# Prometheus对应的参数

## 主机

#cpu usage cpu使用情况

100 - (avg by (instance) (irate(node\_cpu{mode="idle"}[5m])) \* 100)

#cpu系统使用情况

(avg by (instance) (irate(node\_cpu{mode="system"}[2h])) \* 100)

#cpu用户使用情况

(avg by (instance) (irate(node\_cpu{mode="system"}[2h])) \* 100)

#内存总量

node\_memory\_MemTotal

node\_memory\_MemFree

node\_memory\_Cached

node\_memory\_Buffers

#内存使用情况

(node\_memory\_MemTotal-node\_memory\_Buffers-node\_memory\_MemFree-node\_memory\_Cached)

#内存使用率

(node\_memory\_MemTotal-node\_memory\_Buffers-node\_memory\_MemFree-node\_memory\_Cached)\*100/node\_memory\_MemTotal

irate(node\_network\_receive\_bytes{device='eno16777984'}[2h])

irate(node\_network\_transmit\_bytes{device='eno16777984'}[2h])

rate(node\_disk\_bytes\_read{device="sda"}[2h])

rate(node\_disk\_bytes\_written{device="sda"}[2h])

node\_filesystem\_size

node\_filesystem\_free/node\_filesystem\_size

(node\_filesystem\_size-node\_filesystem\_free)\*100/node\_filesystem\_size

## 容器

container\_network\_transmit\_bytes\_total{kubernetes\_pod\_name="apitangrest4m4c2b95-i03jo"}

container\_memory\_usage\_bytes{kubernetes\_pod\_name=~"^apidengxiaoqian-userd4udu89b6.\*$"}

## http://192.168.0.179:9100/metrics

# 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="idle"} 4.88543617e+06

node\_cpu{cpu="cpu0",mode="iowait"} 18362.54

node\_cpu{cpu="cpu0",mode="irq"} 0

node\_cpu{cpu="cpu0",mode="nice"} 340.36

node\_cpu{cpu="cpu0",mode="softirq"} 3451.06

node\_cpu{cpu="cpu0",mode="steal"} 0

node\_cpu{cpu="cpu0",mode="system"} 107359.96

node\_cpu{cpu="cpu0",mode="user"} 204316.5

node\_cpu{cpu="cpu1",mode="guest"} 0

node\_cpu{cpu="cpu1",mode="idle"} 4.87479824e+06

node\_cpu{cpu="cpu1",mode="iowait"} 17666.37

node\_cpu{cpu="cpu1",mode="irq"} 0

node\_cpu{cpu="cpu1",mode="nice"} 338.44

node\_cpu{cpu="cpu1",mode="softirq"} 5023.73

node\_cpu{cpu="cpu1",mode="steal"} 0

node\_cpu{cpu="cpu1",mode="system"} 109965.79

node\_cpu{cpu="cpu1",mode="user"} 209833.96

## open-falcon请求参数

https://paas.dev.yihecloud.com/monitor/apps/monitors/nodes?param=192.168.0.179&grpId=242&cpu=81.0&mem=82.0&max=3776&disk=83.0

