or similar. Metrics for an instance will then remain in the Pushgateway even if the originating instance is

renamed or removed. This is because the lifecycle of the Pushgateway as a metrics cache is fundamentally separate from the lifecycle of the processes that push metrics to it. Contrast this to Prometheus's usual pull-style monitoring: when an instance disappears (intentional or not), its metrics will automatically disappear along with it. When using the Pushgateway, this is not the case, and you would now have to delete any stale metrics manually or automate this lifecycle synchronization yourself.
Usually, the only valid use case for the Pushgateway is for capturing the outcome of a service-level batch job. A "service-level" batch job is one which is not semantically related to a specific machine or job instance (for example, a batch job that deletes a number of users for an entire service). Such a job's metrics should not include a machine or instance label to decouple the lifecycle of specific machines or instances from the pushed metrics. This decreases the burden for managing stale metrics in the Pushgateway. See also the best practices for monitoring batch jobs.

Alternative strategies

If an inbound firewall or NAT is preventing you from pulling metrics from targets, consider moving the Prometheus server behind the network barrier as well. We generally recommend running Prometheus servers on the same network as the monitored instances.
For batch jobs that are related to a machine (such as automatic security update cronjobs or configuration management client runs), expose the resulting metrics using the Node Exporter's textfile module instead of the Pushgateway.

一大堆的英文 其实总结起来 基本上两点

1） pushgateway 会形成一个单点瓶颈，假如好多个 脚本同时发送给 一个pushgateway的进程
如果这个进程没了，那么监控数据也就没了

2） pushgateway 并不能对发送过来的 脚本采集数据 进行更智能的判断 假如脚本中间采集出问题了
那么有问题的数据 pushgateway一样照单全收 发送给 prometheus

虽然有这么两个所谓的缺点

但是实际上通过我 2年多的使用
对于第一条缺点 其实只要服务器不当机 那么基本上 pushgateway运行还是很稳定的
就算有太多的脚本 同时都发送给一个pushgateway 其实也只是接收的速度会变慢 丢数据没有遇到过 （有的时候 我甚至觉得比exporters还稳定 exporters倒是 有时候会真的就当机 或者 丢失数据 因为exporters开发比较复杂 越简单的东西 往往越稳定）

对于第二条缺点
其实只要我们在写脚本的时候 细心一些 别出错（这也是为什么 我推荐就用 bash 写 因为简单不容易出错， python我可不敢打包票）
那么 监控数据也不会有错误

所以说呢 相信大米 我们就放心的使用pushgateway来尽情采集我们的数据吧

课堂的结尾 给大家留一个作业吧（学习需要勤快）

我们自己搭建pushgateway 并自行编写一个 采集脚本 （bash shell即可）
采集一个 ping 延迟和丢包率吧 ^_^

大米運維課堂