



# 大米運維課堂

## 最前沿開源監控 Prometheus 專題講座

### 第六讲 Prometheus 初探和配置（安装测试）

#### 第六讲内容

- Prometheus 官网下载
- Prometheus 开始安装
- Prometheus 启动运行
- Prometheus 基本配置文件讲解
- 安装第一个exporter =» node\_exporter
- Prometheus 连接exporter获取数据
- Prometheus 命令行入门 第一个数学查询公式

之前我们说过，相对于较难的使用方法，prometheus其实安装的过程 反而是异常的简单

安装Prometheus之前 我们必须先安装 ntp时间同步

(prometheus T\_S 对系统时间的准确性要求很高，必须保证本机时间实时同步)

以Centos7 为例

```
timedatectl set-timezone Asia/Shanghai
```

- \* \* \* \* \* ntpdate -u [cn.pool.ntp.org](http://cn.pool.ntp.org)

### 1) Prometheus 下载

首先 我们去到<http://prometheus.io> 官网

下载最新版本 [prometheus-2.1.0.linux-amd64.tar.gz](https://github.com/prometheus/prometheus/releases/download/v2.1.0/prometheus-2.1.0.linux-amd64.tar.gz)

wget <https://github.com/prometheus/prometheus/releases/download/v2.1.0/prometheus-2.1.0.linux-amd64.tar.gz>

### 2) Prometheus的安装 非常简单

```
[root@server01 download]# tar -xvzf prometheus-2.0.0.linux-amd64.tar.gz
prometheus-2.0.0.linux-amd64/
prometheus-2.0.0.linux-amd64/conssoles/
prometheus-2.0.0.linux-amd64/conssoles/index.html.example
prometheus-2.0.0.linux-amd64/conssoles/node-cpu.html
prometheus-2.0.0.linux-amd64/conssoles/node-disk.html
prometheus-2.0.0.linux-amd64/conssoles/node-overview.html
prometheus-2.0.0.linux-amd64/conssoles/node.html
prometheus-2.0.0.linux-amd64/conssoles/prometheus-overview.html
prometheus-2.0.0.linux-amd64/conssoles/prometheus.html
prometheus-2.0.0.linux-amd64/console_libraries/
prometheus-2.0.0.linux-amd64/console_libraries/menu.lib
prometheus-2.0.0.linux-amd64/console_libraries/prom.lib
prometheus-2.0.0.linux-amd64/prometheus.yml
prometheus-2.0.0.linux-amd64/LICENSE
prometheus-2.0.0.linux-amd64/NOTICE
prometheus-2.0.0.linux-amd64/prometheus
prometheus-2.0.0.linux-amd64/promtool
```

```
cp -rf prometheus-2.0.0.linux-amd64 /usr/local/prometheus
```

