# prom/container-exporter - Docker Hub

https://hub.docker.com/r/prom/container-exporter/

**Container Exporter**

[Prometheus](https://github.com/prometheus/prometheus) exporter exposing container metrics.

The container-exporter requests a list of containers running on the host by talking to a  
container manager. Right now, Docker as container manager is supported.  
It then gathers various container metrics by using [libcontainer](https://github.com/docker/libcontainer)  
and [DockerClient](https://github.com/fsouza/go-dockerclient) and then exposes them for prometheus' consumption.

**Run it as container**

docker run -p 9104:9104 -v /sys/fs/cgroup:/cgroup \

-v /var/run/docker.sock:/var/run/docker.sock prom/container-exporter

**Support for labels**

Specify all Docker label whose values you would like to tag your Prometheus metrics with by using the -labels parameter to the container exporter binary (or docker container). For example if you have a container labeled with LabelA and LabelB and a second container labeled with LabelB and LabelC as shown below. You can launch container exporter with the parameter -labels=LabelA,LabelB,LabelC.

docker run --name ContainerA --label LabelA=ValueA --label LabelB=ValueB [IMAGE]

docker run --name ContainerB --label LabelB=ValueB --label LabelC=ValueC [IMAGE]

docker run -p 9104:9104 -v /sys/fs/cgroup:/cgroup \

-v /var/run/docker.sock:/var/run/docker.sock prom/container-exporter -labels=LabelA,LabelB,LabelC

This will load to the metrics shown below. Note that an empty string is reported for any container that does not define a label that is specified to container exporter.

container\_cpu\_throttled\_periods\_total{LabelA="ValueA",LabelB="ValueB",LabelC="",name="ContainerA"...

container\_cpu\_throttled\_periods\_total{LabelA="",LabelB="ValueB",LabelC="ValueC",name="ContainerB"...

Docker Pull Command



prom

Source Repository

[docker-infra/container\_exporter](https://github.com/docker-infra/container_exporter)

# prom/node-exporter – Docker Hub

https://hub.docker.com/r/prom/node-exporter/

Node exporter

Prometheus exporter for machine metrics, written in Go with pluggable metric  
collectors.

**Building and running**

make

./node\_exporter <flags>

**Running tests**

make test

**Available collectors**

By default the build will include the native collectors that expose information  
from /proc.

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

**Enabled by default**

| **Name** | **Description** |
| --- | --- |
| attributes | Exposes attributes from the configuration file. Deprecated, use textfile module instead. |
| diskstats | Exposes disk I/O statistics from /proc/diskstats. |
| filesystem | Exposes filesystem statistics, such as disk space used. |
| loadavg | Exposes load average. |
| meminfo | Exposes memory statistics from /proc/meminfo. |
| netdev | Exposes network interface statistics from /proc/netstat, such as bytes transferred. |
| netstat | Exposes network statistics from /proc/net/netstat. This is the same information as netstat -s. |
| stat | Exposes various statistics from /proc/stat. This includes CPU usage, boot time, forks and interrupts. |
| textfile | Exposes statistics read from local disk. The --collector.textfile.directory flag must be set. |
| time | Exposes the current system time. |

**Disabled by default**

| **Name** | **Description** |
| --- | --- |
| bonding | Exposes the number of configured and active slaves of Linux bonding interfaces. |
| gmond | Exposes statistics from Ganglia. |
| interrupts | Exposes detailed interrupts statistics from /proc/interrupts. |
| lastlogin | Exposes the last time there was a login. |
| megacli | Exposes RAID statistics from MegaCLI. |
| ntp | Exposes time drift from an NTP server. |
| runit | Exposes service status from [runit](http://smarden.org/runit/). |

**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 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

This is what i do and it works fine:  
1) Mount these

* /proc:/host/proc
* /sys:/host/sys
* /:/rootfs
* /etc/hostname:/etc/host\_hostname

Use these arguments:  
-collector.procfs /host/proc \  
-collector.sysfs /host/sys \  
-collector.filesystem.ignored-mount-points "^/(sys|proc|dev|host|etc)($|/)" \  
--collector.textfile.directory /etc/node-exporter/ \  
--collectors.enabled="conntrack,diskstats,entropy,filefd,filesystem,loadavg,mdadm,meminfo,netdev,netstat,stat,textfile,time,vmstat,ipvs"

docker run -d -p 9100:9100 prom/node-exporter:v0.14.0-rc.1  
617a97b0332db716c79c43586eedf174e9a53c34399db21ab050324e5cea0b41  
docker: Error response from daemon: Container command '/bin/node\_exporter' not found or does not exist

Anyone help me?

