最前沿開源監控 Prometheus專題講座

第九讲：企业级监控数据采集方法

第九讲内容如下

1) prometheus 服务端的安装和后台稳定运行
2) prometheus 服务端配置文件写法
3) node_exporter 安装和后台运行
4) node_exporter 观察 和 采集数据
5) prometheus 查询采集回来的各种数据
6) 使用我们之前的学过的 prometheus 命令行的形式 练习组合各种监控图

（一） **prometheus** 服务端的安装和后台稳定运行

prometheus 的下载地址

https://prometheus.io/download/#prometheus

下载和安装的过程 相对非常的简单

prometheus-2.2.0-rc.0.linux-amd64.tar.gz 最新版 prometheus_server下载

[root@server04 down]# tar -xvzf prometheus-2.2.0-rc.0.linux-amd64.tar.gz
cp -r prometheus-2.2.0-rc.0.linux-amd64 /usr/local/

之后解压缩后 即可直接运行在默认的9090端口 直接就可以使用了

```
[root@server04 prometheus-2.2.0-rc.0.linux-amd64]# ./prometheus
level=info ts=2018-02-17T14:40:43.998504055Z caller=main.go:225 msg="Starting Prometheus" version="(version=2.2.0-rc.0, branch=HEAD, revision=1fe05d40e4b2f4f7479048b1cc3c42865eb73bab)"
level=info ts=2018-02-17T14:40:43.998579894Z caller=main.go:226 build_context="(go=go1.9.2, user=root@f7abb25edc70, date=20180213-11:40:47)"
level=info ts=2018-02-17T14:40:43.998601889Z caller=main.go:227 host_details="(Linux 2.6.32-431.el6.x86_64 #1 SMP Sun Nov 10 22:19:54 EST 2013 x86_64 server04.example.com (none))"
level=info ts=2018-02-17T14:40:43.998619728Z caller=main.go:228 fd_limits="(soft=65535, hard=65535)"
level=info ts=2018-02-17T14:40:44.003481175Z caller=main.go:502 msg="Starting TSDB ..."
level=info ts=2018-02-17T14:40:44.016387501Z caller=web.go:383 component=web msg="Start listening for connections" address=0.0.0.0:9090
level=info ts=2018-02-17T14:40:44.017370287Z caller=main.go:512 msg="TSDB started"
level=info ts=2018-02-17T14:40:44.017392597Z caller=main.go:588 msg="Loading configuration file" filename=prometheus.yml
level=info ts=2018-02-17T14:40:44.018308542Z caller=main.go:489 msg="Server is ready to receive web requests."
```