start:1494222189

step:60

timeLength:2

endpointCounters[0][counter]:cpu.busy

endpointCounters[0][endpoint]:192.168.0.179

endpointCounters[1][counter]:cpu.system

endpointCounters[1][endpoint]:192.168.0.179

endpointCounters[2][counter]:cpu.user

endpointCounters[2][endpoint]:192.168.0.179

endpointCounters[3][counter]:mem.memtotal

endpointCounters[3][endpoint]:192.168.0.179

endpointCounters[4][counter]:mem.memused

endpointCounters[4][endpoint]:192.168.0.179

endpointCounters[5][counter]:mem.memused.percent

endpointCounters[5][endpoint]:192.168.0.179

endpointCounters[6][counter]:net.if.in.bytes/iface=eno16777984

endpointCounters[6][endpoint]:192.168.0.179

endpointCounters[7][counter]:net.if.out.bytes/iface=eno16777984

endpointCounters[7][endpoint]:192.168.0.179

endpointCounters[8][counter]:disk.io.read\_bytes/device=sda

endpointCounters[8][endpoint]:192.168.0.179

endpointCounters[9][counter]:disk.io.read\_bytes/device=sdb

endpointCounters[9][endpoint]:192.168.0.179

endpointCounters[10][counter]:disk.io.write\_bytes/device=sda

endpointCounters[10][endpoint]:192.168.0.179

endpointCounters[11][counter]:disk.io.write\_bytes/device=sdb

endpointCounters[11][endpoint]:192.168.0.179

endpointCounters[12][counter]:df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/400beb3d6cd6b47ad417a92ce7bda96899248c3eaeefda1757f300f4e8b0f045/shm

endpointCounters[12][endpoint]:192.168.0.179

endpointCounters[13][counter]:df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/943a7709ae9c4d39417e3711ab018d675e98d2aaead39a33fe2df56abba02c71/shm

endpointCounters[13][endpoint]:192.168.0.179

endpointCounters[14][counter]:df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/aa31b77d5ed5e73ea3f360160061b22af016ce12e5c3f66425278393c856696c/shm

endpointCounters[14][endpoint]:192.168.0.179

endpointCounters[15][counter]:df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/baeefd5d950ee8191b3be76a78e0f4161af5808382ab1e2e7282c943ee982c57/shm

endpointCounters[15][endpoint]:192.168.0.179

endpointCounters[16][counter]:df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/c154226e4d96e45c161c46bfbe13346ecb80db52c489fd191084e806ca501c7d/shm

endpointCounters[16][endpoint]:192.168.0.179

endpointCounters[17][counter]:df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/c3c502cee3a6b97877c26708e2c8356ae2b0e1c0adb3c948da39dbb179d9ff6a/shm

endpointCounters[17][endpoint]:192.168.0.179

endpointCounters[18][counter]:df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/c3e91e319fa998235048b92fd2796920d1acc6c9b5629f470d368d0ff25f3d5f/shm

endpointCounters[18][endpoint]:192.168.0.179

endpointCounters[19][counter]:df.bytes.used.percent/fstype=xfs,mount=/

endpointCounters[19][endpoint]:192.168.0.179

endpointCounters[20][counter]:df.bytes.used.percent/fstype=xfs,mount=/boot

endpointCounters[20][endpoint]:192.168.0.179

endpointCounters[21][counter]:df.bytes.used.percent/fstype=xfs,mount=/data

endpointCounters[21][endpoint]:192.168.0.179

endpointCounters[22][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker

endpointCounters[22][endpoint]:192.168.0.179

endpointCounters[23][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/03c169228a0c317eff71bbc07dc24fefe2c911fc807d79e5e7ebac68f2204783

endpointCounters[23][endpoint]:192.168.0.179

endpointCounters[24][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/0be66d4cbc9cab1dfe13589022949b24e7022419f961f713f0da3f4fb80680d1

endpointCounters[24][endpoint]:192.168.0.179

endpointCounters[25][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/35c5948f39edc9aeab275ef3c6162efed6df33b60d8f68af9025835bd724dcfa

endpointCounters[25][endpoint]:192.168.0.179

endpointCounters[26][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/4ceaac54e414287f3dc1a9de9ba99c4da3819577297d6353069250385a58ea5b

endpointCounters[26][endpoint]:192.168.0.179

endpointCounters[27][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/9a27acd8b777f209529481ab485ba4be8586930907a9b09d9b9cfbf0e478d47b

endpointCounters[27][endpoint]:192.168.0.179

endpointCounters[28][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/cb04a312c0388ce5d5102ebae4ff5ed0354095223f5d2bc6c757d81172ecf274

endpointCounters[28][endpoint]:192.168.0.179

endpointCounters[29][counter]:df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/dff364e38e1c2e2785e176071493b2423fa05c6a306d745db8f6f8c354a56c66

endpointCounters[29][endpoint]:192.168.0.179

## open-falcon获取结果

{

"cf": "AVERAGE",

"code": 0,

"data": [{

"name": null,

"endpoint": "192.168.0.179",

"counter": "cpu.busy",

"data": [{

"x": 1494223500000,

"y": 4.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "cpu.system",

"data": [{

"x": 1494223500000,

"y": 1.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "cpu.user",

"data": [{

"x": 1494223500000,

"y": 1.500000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "mem.memtotal",