# prometheus/node-exporter · Quay

https://quay.io/repository/prometheus/node-exporter

docker pull quay.io/prometheus/node-exporter

docker pull prom/node-exporter

docker run -d -p 9100:9100 --net="host" prom/node-exporter

# prom/mysqld-exporter - Docker Hub

https://hub.docker.com/r/prom/mysqld-exporter/

Prometheus exporter for MySQL server metrics.  
Supported MySQL versions: 5.1 and up.  
NOTE: Not all collection methods are support on MySQL < 5.6

**Building and running**

**Required Grants**

CREATE USER 'exporter'@'localhost' IDENTIFIED BY 'XXXXXXXX';

GRANT PROCESS, REPLICATION CLIENT ON \*.\* TO 'exporter'@'localhost';

GRANT SELECT ON performance\_schema.\* TO 'exporter'@'localhost';

**Build**

make

**Running**

Running using an environment variable:

export DATA\_SOURCE\_NAME='login:password@(hostname:port)/'

./mysqld\_exporter <flags>

Running using ~/.my.cnf:

./mysqld\_exporter <flags>

**Collector Flags**

| **Name** | **MySQL Version** | **Description** |
| --- | --- | --- |
| collect.global\_status | 5.1 | Collect from SHOW GLOBAL STATUS (Enabled by default) |
| collect.global\_variables | 5.1 | Collect from SHOW GLOBAL VARIABLES (Enabled by default) |
| collect.slave\_status | 5.1 | Collect from SHOW SLAVE STATUS (Enabled by default) |
| collect.binlog\_size | 5.1 | Collect the current size of all registered binlog files |
| collect.info\_schema.innodb\_metrics | 5.6 | Collect metrics from information\_schema.innodb\_metrics. |
| collect.auto\_increment.columns | 5.1 | Collect auto\_increment columns and max values from information\_schema. |
| collect.engine\_tokudb\_status | 5.6 | Collect from SHOW ENGINE TOKUDB STATUS. |
| collect.info\_schema.userstats | 5.1 | If running with userstat=1, set to true to collect user statistics. |
| collect.info\_schema.innodb\_tablespaces | 5.7 | Collect metrics from information\_schema.innodb\_sys\_tablespaces. |
| collect.info\_schema.tablestats | 5.1 | If running with userstat=1, set to true to collect table statistics. |
| collect.info\_schema.tables | 5.1 | Collect metrics from information\_schema.tables. |
| collect.info\_schema.tables.databases | 5.1 | The list of databases to collect table stats for, or '\*' for all. |
| collect.info\_schema.query\_response\_time | 5.5 | Collect query response time distribution if query\_response\_time\_stats is ON. |
| collect.info\_schema.processlist | 5.1 | Collect thread state counts from information\_schema.processlist. |
| collect.info\_schema.processlist.min\_time | 5.1 | Minimum time a thread must be in each state to be counted. (default: 0) |
| collect.perf\_schema.eventsstatements | 5.6 | Collect metrics from performance\_schema.events\_statements\_summary\_by\_digest. |
| collect.perf\_schema.eventsstatements.limit | 5.6 | Limit the number of events statements digests by response time. (default: 250) |
| collect.perf\_schema.eventsstatements.timelimit | 5.6 | Limit how old the 'last\_seen' events statements can be, in seconds. (default: 86400) |
| collect.perf\_schema.eventsstatements.digest\_text\_limit | 5.6 | Maximum length of the normalized statement text. (default: 120) |
| collect.perf\_schema.indexiowaits | 5.6 | Collect metrics from performance\_schema.table\_io\_waits\_summary\_by\_index\_usage. |
| collect.perf\_schema.tableiowaits | 5.6 | Collect metrics from performance\_schema.table\_io\_waits\_summary\_by\_table. |
| collect.perf\_schema.tablelocks | 5.6 | Collect metrics from performance\_schema.table\_lock\_waits\_summary\_by\_table. |
| collect.perf\_schema.file\_events | 5.5 | Collect metrics from performance\_schema.file\_summary\_by\_event\_name. |
| collect.perf\_schema.eventswaits | 5.5 | Collect metrics from performance\_schema.events\_waits\_summary\_global\_by\_event\_name. |

**General Flags**

| **Name** | **Description** |
| --- | --- |
| config.my-cnf | Path to .my.cnf file to read MySQL credentials from. (default: ~/.my.cnf) |
| log.level | Logging verbosity (default: info) |
| log\_slow\_filter | Add a log\_slow\_filter to avoid exessive MySQL slow logging. NOTE: Not supported by Oracle MySQL. |
| web.listen-address | Address to listen on for web interface and telemetry. |
| web.telemetry-path | Path under which to expose metrics. |
| version | Print the version information. |

**Setting the MySQL server's data source name**

The MySQL server's [data source name](http://en.wikipedia.org/wiki/Data_source_name)  
must be set via the DATA\_SOURCE\_NAME environment variable.  
The format of this variable is described at <https://github.com/go-sql-driver/mysql#dsn-data-source-name>.