### 3) Prometheus 启动 和 后台运行

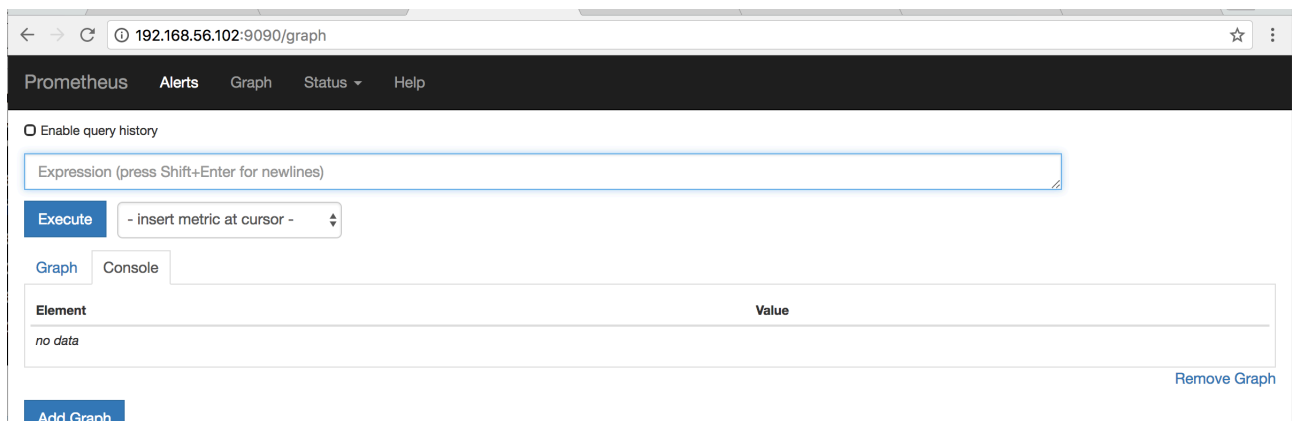
启动也很简单

./prometheus

```
[root@server02 prometheus]# ./prometheus
level=info ts=2018-02-05T03:20:02.342855977Z caller=main.go:225 msg="Starting Prometheus" version="(version=2.1.0, branch=HEAD, revision=85f23d82a045d103ea7f3c89a91fba4a93e6367a)"
level=info ts=2018-02-05T03:20:02.342914528Z caller=main.go:226 build_context="(go=go1.9.2, user=root@6e784304d3ff, date=20180119-12:01:23)"
level=info ts=2018-02-05T03:20:02.342929417Z caller=main.go:227 host_details="(Linux 3.10.0-514.el7.x86_64 #1 SMP Tue Nov 22 16:42:41 UTC 2016 x86_64 server02.example.com (none))"
level=info ts=2018-02-05T03:20:02.342941275Z caller=main.go:228 fd_limits="(soft=1024, hard=4096)"
level=info ts=2018-02-05T03:20:02.3481105Z caller=main.go:499 msg="Starting TSDB ..."
level=info ts=2018-02-05T03:20:02.405710855Z caller=web.go:383 component=web msg="Start listening for connections" address=0.0.0.0:9090
level=info ts=2018-02-05T03:20:02.447091187Z caller=main.go:509 msg="TSDB started"
level=info ts=2018-02-05T03:20:02.447147085Z caller=main.go:585 msg="Loading configuration file" filename=prometheus.yml
level=info ts=2018-02-05T03:20:02.448037847Z caller=main.go:486 msg="Server is ready to receive web requests."
level=info ts=2018-02-05T03:20:02.448168059Z caller=manager.go:59 component="scrape manager" msg="Starting scrape manager..."
```

之后默认运行在 9090

浏览器可以直接打开访问 无账号密码验证（如果希望加上验证，可以使用类似apache httpd 方式添加）



4) 接下来 我们来简单看一下 Prometheus的主配置文件

其实prometheus解压安装之后，就默认自带了一个 基本的配置文件如下  
./prometheus --yaml

```
[root@server02 ~]# cd /usr/local/prometheus/
[root@server02 prometheus]# ls
console_libraries  consoles  data  LICENSE  NOTICE  prometheus  prometheus.yml  promtool
[root@server02 prometheus]# cat prometheus.yml
# my global config
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_timeout is set to the global default (10s).

# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
        - targets:
            # - alertmanager:9093

# Load rules once and periodically evaluate them according to the global 'evaluation_interval'.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:
# Here it's Prometheus itself.
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: ['localhost:9090']
```

我们来大致讲解一下 配置文件的内容

```
# my global config
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_timeout is set to the global default (10s).
```

前两个全局变量

scrape\_interval. 抓取采样数据的 时间间隔， 默认 每15秒去被监控机上 采样一次 => 5s

这个就是我们所说的 prometheus的自定义 数据采集频率了

evaluation\_interval. 监控数据规则的评估频率 grafana

这个参数是prometheus多长时间 会进行一次 监控规则的评估

举个例：假如 我们设置 当 内存使用量 > 70%时 发出报警 这么一条rule（规则）  
那么prometheus 会默认 每15秒来执行一次这个规则 检查内存的情况

```
# Alertmanager configuration
alerting:
  alertmanagers:
    - static_configs:
      - targets:
        # - alertmanager:9093
```