"data": [{

"x": 1494223500000,

"y": 3959754752.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "mem.memused",

"data": [{

"x": 1494223500000,

"y": 2921119744.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "mem.memused.percent",

"data": [{

"x": 1494223500000,

"y": 73.770219

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "net.if.in.bytes/iface=eno16777984",

"data": [{

"x": 1494223500000,

"y": 34565.933333

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "net.if.out.bytes/iface=eno16777984",

"data": [{

"x": 1494223500000,

"y": 7996.433333

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "disk.io.read\_bytes/device=sda",

"data": [{

"x": 1494223500000,

"y": 0.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "disk.io.read\_bytes/device=sdb",

"data": [{

"x": 1494223500000,

"y": 24576.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "disk.io.write\_bytes/device=sda",

"data": [{

"x": 1494223500000,

"y": 0.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "disk.io.write\_bytes/device=sdb",

"data": [{

"x": 1494223500000,

"y": 138240.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/400beb3d6cd6b47ad417a92ce7bda96899248c3eaeefda1757f300f4e8b0f045/shm",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/943a7709ae9c4d39417e3711ab018d675e98d2aaead39a33fe2df56abba02c71/shm",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/aa31b77d5ed5e73ea3f360160061b22af016ce12e5c3f66425278393c856696c/shm",

"data": [{

"x": 1494223500000,

"y": 0.000000

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/baeefd5d950ee8191b3be76a78e0f4161af5808382ab1e2e7282c943ee982c57/shm",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/c154226e4d96e45c161c46bfbe13346ecb80db52c489fd191084e806ca501c7d/shm",

"data": [{

"x": 1494223500000,

"y": null

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/c3c502cee3a6b97877c26708e2c8356ae2b0e1c0adb3c948da39dbb179d9ff6a/shm",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=tmpfs,mount=/var/lib/docker/containers/c3e91e319fa998235048b92fd2796920d1acc6c9b5629f470d368d0ff25f3d5f/shm",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/",

"data": [{

"x": 1494223500000,

"y": 69.868235

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/boot",

"data": [{

"x": 1494223500000,

"y": 30.564622

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/data",

"data": [{

"x": 1494223500000,

"y": 70.092575

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/03c169228a0c317eff71bbc07dc24fefe2c911fc807d79e5e7ebac68f2204783",

"data": [{

"x": 1494223500000,

"y": null

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/0be66d4cbc9cab1dfe13589022949b24e7022419f961f713f0da3f4fb80680d1",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/35c5948f39edc9aeab275ef3c6162efed6df33b60d8f68af9025835bd724dcfa",

"data": [{

"x": 1494223500000,

"y": 2.802846

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/4ceaac54e414287f3dc1a9de9ba99c4da3819577297d6353069250385a58ea5b",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/9a27acd8b777f209529481ab485ba4be8586930907a9b09d9b9cfbf0e478d47b",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/cb04a312c0388ce5d5102ebae4ff5ed0354095223f5d2bc6c757d81172ecf274",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}, {

"name": null,

"endpoint": "192.168.0.179",

"counter": "df.bytes.used.percent/fstype=xfs,mount=/var/lib/docker/devicemapper/mnt/dff364e38e1c2e2785e176071493b2423fa05c6a306d745db8f6f8c354a56c66",

"data": [{

"x": 1494223500000,

"y": 9.707861

}

]

}

],

"start": 1494223498,

"end": 1494223501

}

# GitLab 8.16 Now Includes Monitoring and Extends Auto Deploy to Google Container Engine

- 推酷 http://www.tuicool.com/articles/NZ3EJfb

时间 2017-01-28 00:00:00  [InfoQ](http://www.tuicool.com/sites/FbANJb)