不过 我们作为合格的运维工程师，对生产环境上 运行的软件需要多考虑一些更合理的运行模式
接下来 咱们来一起看下

1） 我们需要让prometheus_server 运行在后台 而不是前端（客户端退出后 prometheus也退出了 这个很好理解）

相对比较好的运行后台模式 这里给大家推荐两种方法
& Nohup

第一种： 安装**screen**工具 放入后台运行

```
[root@server04 ~]# screen  -ls
No Sockets found in /var/run/screen/S-root.

[root@server04 ~]#
[root@server04 ~]#
[root@server04 ~]# screen
```

```
[root@server04 ~]# cd /usr/local/prometheus-2.2.0-rc.0.linux-amd64/
[root@server04 prometheus-2.2.0-rc.0.linux-amd64]#
[root@server04 prometheus-2.2.0-rc.0.linux-amd64]# ./prometheus
level=info ts=2018-02-17T14:43:13.525492477Z caller=main.go:225 msg="Starting Prometheus" version="(version=2.2.0-rc.0, branch=HEAD, revision=1
fe05d40e4b2f4f7479048b1cc3c42865eb73bab)"
level=info ts=2018-02-17T14:43:13.52554881Z caller=main.go:226 build_context="(go=go1.9.2, user=root@f7abb25edc70, date=20180213-11:40:47)"
level=info ts=2018-02-17T14:43:13.525564374Z caller=main.go:227 host_details="(Linux 2.6.32-431.el6.x86_64 #1 SMP Sun Nov 10 22:19:54 EST 2013
x86_64 server04.example.com (none))"
level=info ts=2018-02-17T14:43:13.52557581Z caller=main.go:228 fd_limits="(soft=65535, hard=65535)"
level=info ts=2018-02-17T14:43:13.528256563Z caller=main.go:502 msg="Starting TSDB ..."
level=info ts=2018-02-17T14:43:13.541192942Z caller=web.go:383 component=web msg="Start listening for connections" address=0.0.0.0:9090
level=info ts=2018-02-17T14:43:13.54876269Z caller=main.go:512 msg="TSDB started"
level=info ts=2018-02-17T14:43:13.548805771Z caller=main.go:588 msg="Loading configuration file" filename=prometheus.yml
level=info ts=2018-02-17T14:43:13.549712547Z caller=main.go:489 msg="Server is ready to receive web requests."
```

```
[root@server04 ~]# screen  -ls
There is a screen on:
        5796.pts-1.server04     (Detached)
1 Socket in /var/run/screen/S-root.
```

这种screen放入后台运行的方式 最简单快速了
最适合懒人使用....比如我 哈哈😄

screen还有另外一个好处 就是 可以随时切换进入 程序前台窗口 查看各种调试信息

screen 也有不好的地方

- 不够正规化 总觉得还是个临时办法
- screen -l 提供的后台 列表 不够人性化，很多时候 你记不住 到底哪个是哪个

- 很容易被误关闭 操作的时候 ctrl +ad / ctrl +d 不小心操作错了 直接就退出去了..

第二种： 使用daemonize 放入后台方式

daemonize Unix系统后台守护进程管理软件

优点：更加正规 后台运行更稳定

```
git clone git://github.com/bmc/daemonize.git
```

```
sh configure && make && sudo make install
```

daemonize -c /data/prometheus/ **/data/prometheus/up.sh**

-c 是指定运行路径

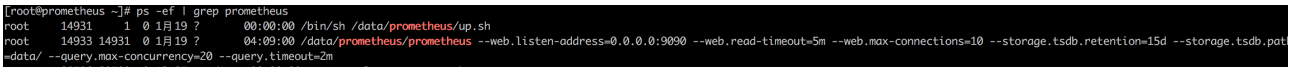
/data/prometheus/up.sh 是运行路径下的 一个启动脚本

下面是这个启动脚本的内容

内容：就是开启prometheus进程

[root@prometheus yd]# cat /data/prometheus/up.sh
/data/prometheus/prometheus --web.listen-address="0.0.0.0:9090" --web.read-timeout=5m --web.max-connections=10 --storage.tsdb.retention=15d --storage.tsdb.path="data/" --query.max-concurrency=20 --query.timeout=2m

然后 我们来看下 ./prometheus 在实际企业运行时 启动参数的合理配置



--web.read-timeout=5m Maximum duration before timing out read of the request, and closing idle connections.
请求链接的最大等待时间 prometheus process -> GET PUSH
防止 太多的空闲链接 占用资源

--web.max-connections=512 Maximum number of simultaneous connections.
最大链接数

--storage.tsdb.retention=15d
How long to retain samples in the storage.
prometheus开始采集监控数据后 会存在内存中和硬盘中
对于保留期限的设置 很重要 太长的话 硬盘和内存都吃不消 / 太短的话 要查历史数据就没有了
企业中设置 15天为宜