Alertmanager 是prometheus的一个用于管理和发出报警的 插件

我们这里对 Alertmanger 暂时先不做介绍 暂时也不需要（我们采用 4.0最新版的 Grafana , 本身就已经支持报警发出功能了 往后我们会学习到）

再往后 从这里开始 进入**prometheus**重要的 配置采集节点的设置

**# Here it's Prometheus itself.**

```
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: ['localhost:9090']
```

先定义一个 job的名称

```
- job_name: 'prometheus'
```

然后 定义监控节点 targets

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

static_configs:
  - targets: ['localhost:9090']
```

- targets的设定

以这种形式设定 默认带了一个 prometheus本机的

static\_configs:

- targets: ['localhost:9090']

这里可以继续 扩展加入 其他需要被监控的节点

如下是一个 生产配置例子

- job\_name: 'aliyun'

static\_configs:

- targets: ['server04:9100','IP:9100','nginx06:9100','web7:9100','redis1:9100','log:9100','redis2:9100']

prometheusserver \_ /etc/hosts, local\_dns server

可以看到 targets可以并列写入 多个节点

用逗号隔开， 机器名+端口号

端口号：通常用的就是 exporters 的端口

在这里 9100 其实是 node\_exporter 的默认端口

如此 prometheus就可以通过配置文件 识别监控的节点，持续开始采集数据

prometheus到此就算初步的搭建好了（后续课程 还有各种扩展的 优化）

5) 光搭建好prometheus\_server 是不够的，我们需要给监控节点 搭建第一个exporter 用来采样数据

我们就选用 企业中最常用的 node\_exporter 这个插件

node\_exporter 之前介绍过，是一个以http\_server方式运行在后台，并且持续不断采集 Linux 系统中 各种操作系统本身相关的监控参数的程序

其采集量是很大很全的，往往默认的采集项目 就远超过你的实际需求

接下来我们来看下 node\_exporter是怎么回事

一样 先下载node\_exporter 从官网

[https://prometheus.io/download/#node\\_exporter](https://prometheus.io/download/#node_exporter)

下载之后 解压缩 然后直接运行即可

```
[root@log node_exporter]# ./node_exporter
INFO[0000] Starting node_exporter (version=0.15.2, branch=HEAD, revision=98bc64930d34878b84a0f87dfe6e1a6da61e532d) source="node_exporter.go:43"
INFO[0000] Build context (go=go1.9.2, user=root@d5c4792c921f, date=20171205-14:50:53) source="node_exporter.go:44"
INFO[0000] No directory specified, see --collector.textfile.directory source="textfile.go:57"
INFO[0000] Enabled collectors: source="node_exporter.go:50"
INFO[0000] - time source="node_exporter.go:52"
INFO[0000] - ipvs source="node_exporter.go:52"
INFO[0000] - conntrack source="node_exporter.go:52"
INFO[0000] - netdev source="node_exporter.go:52"
INFO[0000] - diskstats source="node_exporter.go:52"
INFO[0000] - entropy source="node_exporter.go:52"
INFO[0000] - timex source="node_exporter.go:52"
INFO[0000] - xfs source="node_exporter.go:52"
INFO[0000] - loadavg source="node_exporter.go:52"
INFO[0000] - stat source="node_exporter.go:52"
INFO[0000] - uname source="node_exporter.go:52"
INFO[0000] - edac source="node_exporter.go:52"
INFO[0000] - textfile source="node_exporter.go:52"
INFO[0000] - meminfo source="node_exporter.go:52"
INFO[0000] - bcache source="node_exporter.go:52"
INFO[0000] - filefd source="node_exporter.go:52"
INFO[0000] - cpu source="node_exporter.go:52"
INFO[0000] - sockstat source="node_exporter.go:52"
INFO[0000] - zfs source="node_exporter.go:52"
INFO[0000] - wifi source="node_exporter.go:52"
INFO[0000] - infiniband source="node_exporter.go:52"
INFO[0000] - hwmon source="node_exporter.go:52"
INFO[0000] - vmstat source="node_exporter.go:52"
INFO[0000] - filesystem source="node_exporter.go:52"
INFO[0000] - netstat source="node_exporter.go:52"
INFO[0000] - mdadm source="node_exporter.go:52"
INFO[0000] - arp source="node_exporter.go:52"
INFO[0000] Listening on :9100 source="node_exporter.go:76"
```

node\_exporter的运行更加简单 如上图所示

运行起来以后 我们使用netstat -tnlp 可以来看下 node\_exporter进程的状态

```
[root@log node_exporter]# netstat -tnlp | grep node
tcp        0      0 :::9100          :::*              LISTEN      8273/node_exporter
[root@log node_exporter]#
```

这里就可以看出 node\_exporter默认工作在9100端口  
可以响应 prometheus\_server发过来的 HTTP\_GET请求

也可以响应其他方式的 HTTP\_GET请求

我们自己就可以发送 测试