**Using Docker**

You can deploy this exporter using the [prom/mysqld-exporter](https://registry.hub.docker.com/u/prom/mysqld-exporter/) Docker image.

For example:

docker pull prom/mysqld-exporter

docker run -d -p 9104:9104 --link=my\_mysql\_container:bdd \

-e DATA\_SOURCE\_NAME="user:password@(bdd:3306)/database" prom/mysqld-exporter

# docker运行prometheus - go4it

https://my.oschina.net/go4it/blog/855309

**pull镜像**

docker pull prom/prometheus:latest

**运行**

docker run -p 9090:9090 \

-v /tmp/prometheus-data:/prometheus-data \

prom/prometheus

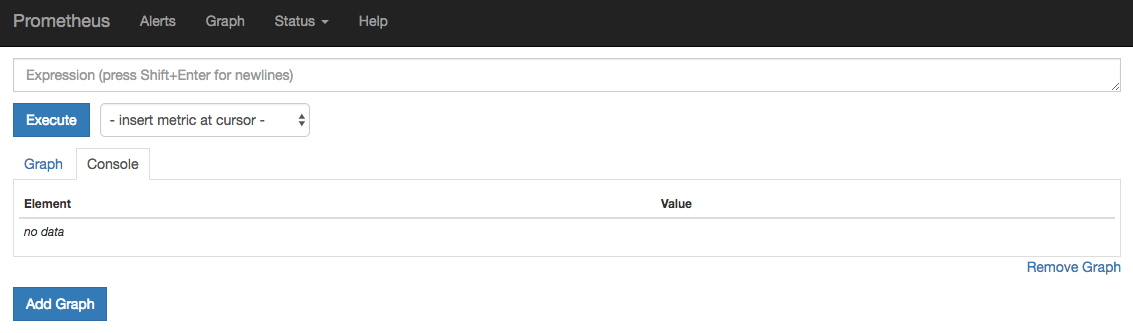
如果要映射配置文件

docker run -p 9090:9090 -v /tmp/prometheus.yml:/etc/prometheus/prometheus.yml \

-v /tmp/prometheus-data:/prometheus-data \

prom/prometheus

**访问**

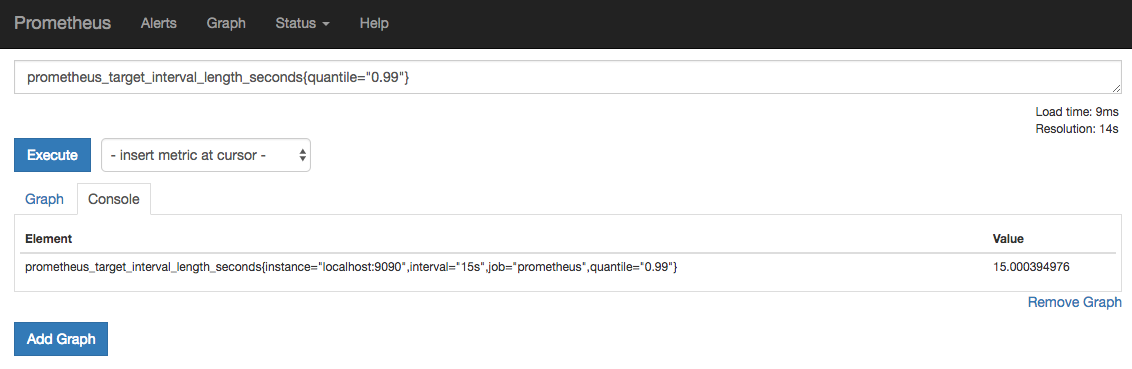
<http://192.168.99.100:9090/graph> 

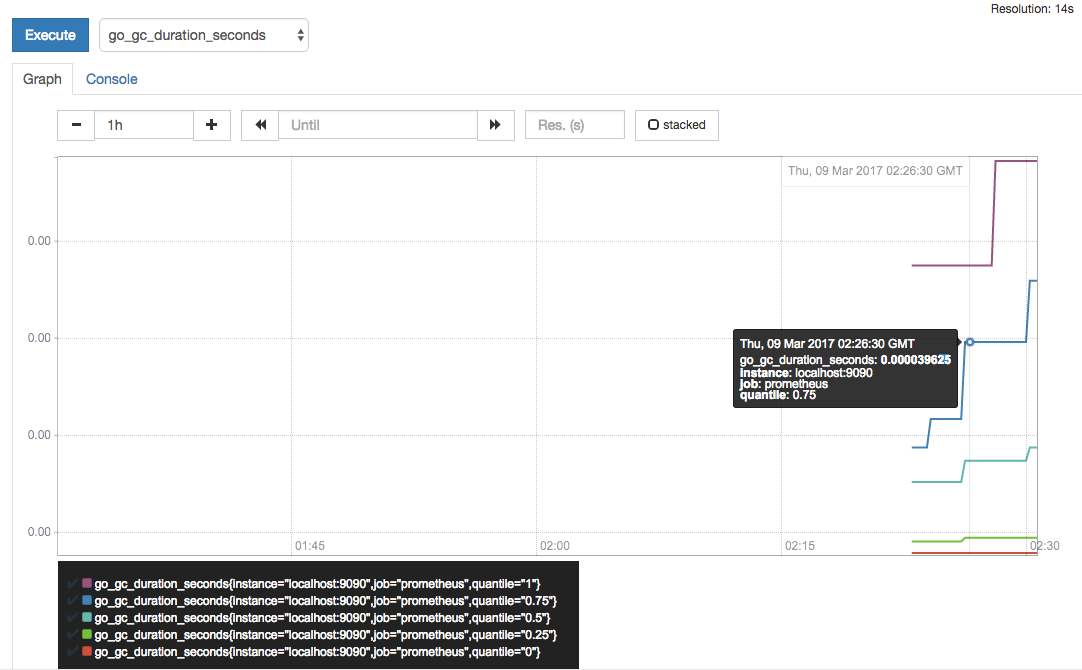
计算实例，指标可以从[http://192.168.99.100:9090/metrics中找](http://192.168.99.100:9090/metrics%E4%B8%AD%E6%89%BE)

prometheus\_target\_interval\_length\_seconds{quantile="0.99"}

或者

count(prometheus\_target\_interval\_length\_seconds)



查看图形 

**doc**

* [prometheus-install](https://prometheus.io/docs/introduction/install/)
* [prometheus expression language documentation](https://prometheus.io/docs/querying/basics/)