_

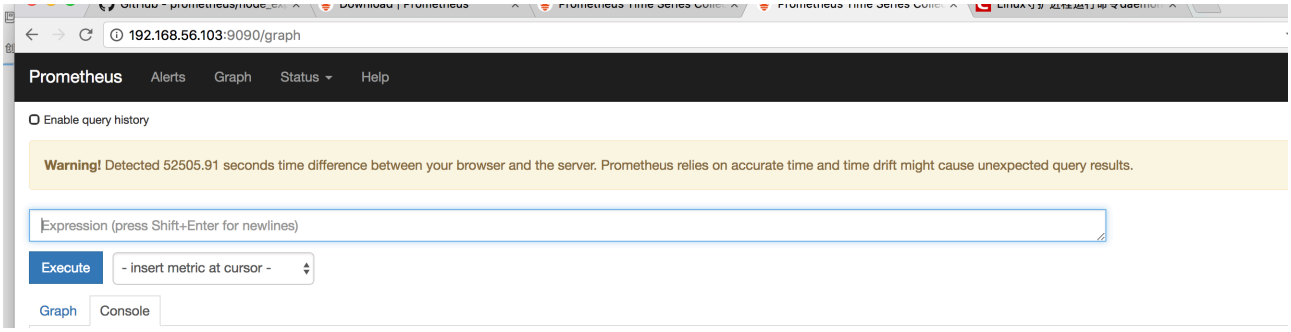
--storage.tsdb.path="data/"
Base path for metrics storage.
存储数据路径 这个也很重要 不要随便放在一个地方就执行 会把/ 根目录塞满了

--query.timeout=2m Maximum time a query may take before being aborted.

--query.max-concurrency=20 Maximum number of queries executed concurrently.