```
[root@log node_exporter]# curl localhost:9100/metrics
```



```

node_vmstat_unevictable_pgs_scanned 0
# HELP node_vmstat_unevictable_pgs_stranded /proc/vmstat information field unevictable_pgs_stranded.
# TYPE node_vmstat_unevictable_pgs_stranded untyped
node_vmstat_unevictable_pgs_stranded 0
# HELP node_vmstat_zone_reclaim_failed /proc/vmstat information field zone_reclaim_failed.
# TYPE node_vmstat_zone_reclaim_failed untyped
node_vmstat_zone_reclaim_failed 0
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 5107.68
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 65535
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 11
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.0078592e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.51359570948e+09
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 8.39794688e+08

```

执行curl之后，我们看到 node\_exporter 给我们返回了 大量的这种 metrics类型 K/V数据  
关于 metrics 和 k/v 之前咱们介绍过了 就不再重复了

而这些 返回的 K/V数据， 其中的Key的名称 就可以直接复制黏贴 在prometheus的查询命令  
行 来查看结果了

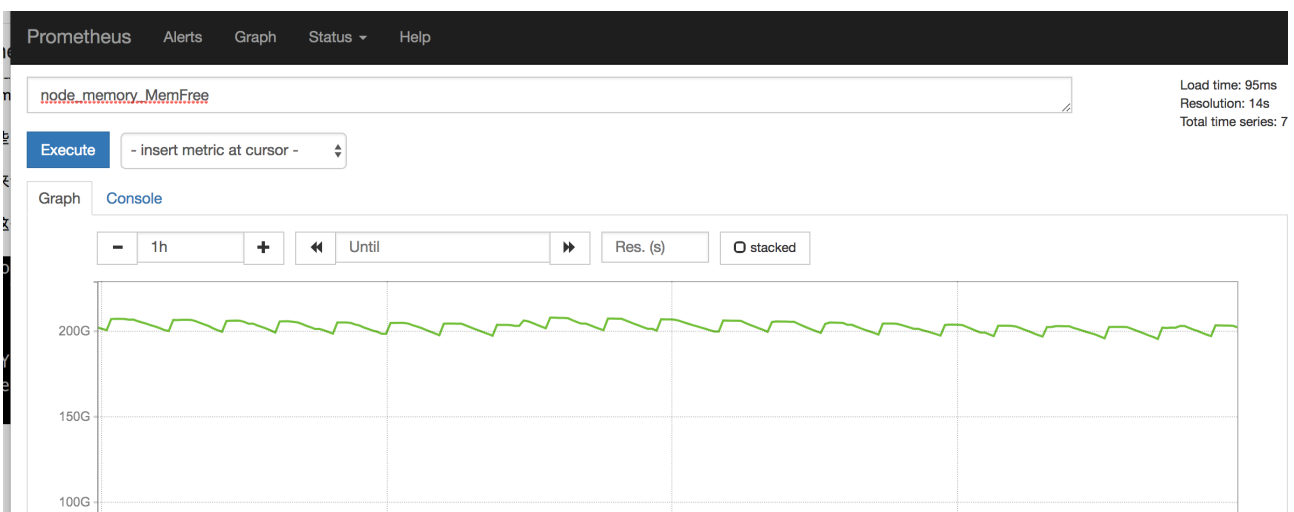
我们来试一试

就用这一项看看 node\_memory\_MemFree