# 2@使用 Prometheus 监控 Docker 容器 - 推酷

http://www.tuicool.com/articles/QBfANf

时间 2015-02-03 14:12:05  [极客头条](http://www.tuicool.com/sites/e2aA32)

原文  [http://segmentfault.com/blog/yexiaobai/1190000002527178](http://segmentfault.com/blog/yexiaobai/1190000002527178?utm_source=tuicool&utm_medium=referral)

主题 [Docker](http://www.tuicool.com/topics/11090139)

该文中介绍的 Prometheus 的项目地址是 [***https://prometheus.github.io/***](https://prometheus.github.io/)

监控 Docker

在容器中运行你所有的服务使得获取深度资源和运行特性成为可能，因为每个容器运行在它们自己的 cgroup 中并且 Linux 内核给我们提供了各种各样的指标（metrics）。

尽管有一些其他的 Docker 监控工具，我将给你们展示我为什么认为 SoundCloud [***最新发布的 Prometheus***](https://prometheus.github.io/) 是最适合监控基于容器的基础架构。

Prometheus 特点是 [***高维度数据模型***](http://prometheus.github.io/docs/concepts/data_model/) ，时间序列是通过一个度量值名字和一套键值对识别。灵活的 [***查询语言***](http://prometheus.github.io/docs/querying/basics/) 允许查询和绘制数据。它采用了先进的 [***度量标准类型***](http://prometheus.github.io/docs/concepts/metric_types/) 像汇总（summaries），从指定时间跨度的总数构建 [***比率***](http://prometheus.github.io/docs/querying/functions/#rate%28%29) 或者是在任何异常的时候 [***报警***](http://prometheus.github.io/docs/querying/rules/) 并且没有任何依赖，中断期间使它成为一个可靠的系统进行调试。

我会集中讲为什么该数据模型和查询语言如此贴合容器式和动态基础设施，对于这些基础设施，你应该想着整个服务集群而不是单个服务器实例，把服务器想成牛群中的牛而不是各家自养分散开的宠物。

传统方法

比如说你想监控你容器的内存使用率。不支持维度数据，这样一个名为 webapp123 的容器的指标，可能被称为 container\_memory\_usage\_bytes\_webapp123 。