上面这两项 是对 用户执行prometheus 查询时候的 优化设置 防止太多的用户同时查询，也防止单个用户执行过大的查询而一直不退出

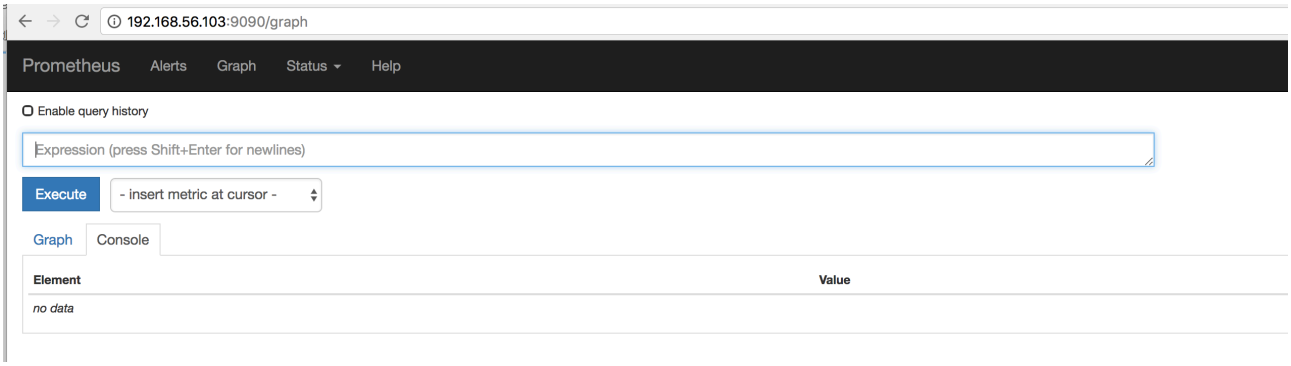
如上这几项参数 配置上去后 prometheus运行 就相对稳妥多了 😀



web上直接输入 ip:port 就可以进入首页

这里遇到一个问题

就是 prometheus 对系统时间 非常敏感 一定要时时刻刻 保证系统时间同步 不然 曲线是乱的
ntpdate 循环同步时间后 错误提示就没有了



prometheus 运行时 存放的历史数据 在这儿

其中 这些 长串字母的 是历史数据保留

而 当前近期数据 实际上保留在内存中

并且 按照一定间隔 存放在 wal / 目录中 防止突然断电 或者 重启 以用来 恢复内存中的数据 （这个咱们之前也提到过）

（二） prometheus 服务端配置文件写法

prometheus_server 安装稳妥之后 咱们要来看下配置文件

/data/prometheus/prometheus.yml

配置文件是 运行在哪个目录 就默认读取 哪个目录下 prometheus.yml文件
除非使用参数指定其他位置的 配置文件

配置文件中 我们主要关注 这几个地方

```
global:
  scrape_interval:     15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.


scrape_configs:
# The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: 'prometheus'

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ['prometheus.server:9090','prometheus.server:9100']

  - job_name: 'pushgateway'

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

  - job_name: 'aliyun'

    static_configs:
      - targets:
['server4:9100','web3:9100','server6:9100','sesrver7:9100','web8:9100','log1:9100','mysql1:9100']
```

一个就是 全局变量 scrape_interval 设置多少时间间隔 采集一次数据

另一个 就是 job和targets的写法 配置一个job的标签，然后 在这个标签下 定义 我们需要监控的机器

targets: [‘server4:9100’

hostname + 端口

而这里的端口 实际上 就是 node_exporter的默认运行端口

hostname server01 prometues => dns

/etc/hosts

local dns server

（三） node_exporter 安装和后台运行

node_exporter的下载和安装 跟prometheus也是一样的
这里就不再演示了

下载地址为
https://prometheus.io/download/#node_exporter

同样适用 daemonize 放入被监控服务器后台运行

[root@web7 ~]# ps -ef | grep node_exporter

root   7400   1 0 2017 ?   00:00:00 /bin/sh /usr/local/node_exporter/up.sh

root 7402 7400 0 2017 ? 01:34:41 /usr/local/node_exporter/node_exporter
默认运行在 9100端口

（四） node_exporter 观察 和 采集数据

运行在后台以后

我们需要 针对这个 node_exporter 进行初步的手动查询 以确保 正常获取监控数据

用如下方法 本地查询
[root@web7 ~]# curl localhost:9100/metrics

-node_cpu{cpu="cpu7",mode="steal"} 0
node_cpu{cpu="cpu7",mode="system"} 131503.32
-node_cpu{cpu="cpu7",mode="user"} 397798.53