原文  [http://www.infoq.com/news/2017/01/gitlab-816-gce](http://www.tuicool.com/articles/hit/NZ3EJfb)

主题 [GitLab](http://www.tuicool.com/topics/11110179)

Following the introduction of [***auto-deploy to Kubernetes on OpenShift***](https://www.infoq.com/news/2016/12/gitlab-8-15-released) last month, GitLab 8.16 makes auto-deploy available on Google Cloud. Additionally, GitLab 8.16 improves its issue search and filter UI, and includes monitoring tool [***Prometheus***](https://prometheus.io/) and Slack-alternative [***Mattermost***](https://about.mattermost.com/) .

[***According***](https://about.gitlab.com/2017/01/22/gitlab-8-16-released/) to GitLab VP of product, Job van der Voort, making auto-deploy available on Google Container Engine (GCE) allows a much larger developer audience to take advantage of the possibility of deploying an app to Kubernetes from a GitLab instance using its [***auto-scaling CI***](https://about.gitlab.com/2016/03/29/gitlab-runner-1-1-released/)features. [***A number of steps***](https://gitlab.com/gitlab-org/kubernetes-gitlab-demo/blob/master/README.md) are required to deploy GitLab to Kubernetes on GCE, which makes the entire process not entirely straightforward, although it can be completed in [***less than 30 minutes***](https://www.youtube.com/watch?v=3A8mdJl_icM) according to GitLab.

The inclusion of Prometheus within GitLab 8.16 is the first step in GitLab’s [***roadmap to make monitoring an integral part of GitLab CI***](https://about.gitlab.com/2017/01/05/prometheus-and-gitlab/) . Prometheus should make it possible to gather early feedback about deployments and automatically revert those that cause problems. To be able to connect to Prometheus console, which can provide metrics relating to CPU, memory, and throughput, you should first set up port forwarding so the private Prometheus server becomes accessible on **localhost:9090** :

kubectl -n gitlab get pods -l name=gitlab -o name | sed 's/^.\*\///' | xargs -I{} kubectl port-forward -n gitlab {} 9090:9090

Following are a few examples of queries that can be sent to Prometheus:

* % Memory Used: **(1 - ((node\_memory\_MemFree + node\_memory\_Cached) / node\_memory\_MemTotal)) \* 100**
* % CPU Load: **1 - rate(node\_cpu{mode="idle"}[5m])**
* Data Transmitted: **irate(node\_network\_transmit\_bytes[5m])**
* Data Received: **irate(node\_network\_receive\_bytes[5m])**

Other notable features in GitLab 8.16 are:

* Improved issue search and filter interface to make it more natural and intuitive, according to GitLab.
* Support for removing merge request approval.
* Support for deploy keys granting write privilege, in addition to already existing read-only deploy keys.
* New **merge** command allows to merge a PR by simply typing **/merge** in the description or comment to an issue.
* GitLab Runner 1.10, including a bunch of [***improvements and fixes***](https://gitlab.com/gitlab-org/gitlab-ci-multi-runner/blob/v1.10.0/CHANGELOG.md) .
* Mattermost 3.6 is now included in GitLab.

GitLab 8.16 can be [***installed***](https://about.gitlab.com/installation/) or [***updated***](https://about.gitlab.com/update/) using a variety of methods, including images for several hosting providers, Docker containers, or OS-specific packages.