但是如果你想展示所有你的 webapp123 容器的内存利用率？更先进的监控解决方案像 [***graphite***](https://github.com/graphite-project) 支持这样。它的特性是层次，树状数据模型，这样的指标可能被称为 container.memory\_usage\_bytes.webapp123 。现在你可以使用正则表达式像 container.memory\_usage\_bytes.webapp\* 来绘制所有你的 ‘webapp’ 容器的内存使用率。Graphite 也支持函数像 sum() 来通过使用一个表达式像 sum(container.memory\_usage\_bytes.webapp\*) 聚合你所有服务器上的应用的内存使用率。

这是非常伟大并且有用的，但是有限制性。如果你不想聚合一个给定名字的所有容器而是一个给定镜像的？或者你想把部署你的 canary 同在你生产环境的服务器对比？

可以为每个用例想出一个层次结构，但是没有一个支持它们。现实情况显示，你预先往往不知道哪个问题需要从新回答一次并且你开始研究。

Prometheus

Prometheus 支持维度数据，你可以拥有全局和简单的指标名像 container\_memory\_usage\_bytes ，使用多个维度来标识你服务的指定实例。

我已经创建了一个简单的 [***container-exporter***](https://github.com/docker-infra/container_exporter) 来收集 Docker 容器的指标以及输出给 Prometheus 来消费。这个输出器使用容器的名字，id 和 镜像作为维度。额外的 per-exporter 维度可以在 prometheus.conf 中设置。

如果你使用指标名字直接作为一个查询表达式，它将返回有这个使用这个指标名字作为标签的所有时间序列。

container\_memory\_usage\_bytes{env=**"prod"**,id=**"23f731ee29ae12fef1ef6726e2fce60e5e37342ee9e35cb47e3c7a24422f9e88"**,instance=**"http://1.2.3.4:9088/metrics"**,job=**"container-exporter"**,name=**"haproxy-exporter-int"**,image=**"prom/haproxy-exporter:latest"**} **11468800.000000**

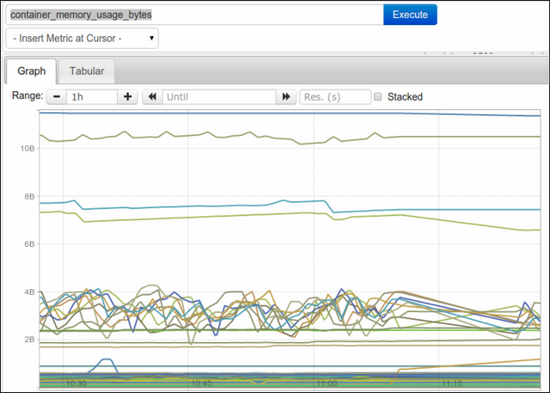
container\_memory\_usage\_bytes{env=**"prod"**,id=**"57690ddfd3bb954d59b2d9dcd7379b308fbe999bce057951aa3d45211c0b5f8c"**,instance=**"http://1.2.3.5:9088/metrics"**,job=**"container-exporter"**,name=**"haproxy-exporter"**,image=**"prom/haproxy-exporter:latest"**} **16809984.000000**

container\_memory\_usage\_bytes{env=**"prod"**,id=**"907ac267ebb3299af08a276e4ea6fd7bf3cb26632889d9394900adc832a302b4"**,instance=**"http://1.2.3.2:9088/metrics"**,job=**"container-exporter"**,name=**"node-exporter"**,image=**"prom/container-exporter:latest"**}

...

...

如果你运行了许多容器，这个看起来像这样

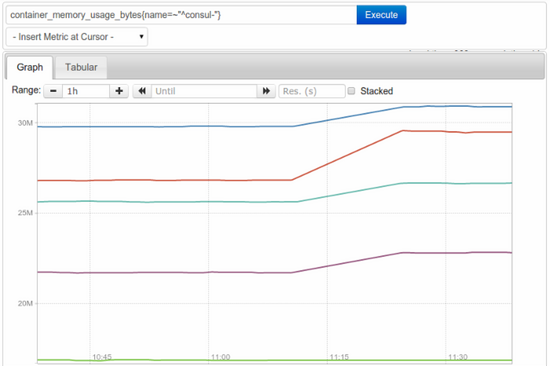


为了帮助你使得这数据更有意义，你可以过滤（filter） and/or 聚合（aggregate） 这些指标。

切片 & 切块（Slice & Dice）

使用 Prometheus 的查询语言，你可以对你想的任何维度的数据切片和切块。如果你对一个给定名字的所有容器感兴趣，你可以使用一个表达式像 container\_memory\_usage\_bytes{name="consul-server"} ，这个将仅仅显示 name == "consul-server" 的时间序列。