prometheus 15s GET请求

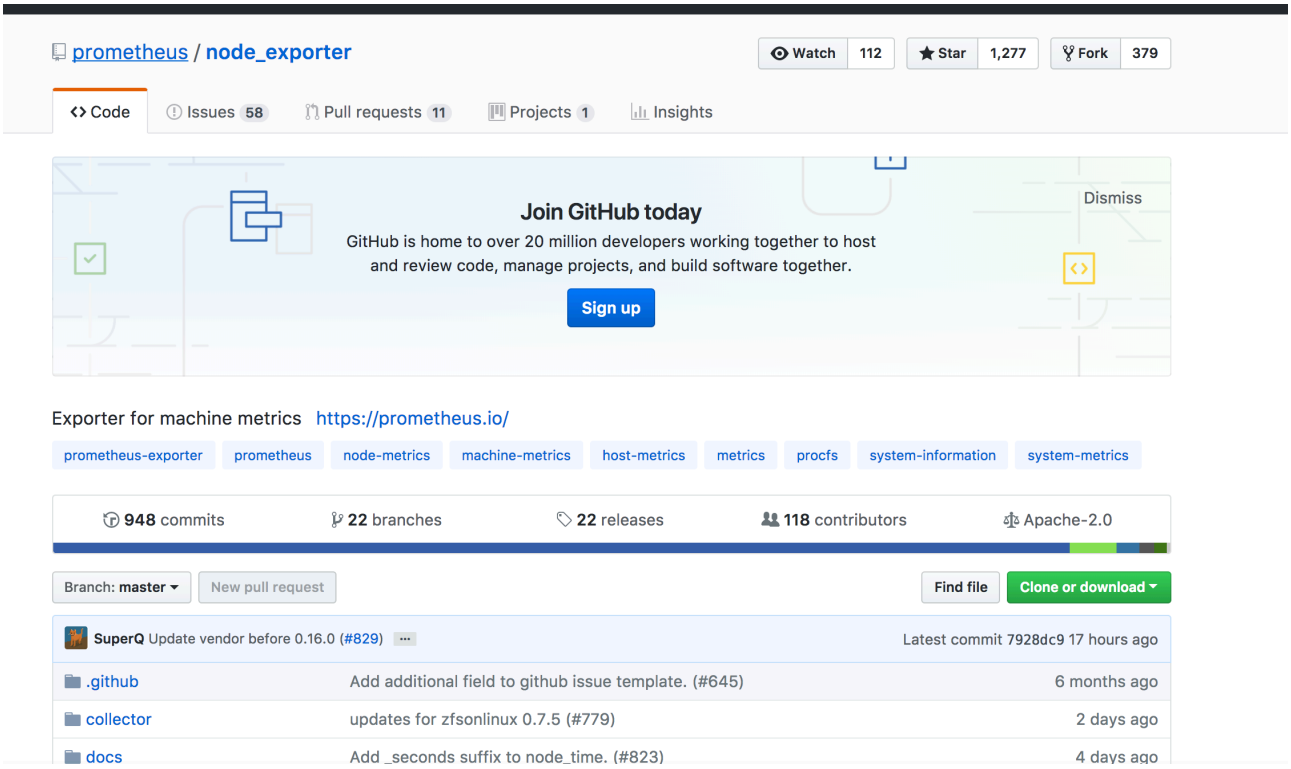
只有挑几个重要的 key 看到有数据采集上来 就OK了
prometheus 命令行

node_exporter 经过实际测试 在 Centos/Redhat 任意版本系统中均可以顺利运行 （ubantu debian ）
还没有遇到过 什么问题

给node_exporter的开发者 点个赞 👌

然后 我们去到 node_exporter在 github上的地址
来看看 我们伟大的社区开发者们 都给咱们提供了 哪些有用的采集项目

https://github.com/prometheus/node_exporter



尽然在 github上 那就很自然了 咱们的node_exporter 也一样是开源项目
本身使用 go语言发开

后面 提供了 node_exporter 默认开启 和 不开启的 监控项目

可以监控的内容非常的多 庞大
感兴趣的朋友 可以自行 满满挖掘

Enabled by default

Name	Description	OS
arp	Exposes ARP statistics from `/proc/net/arp`.	Linux
bcache	Exposes bcache statistics from `/sys/fs/bcache/`.	Linux
conntrack	Shows conntrack statistics (does nothing if no `/proc/sys/net/netfilter/` present).	Linux
cpu	Exposes CPU statistics	Darwin, Dragonfly, FreeBSD, Linux
diskstats	Exposes disk I/O statistics.	Darwin, Linux
edac	Exposes error detection and correction statistics.	Linux
entropy	Exposes available entropy.	Linux
exec	Exposes execution statistics.	Dragonfly, FreeBSD
filefd	Exposes file descriptor statistics from `/proc/sys/fs/file-nr`.	

Linux
filesystem
Exposes filesystem statistics, such as disk space used.
Darwin, Dragonfly, FreeBSD, Linux, OpenBSD
hwmon
Expose hardware monitoring and sensor data from `/sys/class/hwmon/`.
Linux
infiniband
Exposes network statistics specific to InfiniBand and Intel OmniPath configurations.
Linux
ipvs
Exposes IPVS status from `/proc/net/ip_vs` and stats from `/proc/net/ip_vs_stats`.
Linux
loadavg
Exposes load average.
Darwin, Dragonfly, FreeBSD, Linux, NetBSD, OpenBSD, Solaris
mdadm
Exposes statistics about devices in `/proc/mdstat` (does nothing if no `/proc/mdstat` present).
Linux
meminfo
Exposes memory statistics.
Darwin, Dragonfly, FreeBSD, Linux, OpenBSD
netdev
Exposes network interface statistics such as bytes transferred.
Darwin, Dragonfly, FreeBSD, Linux, OpenBSD
netstat