# GitLab.org / kubernetes-gitlab-demo · GitLab

https://gitlab.com/gitlab-org/kubernetes-gitlab-demo

继上个月在[OpenShift上引入自动部署支持Kubernetes](https://www.infoq.com/news/2016/12/gitlab-8-15-released)后，GitLab 8.16在Google Cloud上提供了自动部署功能。此外，GitLab 8.16改进了其问题搜索和过滤器界面，并包括监控工具[Prometheus](https://prometheus.io/)和Slack的替代者[Mattermost](https://about.mattermost.com/)。

[据Gitlab产品副总裁](https://about.gitlab.com/2017/01/22/gitlab-8-16-released/)Job van der Voort介绍，在Google Container Engine（GCE）上提供自动部署功能，GitLab实例使用其[自动缩放持续集成](https://about.gitlab.com/2016/03/29/gitlab-runner-1-1-released/)（auto-scaling CI）功能部署应用到Kubernetes，将允许更多的开发者从这种可能性中获得好处。根据Gitlab，在GCE上将GitLab部署到Kubernetes需要[许多步骤](https://gitlab.com/gitlab-org/kubernetes-gitlab-demo/blob/master/README.md)，尽管它可以在[不到30分钟](https://www.youtube.com/watch?v=3A8mdJl_icM)内完成，这仍然使整个过程不那么直观。

[为使监控成为Gitlab持续集成的一个组成部分](https://about.gitlab.com/2017/01/05/prometheus-and-gitlab/)，产品规划中的第一步就是在GitLab 8.16中包含Prometheus。Prometheus应该能够收集关于部署的早期反馈，并自动回退那些导致问题的部署。为了能够连接到可以提供与CPU、内存和吞吐量相关的指标的Prometheus控制台，你应该首先设置端口转发，以便在localhost：9090上访问私有的Prometheus服务器：

kubectl -n gitlab get pods -l name=gitlab -o name | sed 's/^.\*\///' | xargs -I{} kubectl port-forward -n gitlab {} 9090:9090

以下是几个可以发送到Prometheus的查询示例：

* 内存使用百分比：(1 - ((node\_memory\_MemFree + node\_memory\_Cached) / node\_memory\_MemTotal)) \* 100
* CPU负荷百分比：1 - rate(node\_cpu{mode="idle"}[5m])
* 发送的数据：irate(node\_network\_transmit\_bytes[5m])
* 接收的数据：irate(node\_network\_receive\_bytes[5m])

GitLab 8.16的其他重要特性包括：

* 改进的问题搜索和过滤器界面，使其更自然和直观。
* 支持撤销合并请求的批准。
* 支持授予写入权限的部署密钥，以及现有的只读部署密钥。
* 新的merge命令允许在问题的描述或注释中简单地键入/merge来合并PR。
* GitLab Runner 1.10，包括一系列[改进和修复](https://gitlab.com/gitlab-org/gitlab-ci-multi-runner/blob/v1.10.0/CHANGELOG.md)。
* Mattermost 3.6现在包含在GitLab中。

GitLab 8.16可以使用各种方法[安装](https://about.gitlab.com/installation/)或[更新](https://about.gitlab.com/update/)，其中包括使用支持多个托管提供商的镜像，Docker容器以及针对特定操作系统的软件包。

**查看英文原文** ： [GitLab 8.16 Now Includes Monitoring and Extends Auto Deploy to Google Container Engine](https://www.infoq.com/news/2017/01/gitlab-816-gce)

感谢[王纯超](http://www.infoq.com/cn/author/%E7%8E%8B%E7%BA%AF%E8%B6%85)对本文的审校。

# 2@Undoing the benefits of labels | Robust Perception

https://www.robustperception.io/undoing-the-benefits-of-labels/

**It can seem like a good idea to use recording rules to make more explicit the content of a time series, particularly for those not used to labels. However this usually leads to confusing names and losing the benefits of labels.**

**Every so often you come across**[**recording rules**](https://prometheus.io/docs/querying/rules/)**of the form:**

**node\_disk\_bytes\_read:sda = rate(node\_disk\_bytes\_read{device="sda"}[5m])**

**node\_disk\_bytes\_read:sdb = rate(node\_disk\_bytes\_read{device="sdb"}[5m])**

**node\_disk\_bytes\_read:sdc = rate(node\_disk\_bytes\_read{device="sdc"}[5m])**

**node\_md\_disks:md0 = node\_md\_disks{device="md0"}**

**This is making work for yourself, while also losing one of the biggest wins of Prometheus – labels.**

**With rules like this every time there’s a new label value you need to update your rules. You cannot manipulate these style of time series en-masse, as they don’t share a metric name. You may also be introducing additional race conditions and graph artifacts. The name also makes it unclear exactly what the time series represents.**