Prometheus 也支持正则表达式，因此匹配完整的脚本你可以这样做 container\_memory\_usage\_bytes{name=~"^consul"} ，这将展示起来像这样：

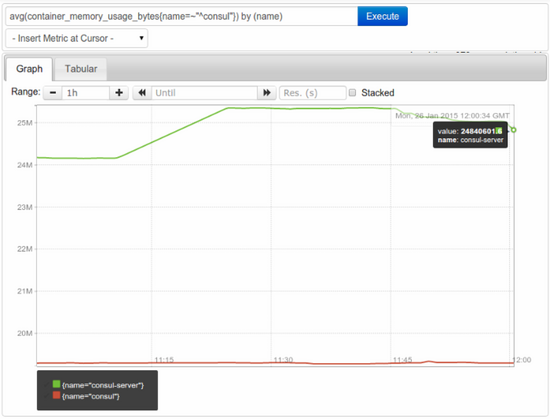


你也使用使用任何维度过滤，因此你可以获取在一个给定主机，给定环境和给定区域上所有容器的指标。

聚合（Aggregation）

和 Graphite 类似，Prometheus 支持聚合函数但是它的维度更加丰富。使用 sum(container\_memory\_usage\_bytes{name=~"^consul"}) 按预期汇总你所有 "consul-\*" 的内存使用率。

现在比如说你想看你的 'consul' 和 'consul-server' 容器平均内存使用率的不同，这可以通过提供维度保存这聚合结果像 avg(container\_memory\_usage\_bytes{name=~"^consul"}) by (name) 来实现：



如果你在多个区域有服务并且配置了区域名作为一个额外的标签对，你也可以保存维度来展示每个名字和区域的内存使用率，通过使用一个像这样的表达式 avg(container\_memory\_usage\_bytes{name=~"^consul"}) by (name,zone) 。

使用 Prometheus + Container-Exporter

正如你所知，我喜欢在容器中运行一切，包括 container-exporter 和 Prometheus，运行 container-exporter 应该是非常容易的：

docker **run -p 8080:8080 -v /sys/fs/cgroup:/cgroup \**

-v /var/**run/docker.sock:/var/run/docker.sock prom/container-exporter**

现在你需要安装 Prometheus。关于这个参考 [***官方文档***](http://prometheus.github.io/docs/introduction/install/) 。为了使得 Prometheus 从 container-exporter 拉取指标，你需要把它作为目标添加到配置。比如：

**job**: {

**name**: **"container-exporter"**

**scrape\_interval**: **"1m"**

**target\_group**: {

**labels**: {

**label**: {

**name**: **"zone"**

**value**: **"us-east-1"**

}

**label**: {

**name**: **"env"**

**value**: **"prod"**

}

}

**target**: **"http://1.2.3.4:8080/metrics"**

}

}

现在从新构建你的镜像如文档中描述的那样并启动它。Prometheus 现在应该每 60s 轮询你的 container-exporter。

总结

因为 Prometheus 的灵活性，它的性能和最小化依赖，它是我选择的监控系统。这就是为什么从去年起我介绍了 Prometheus 作为我们监控 Docker 的主要监控系统。

# 2@基于Prometheus做多维度的容器监控

http://m.blog.csdn.net/article/details?id=52714306

基于Prometheus做多维度的容器监控

发表于2016/10/1 0:18:57  2924人阅读

分类： docker

**什么是prometheus？**

prometheus从官方介绍来说，他是一个开源的系统监控和报警工具，最初由SoundCloud推出。自2012成立以来，许多公司和组织都采用了prometheus，项目有一个非常活跃的开发者和用户社区。它现在是一个独立的开源项目，并独立于任何公司。

**它具有以下特性：**

1. 多维度数据模型（由键/值对确定的时间序列数据模型）

2. 具有一个灵活的查询语言来利用这些维度

3. 不依赖分布式存储；单个服务器节点工作。

4. 时间序列的采集是通过HTTP pull的形式，解决很多push架构的问题。

5. 通过中介网关支持短时间序列数据的收集

6. 监控目标是通过服务发现或静态配置

7. 多种数据展示面板支持，例如grafana

**怎么使用prometheus监控容器**

prometheus监控不同的目标服务需要实现不同的exporter插件,早期的时候，官方出了container-exporter项目，但是现在项目已经停止。推荐使用谷歌的cAdvisor项目作为prometheus的exporter。cAdvisor作为一个监控单机容器的项目，数据较为全面，但是也有很大的问题，例如io等数据没有等等。结合prometheus后就能在整个集群监控查询容器。举个例子，你有一个项目有3个容器分布在三台机器，你怎么监控整个项目的流量，内存量，负载量的实时数据。这就是prometheus的多维度查询解决的问题，数据从3台机器的cadvisor得到每个容器的数据，它的多维度查询语法就能让你得到你想要的数据。