Exposes network statistics from `/proc/net/netstat`. This is the same information as `netstat -s`.
Linux
nfs
Exposes NFS client statistics from `/proc/net/rpc/nfs`. This is the same information as `nfsstat -c`.
Linux
nfsd
Exposes NFS kernel server statistics from `/proc/net/rpc/nfsd`. This is the same information as `nfsstat -s`.
Linux
sockstat
Exposes various statistics from `/proc/net/sockstat`.
Linux
stat
Exposes various statistics from `/proc/stat`. This includes boot time, forks and interrupts.
Linux
textfile
Exposes statistics read from local disk. The `--collector.textfile.directory` flag must be set.
any
time
Exposes the current system time.
any
timex
Exposes selected adjtimex(2) system call stats.
Linux
uname
Exposes system information as provided by the uname system call.
Linux

vmstat
Exposes statistics from `/proc/vmstat`.
Linux
wifi
Exposes WiFi device and station statistics.
Linux
xfs
Exposes XFS runtime statistics.
Linux (kernel 4.4+)
zfs
Exposes ZFS performance statistics.
Linux
Disabled by default

Name
Description
OS
bonding
Exposes the number of configured and active slaves of Linux bonding interfaces.
Linux
buddyinfo
Exposes statistics of memory fragments as reported by /proc/buddyinfo.
Linux
devstat
Exposes device statistics
Dragonfly, FreeBSD
drbd
Exposes Distributed Replicated Block Device statistics (to version 8.4)
Linux
interrupts

Exposes detailed interrupts statistics.
Linux, OpenBSD
ksmd
Exposes kernel and system statistics from `/sys/kernel/mm/ksm`.
Linux
logind
Exposes session counts from logind.
Linux
meminfo_numa
Exposes memory statistics from `/proc/meminfo_numa`.
Linux
mountstats
Exposes filesystem statistics from `/proc/self/mountstats`. Exposes detailed NFS client statistics.
Linux
ntp
Exposes local NTP daemon health to check time
any
qdisc
Exposes queuing discipline statistics
Linux
runit
Exposes service status from runit.
any
supervisord
Exposes service status from supervisord.
any
systemd
Exposes service and system status from systemd.
Linux
tcpstat

Exposes TCP connection status information from `/proc/net/tcp` and `/proc/net/tcp6`. (Warning: the current version has potential performance issues in high load situations.)

另外 这些项目 可以通过 node_exporter的自定义 启动参数 来实现 开启

./node_exporter —help

--collector.arp Enable the arp collector (default: enabled).
--collector.bcache Enable the bcache collector (default: enabled).
--collector.bonding Enable the bonding collector (default: disabled).
--collector.buddyinfo Enable the buddyinfo collector (default: disabled).
--collector.conntrack Enable the conntrack collector (default: enabled).
--collector.cpu Enable the cpu collector (default: enabled).
--collector.diskstats Enable the diskstats collector (default: enabled).
--collector.drbd Enable the drbd collector (default: disabled).
--collector.edac Enable the edac collector (default: enabled).
--collector.entropy Enable the entropy collector (default: enabled).