**If you want to refer to the number of disks in md0 the canonical way to do that is:**

**node\_md\_disks{device="md0"}**

**It’s about the same length, clarifies what md0 is and there’s no need for recording rules!**

**If you are doing some computation and want to record the values, it’s best to do it all at once:**

**instance\_device:node\_disk\_bytes\_read:rate5m = rate(node\_disk\_bytes\_read{job="node"}[5m])**

**This doesn’t need updating as you gain new disks, and specifying job="node" means you won’t accidentally apply this rule to other unrelated jobs that happen to also export a metric by this name.**

**The instance\_device:node\_disk\_bytes\_read:rate5m name has further advantages. The instance\_device tells you exactly which labels are in play, making it easier to visually inspect expressions for correct label handling. The rate5m at the end lets you know what calculations have been performed on it.**

**You may be tempted to encode other information into recording rules names such as some token to indicate whether to federate individual time series. This is to be avoided as it is unrelated to the identity of the data and thus may change over time, similar to how**[**target labels should be kept constant**](http://www.robustperception.io/target-labels-are-for-life-not-just-for-christmas/)**. In general, a recording rule should either have zero colons or two colons.**

**The full conventions for naming recording rules are part of the Prometheus**[**best practices**](https://prometheus.io/docs/practices/rules/)**. With everyone following the same naming scheme everyone wins, as at a glance we can all understand and reuse each others’ rules!**

[**best practices**](https://www.robustperception.io/tag/best-practices/)**,**[**prometheus**](https://www.robustperception.io/tag/prometheus/)**,**[**promql**](https://www.robustperception.io/tag/promql/)

# performance - Prometheus - Convert cpu\_user\_seconds to CPU Usage %?

- Stack Overflow

http://stackoverflow.com/questions/34923788/prometheus-convert-cpu-user-seconds-to-cpu-usage

Currently i'm monitoring docker containers via Prometheus.io. My problem is that i'm just getting "cpu\_user\_seconds\_total" or "cpu\_system\_seconds\_total". My question is how to convert this ever-increasing value to a CPU percentage?

Currently i'm querying:

rate(container\_cpu\_user\_seconds\_total[30s])

But I don't think that it is quite correct (comparing to top).

How to convert cpu\_user\_seconds\_total to CPU percentage? (Like in top)

Rate returns a per second value, so multiplying by 100 will give a percentage:

rate(container\_cpu\_user\_seconds\_total[30s]) \* 100

I also found this way to get CPU Usage to be accurate:

100 - (avg by (instance) (irate(node\_cpu{job="node",mode="idle"}[5m])) \* 100)

From: <http://www.robustperception.io/understanding-machine-cpu-usage/>

rate(node\_cpu{mode="idle"}[5m])

# 2@Understanding Machine CPU usage | Robust Perception

https://www.robustperception.io/understanding-machine-cpu-usage/

**High CPU load is a common cause of issues. Let’s look at how to dig into it with**[**Prometheus**](https://prometheus.io/)**and the**[**Node exporter**](https://github.com/prometheus/node_exporter)**.**