```

[root@log node_exporter]# curl localhost:9100/metrics | grep node_memory_MemFree
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %         0         0             Dload  Upload    Total   Spent    Left   Speed
  0      0    0     0    0     0      0      0  --:--:-- --:--:-- --:--:--          0# HELP node_memory_MemFree gauge
# TYPE node_memory_MemFree gauge
node_memory_MemFree 3.95296768e+08
100 96622 100 96622    0     0 4782k      0  --:--:-- --:--:-- --:--:-- 4966k

```





直接就可以看到曲线了 对吗?

其实这个 就是 最简单的 来查看一下 服务器的 空闲内存状态的方式

觉得很简单对吗? prometheus不也就如此而已吗 没什么难的 我暗中呵呵。。。。 \_-

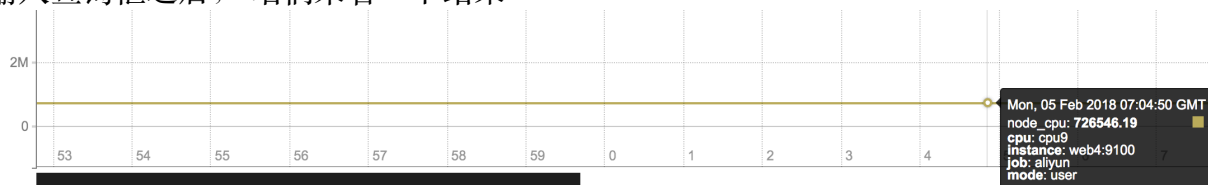
好吧 那么我们接下来 再来看下一个 难一些的例子 **CPU使用率** 的获取方式

node\_cpu这个 key 也是node\_exporter返回的一个 用来统计 CPU使用率的

```
[root@log node_exporter]# curl localhost:9100/metrics | grep -i node_cpu | head
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total   Spent    Left   Speed
100 96610  100 96610    0     0 8474k      0 --:--:-- --:--:-- --:--:-- 9434k
# HELP node_cpu Seconds the cpus spent in each mode.
# TYPE node_cpu counter
node_cpu{cpu="cpu0",mode="guest"} 0
node_cpu{cpu="cpu0",mode="guest_nice"} 0
node_cpu{cpu="cpu0",mode="idle"} 3.77805926e+06
```

node\_cpu|

输入查询框之后, 咱们来看一下结果



疑??? 怎么回事 CPU不是应该是使用率吗? 类似 百分 50% 80%这样的数据才对啊



怎么返回是一个持续不断累加的 近似于直线的 庞大的数字呢?

```
Mon, 05 Feb 2018 07:04:50 GMT
node_cpu: 726546.19
cpu: cpu9
```

说到这儿, 我们就不得不给大家提一句了 这个其实就关系到 prometheus对Linux数据的采集的精细的特性

其实 prometheus对Linux CPU的采集 并不是 直接给我们返回一个 现成的CPU百分比 而是 返回 Linux中 很底层的 cpu时间片 累积数值 的这样一个数据（

我们平时用惯了 top / uptime这种简便的方式 看CPU使用率，往往浅尝辄止 根本没有好好深入理解 所谓的CPU使用率 在linux中到底是怎么回事）

当懒人当的时间长了 自己问问自己对Linux到底了解多少呢？

大米给大家补补课吧：)

其实 如果想真的弄明白CPU的使用率这个概念 在Linux中 要先从CPU时间 这个概念开始建立

Linux中 CPU时间 实际是指：从操作系统开启算起 CPU就开始工作了 并记录自己在工作 中 总共使用的“时间”的累积量 把它保存在系统中

而累积的CPU使用时间 还会分成几个重要的状态类型

比如 CPU time => 分成 CPU user time / sys time / nice time / idle time / irq / 等等。。。

翻译过来就是 CPU 用户态使用时间，系统/内核态使用时间， nice值分配使用时间，空闲时间，中断时间 等等

那么所谓的 CPU使用率 是什么意思呢？

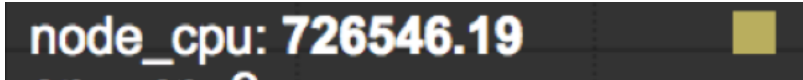
CPU使用率 最准确的定义 其实是 CPU各种状态中 除了idle(空闲) 这个状态外，其他所有的 CPU状态的 加合 / 总CPU时间

得出来的 就是我们所说的 CPU使用率 (运行的任务 用户 内核)

那么 回到我们刚才 使用 node\_cpu 这个key 如果直接输入进去

他返回的 其实是 CPU各个核CPU各个核 各个状态下 从开机开始 一直累积下来的 CPU使用时间的累积值

所以我们才 看到这么一个