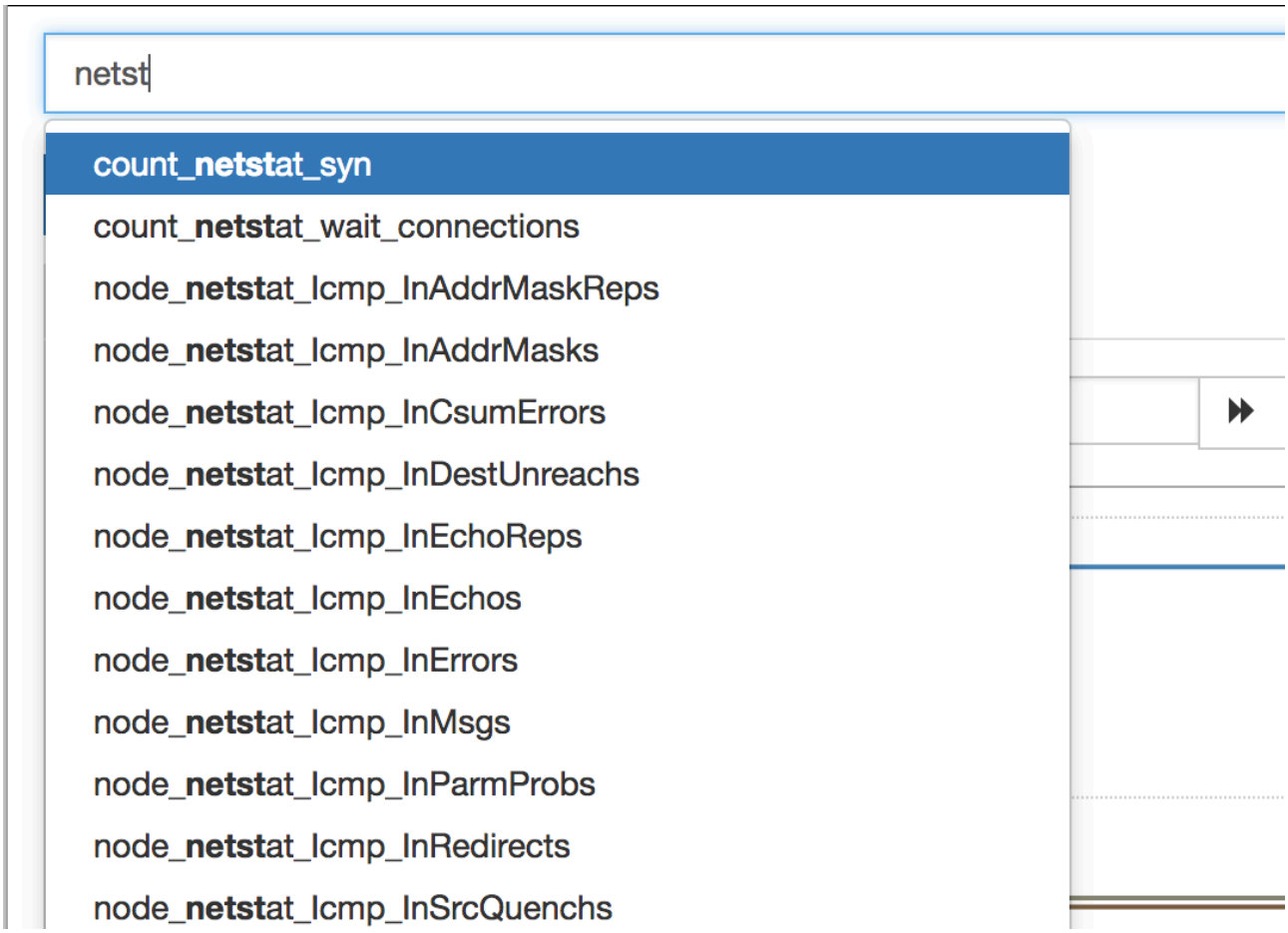
--collector.filefd Enable the filefd collector (default: enabled).
--collector.filesystem Enable the filesystem collector (default: enabled).
--collector.gmond Enable the gmond collector (default: disabled).
--collector.hwmon Enable the hwmon collector (default: enabled).
--collector.infiniband Enable the infiniband collector (default: enabled).
--collector.interrupts Enable the interrupts collector (default: disabled).
--collector.ipvs Enable the ipvs collector (default: enabled).
--collector.ksmd Enable the ksmd collector (default: disabled).
--collector.loadavg Enable the loadavg collector (default: enabled).
--collector.logind Enable the logind collector (default: disabled).
--collector.mdadm Enable the mdadm collector (default: enabled).