**On a**[**Node exporters’ metrics page**](http://demo.robustperception.io:9100/metrics)**, part of the output is:**

**# 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="idle"} 2.03442237e+06**

**node\_cpu{cpu="cpu0",mode="iowait"} 3522.37**

**node\_cpu{cpu="cpu0",mode="irq"} 0.48**

**node\_cpu{cpu="cpu0",mode="nice"} 515.56**

**node\_cpu{cpu="cpu0",mode="softirq"} 953.06**

**node\_cpu{cpu="cpu0",mode="steal"} 0**

**node\_cpu{cpu="cpu0",mode="system"} 6605.46**

**node\_cpu{cpu="cpu0",mode="user"} 23343.01**

**node\_cpu{cpu="cpu1",mode="guest"} 0**

**node\_cpu{cpu="cpu1",mode="idle"} 2.03471439e+06**

**node\_cpu{cpu="cpu1",mode="iowait"} 3633.5**

**node\_cpu{cpu="cpu1",mode="irq"} 0.58**

**node\_cpu{cpu="cpu1",mode="nice"} 542.05**

**node\_cpu{cpu="cpu1",mode="softirq"} 880.49**

**node\_cpu{cpu="cpu1",mode="steal"} 0**

**node\_cpu{cpu="cpu1",mode="system"} 6581.92**

**node\_cpu{cpu="cpu1",mode="user"} 23171.06**

**This metric comes from /proc/stat and tell us how many seconds each CPU spent doing each type of work:**

* **user: The time spent in userland**
* **system: The time spent in the kernel**
* **iowait: Time spent waiting for I/O**
* **idle: Time the CPU had nothing to do**
* **irq&softirq: Time servicing interrupts**
* **guest: If you are running VMs, the CPU they use**
* **steal: If you are a VM, time other VMs “stole” from your CPUs**

**These modes are mutually exclusive. A high iowait means that you are disk or network bound, high user or system means that you are CPU bound.**

**These are counters, so to calculate the per-second values we use theirate function in the**[**expression browser**](http://demo.robustperception.io:9090/graph#%5B%7B%22range_input%22%3A%221h%22%2C%22end_input%22%3A%22%22%2C%22step_input%22%3A%22%22%2C%22stacked%22%3A%22%22%2C%22expr%22%3A%22irate(node_cpu%7Bjob%3D%5C%22node%5C%22%7D%5B5m%5D)%22%2C%22tab%22%3A1%7D%5D)**:**

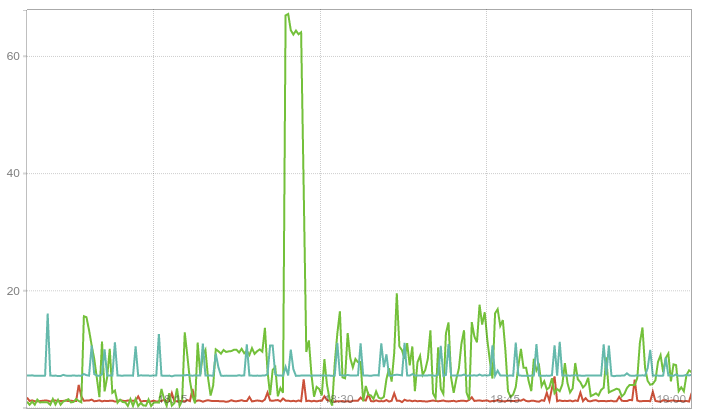
**irate(node\_cpu{job="node"}[5m])**

**We can aggregate this to get the overall value across all CPUs for the machine:**

**sum by (mode, instance) (irate(node\_cpu{job="node"}[5m]))**

**As these values always sum to one second per second for each cpu, the per-second rates are also the ratios of usage. We can use this to calculate the percentage of CPU used, by subtracting the idle usage from 100%:**

**100 - (avg by (instance) (irate(node\_cpu{job="node",mode="idle"}[5m])) \* 100)**

**[](http://www.robustperception.io/wp-content/uploads/2015/12/Screenshot-301215-190415-e1451502301338.png)**

**CPU Used % across several machines**

[**node exporter**](https://www.robustperception.io/tag/node-exporter/)**,**[**prometheus**](https://www.robustperception.io/tag/prometheus/)**,**[**promql**](https://www.robustperception.io/tag/promql/)

# prometheus/node\_exporter:

Exporter for machine metrics

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

Node exporter

Prometheus exporter for hardware and OS metrics exposed by \*NIX kernels, written in Go with pluggable metric collectors.

The [WMI exporter](https://github.com/martinlindhe/wmi_exporter) is recommended for Windows users.

Collectors

There is varying support for collectors on each operating system. The tables below list all existing collectors and the supported systems.

Which collectors are used is controlled by the --collectors.enabled flag.