这里假设你有10台机器部署了容器需要监控，你在10台机器上分别部署cAdvisor容器

sudo docker run \

--volume=/:/rootfs:ro \

--volume=/var/run:/var/run:rw \

--volume=/sys:/sys:ro \

--volume=/var/lib/docker/:/var/lib/docker:ro \

--publish=8080:8080 \

--detach=true \

--name=cadvisor \

google/cadvisor:latest

找一台机器部署prometheus服务，这里依然使用容器部署：

docker run \

-p 9090:9090 \

--log-driver none \

-v /hdd1/prometheus/etc/:/etc/prometheus/ \

-v /hdd1/prometheus/data/:/prometheus/ \

-v /etc/localtime:/etc/localtime \

--name prometheus \

prom/prometheus

创建/hdd1/prometheus/etc/prometheus.yml配置文件

my global config

global:

scrape\_interval: 15s # By default, scrape targets every 15 seconds.

evaluation\_interval: 15s # By default, scrape targets every 15 seconds.

# scrape\_timeout is set to the global default (10s).

# Attach these labels to any time series or alerts when communicating with

# external systems (federation, remote storage, Alertmanager).

external\_labels:

monitor: 'container-monitor'

# Load and evaluate rules in this file every 'evaluation\_interval' seconds.

rule\_files:

- "/etc/prometheus/rules/common.rules"

# 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: 'container'

static\_configs:

- targets: ['10.12.1.129:9090','10.12.1.130:9090','10.50.1.92:9090','10.50.1.93:9090','10.50.1.119:9090']

配置文件中 -targets中的端点填写你的实际cadvisor所在的ip和暴露的端口.正确启动后访问ip:9090就能查询数据了哦。

prometheus缺点

1. 单机缺点，单机下存储量有限，根据你的监控量局限你的存储时间。

2. 内存占用率大，prometheus集成了leveldb，一个能高效插入数据的数据库，在ssd盘下io占用比较高。同时可能会有大量数据堆积内存。但是这是可以配置的。

prometheus适用于监控所有时间序列的项目

目前其生态中已经有很多exporter实现，例如：

Node/system metrics exporter

AWS CloudWatch exporter

Blackbox exporter

Collectd exporter

Consul exporter

Graphite exporter

HAProxy exporter

InfluxDB exporter

JMX exporter

Memcached exporter

Mesos task exporter

MySQL server exporter

SNMP exporter

StatsD exporter

你也可以根据你的需要自己实现exporter，完成你需要的监控任务。

本文来自：[http://blog.yiyun.pro](http://blog.yiyun.pro/)

# 3@用容器轻松搭建Prometheus运行环境 - 推酷

http://www.tuicool.com/articles/FNz2uuB

时间 2016-03-30 19:01:01  [懒程序员改变世界](http://www.tuicool.com/sites/Jbyaqae)

原文  [http://qinghua.github.io/prometheus/](http://qinghua.github.io/prometheus/?utm_source=tuicool&utm_medium=referral)

主题 [Grafana](http://www.tuicool.com/topics/11040110)[Docker](http://www.tuicool.com/topics/11090139)[Vagrant](http://www.tuicool.com/topics/11250044)

[***Prometheus***](https://prometheus.io/) 是一个开源的监控解决方案，包括数据采集、汇聚、存储、可视化、监控、告警等。除了基本的监控数据，也支持通过自定义exporter来获取自己想要的数据。本文从零开始用容器搭建一个prometheus环境，并介绍一些基本功能。

准备工作

我们需要先安装 [***virtualBox***](https://www.virtualbox.org/wiki/Downloads) 和 [***vagrant***](https://www.vagrantup.com/downloads.html) 。通过vagrant来驱动virtualBox搭建一个虚拟测试环境。首先在本地任意路径新建一个空文件夹比如 test ，运行以下命令：

virtual box host

**mkdir** test  
cd test  
vagrant init minimum/ubuntu-trusty64-docker  
vi Vagrantfile

里面应该有一句 config.vm.box = "minimum/ubuntu-trusty64-docker" ，在它的下面添加如下代码，相当于给它分配一台IP是 **192.168.33.18** 的虚拟机。