（五） prometheus 查询采集回来的各种数据

接下来 我们回到 prometheus的主界面
验证一下 我们新部署的 监控机器上的node_exporter 是否给我们 正确返回了 数据

随便挑几个 key 就可以查看

另外 prometheus 的命令行 本身也支持suggest 功能（输入提示）



随便找个key 查询一下 是否有输出图输出 就可以了

本身node_exporter提供的 keys 实在太多了 （因为 都是从Linux系统中的 底层 各种挖掘数据回来）

我们没有时间 也没有必要 把每一个key 都掌握 只要知道 一部分重要的 必须的key 就足够了

（六） 使用我们之前的学过的 prometheus 命令行的形式 练习组合各种监控图

接下来 咱们找一个 比较重要的key 然后 用我们学过的 命令行方式 给他组成一个 临时监控图

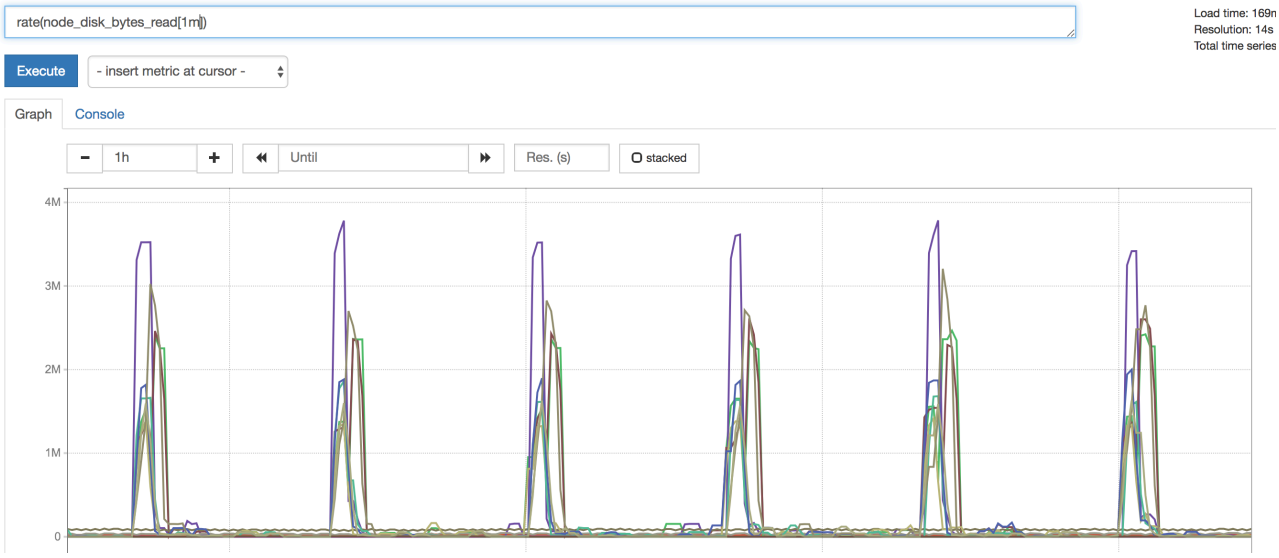
比如
node_cpu

node_memory

node_disk

等等 这里我们不再做演示了

可以自行找几个类似的 keys 组成监控图



不过这里有一个问题..

命令行组成监控图之后 这些图都是临时的 如果我关闭了浏览器 那么下次再想看的时候就没有了 又要重新输入查询语句（麻烦不）
而且prometheus 自带的这个 监控图 实际上 并不美观