Enabled by default

| **Name** | **Description** | **OS** |
| --- | --- | --- |
| arp | Exposes ARP statistics from /proc/net/arp. | Linux |
| conntrack | Shows conntrack statistics (does nothing if no/proc/sys/net/netfilter/ present). | Linux |
| cpu | Exposes CPU statistics | Darwin, Dragonfly, FreeBSD |
| diskstats | Exposes disk I/O statistics from /proc/diskstats. | 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 configurations. | 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 |
| 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 |
| sockstat | Exposes various statistics from /proc/net/sockstat. | Linux |
| stat | Exposes various statistics from /proc/stat. This includes CPU usage, 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 |
| 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](http://open-zfs.org/) performance statistics. | [Linux](http://zfsonlinux.org/) |

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 | Linux |
| interrupts | Exposes detailed interrupts statistics. | Linux, OpenBSD |
| ipvs | Exposes IPVS status from /proc/net/ip\_vs and stats from /proc/net/ip\_vs\_stats. | Linux |
| ksmd | Exposes kernel and system statistics from /sys/kernel/mm/ksm. | Linux |
| logind | Exposes session counts from [logind](http://www.freedesktop.org/wiki/Software/systemd/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 |
| nfs | Exposes NFS client statistics from /proc/net/rpc/nfs. This is the same information asnfsstat -c. | Linux |
| runit | Exposes service status from [runit](http://smarden.org/runit/). | any |
| supervisord | Exposes service status from [supervisord](http://supervisord.org/). | any |
| systemd | Exposes service and system status from [systemd](http://www.freedesktop.org/wiki/Software/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.) | Linux |

Deprecated

These collectors will be (re)moved in the future.

| **Name** | **Description** | **OS** |
| --- | --- | --- |
| gmond | Exposes statistics from Ganglia. | any |
| megacli | Exposes RAID statistics from MegaCLI. | Linux |
| ntp | Exposes time drift from an NTP server. | any |

Textfile Collector

The textfile collector is similar to the [Pushgateway](https://github.com/prometheus/pushgateway), in that it allows exporting of statistics from batch jobs. It can also be used to export static metrics, such as what role a machine has. The Pushgateway should be used for service-level metrics. The textfile module is for metrics that are tied to a machine.

To use it, set the --collector.textfile.directory flag on the Node exporter. The collector will parse all files in that directory matching the glob \*.prom using the [text format](http://prometheus.io/docs/instrumenting/exposition_formats/).

To atomically push completion time for a cron job:

echo my\_batch\_job\_completion\_time $(date +%s) > /path/to/directory/my\_batch\_job.prom.$$

mv /path/to/directory/my\_batch\_job.prom.$$ /path/to/directory/my\_batch\_job.prom

To statically set roles for a machine using labels:

echo 'role{role="application\_server"} 1' > /path/to/directory/role.prom.$$

mv /path/to/directory/role.prom.$$ /path/to/directory/role.prom

Building and running

make

./node\_exporter <flags>

To see all available configuration flags:

./node\_exporter -h

Running tests

make test

Using Docker

The node\_exporter is designed to monitor the host system. It's not recommended to deploy it as Docker container because it requires access to the host system. If you need to run it on Docker, you can deploy this exporter using the [node-exporter Docker image](https://quay.io/repository/prometheus/node-exporter) with the following options and bind-mounts:

docker run -d -p 9100:9100 \

-v "/proc:/host/proc" \

-v "/sys:/host/sys" \

-v "/:/rootfs" \

--net="host" \

quay.io/prometheus/node-exporter \

-collector.procfs /host/proc \

-collector.sysfs /host/sys \

-collector.filesystem.ignored-mount-points "^/(sys|proc|dev|host|etc)($|/)"

Be aware though that the mountpoint label in various metrics will now have /host as prefix.

Using a third-party repository for RHEL/CentOS/Fedora

There is a [community-supplied COPR repository](https://copr.fedorainfracloud.org/coprs/ibotty/prometheus-exporters/). It closely follows upstream releases.