```
node_cpu: 726546.19
```

如下 截取一段网上 对各个CPU状态的时间单位的解释

## 参数 解析 (单位: jiffies)

(jiffies是内核中的一个全局变量, 用来记录自系统启动以来产生的节拍数, 在linux中, 一个节拍大致可理解为操作系统进程调度的最小时间片, 不同linux内核可能值有不同, 通常在1ms到10ms之间)

user (38082) 从系统启动开始累计到当前时刻, 处于用户态的运行时间, 不包含 nice值为负进程。

nice (627) 从系统启动开始累计到当前时刻, nice值为负的进程所占用的CPU时间

system (27594) 从系统启动开始累计到当前时刻, 处于核心态的运行时间

idle (893908) 从系统启动开始累计到当前时刻, 除IO等待时间以外的其它等待时间 iowait (12256) 从系统启动开始累计到当前时刻, IO等待时间(since 2.5.41)

irq (581) 从系统启动开始累计到当前时刻, 硬中断时间(since 2.6.0-test4)

softirq (895) 从系统启动开始累计到当前时刻, 软中断时间(since 2.6.0-

test4) stealstolen(0) which is the time spent in other operating systems when running in a virtualized environment(since 2.6.11)

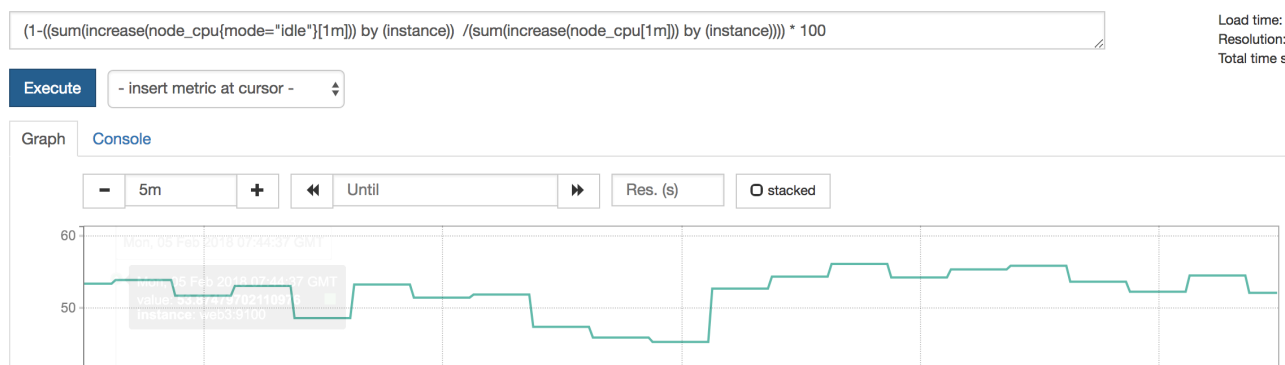
guest(0) which is the time spent running a virtual CPU for guest operating systems under the control of the Linux kernel(since 2.6.24)

所以: 如果在prometheus中 想对CPU的使用率 准确的来查询

正确的方法如下: (难的开始 要来了。。。)

node\_cpu

$$(1 - ((\text{sum}(\text{increase}(\text{node\_cpu}\{\text{mode}=\text{"idle"}\}[1\text{m}])) \text{ by } (\text{instance}))) / (\text{sum}(\text{increase}(\text{node\_cpu}[1\text{m}])) \text{ by } (\text{instance}))) * 100$$



这才是 真正的CPU使用率 V%



呵呵... 现在还觉得promethues 和 Linux 很简单吗?

其实话说回来， prometheus的这种精细的 底层的计算特性 虽然学起来难  
不过带来的 好处也是 显而易见

- 1) prometheus 这种底层数据采集 所形成的监控 其实是最准确 最可信的
- 2) prometheus 本身也逼着使用它的运维同学 你不踏实下来 好好的真正把Linux技术学过关的话 你就没有办法使用好 这个超强力的监控工具了

所以说呢 我们还是踏踏实实的 一步一个脚印的来学习吧 加油~

最后再额外提两句

- 1) 最后使用的 这个复杂的cpu计算 prometheus公式， 我们从下一讲开始 会给大家 详细讲解（不用着急哈）
- 2) 对于Linux操作系统 底层的各种深入知识学习 请关注 大米运维课堂的 系列课程 第三 / 第四阶段 高级=>专级课程（面向 5-10年经验工程师）  
（今天随口提的这个CPU时间计算 只不过是Linux开始深入的 冰山一角而已）

本节课结束 更多内容 请关注后续