Vagrantfile

**config**.vm.network "**private\_network**", **ip**: "192.168.33.18"

这个vagrant镜像已经在ubuntu的基础上帮我们安装了docker，用起来很方便。然后在终端运行以下命令启动并连接虚拟机。

virtual box host

**vagrant** up  
vagrant ssh

搭建环境

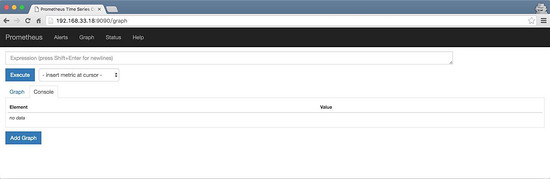
Prometheus的环境搭建起来非常简单，只要一个docker镜像即可。绿色的压缩包安装方式可以参考 [***官方文档***](https://prometheus.io/docs/introduction/getting_started/) 。此外还需要一个配置文件：

cat << EOF >prometheus.yml  
global:  
 scrape\_interval: 15s  
 external\_labels:  
 monitor: 'codelab-monitor'  
scrape\_configs:  
 - job\_name: 'prometheus'  
 scrape\_interval: 5s  
 target\_groups:  
 - targets: ['localhost:9090']  
EOF  
  
sudo mkdir /etc/prometheus  
sudo mv prometheus.yml /etc/prometheus

配置文件中， scrape\_interval 指的是数据获取间隔， prometheus 这个任务里的 scrape\_interval 将会在这个任务里覆盖掉默认的 global 全局值，也就是这个任务每5秒钟获取一次数据，其它任务则是每15秒钟。完整的配置文件格式，请参考 [***官方文档***](http://prometheus.io/docs/operating/configuration/) 。接下来启动Prometheus：

/usr/bin/docker run -d \  
 --name=prometheus \  
 --publish=9090:9090 \  
 -v /etc/prometheus/prometheus.yml:/etc/prometheus/prometheus.yml \  
 -v /var/prometheus/storage:/prometheus \  
 prom/prometheus:0.17.0

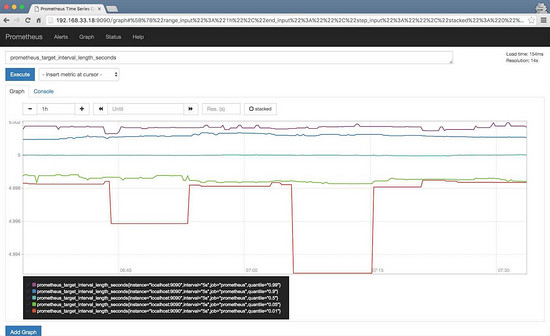
启动完成后，将会在 [***http://192.168.33.18:9090***](http://192.168.33.18:9090/) 看到prometheus的首页：



数据收集

在 [***http://192.168.33.18:9090/metrics***](http://192.168.33.18:9090/metrics) 可以看到prometheus收集到的数据。其中有一个 prometheus\_target\_interval\_length\_seconds ，表示真实的数据获取间隔。在prometheus首页输入它并回车，就可以看到一系列的数据，它们有着不同的quantile，从0.01至0.99不等。0.99的意思是有99%的数据都在这个值以内。如果我们只关心这个数，我们可以输入 prometheus\_target\_interval\_length\_seconds{quantile="0.99"} 来查看。查询还支持函数，比如 count(prometheus\_target\_interval\_length\_seconds) 可以查询数量。完整的表达式可以参考 [***官方文档***](https://prometheus.io/docs/querying/basics/) 。

点击 **Console** 旁边的 **Graph** 标签就可以看见时序图了：



可以随意选择指标和函数试一试，比如 rate(prometheus\_local\_storage\_chunk\_ops\_total[1m]) 。

Exporter

Prometheus支持官方/非官方的许多种 [***exporter***](https://prometheus.io/docs/instrumenting/exporters/) ，如HAProxy，Jenkins，MySQL等，也有一些软件直接支持Prometheus而无需exporter，如Etcd，Kubernetes等。我们试一下node exporter：

docker **run** -d \  
 --name=ne \  
 -p 9100:9100 \  
 prom/node-exporter

Node exporter暴露的端口是9100，所以我们需要修改一下prometheus的配置文件：

cat << EOF >prometheus.yml  
global:  
 scrape\_interval: 15s  
 external\_labels:  
 monitor: 'codelab-monitor'  
scrape\_configs:  
 - job\_name: 'node'  
 scrape\_interval: 5s  
 target\_groups:  
 - targets: ['192.168.33.18:9100']  
EOF  
  
sudo cp prometheus.yml /etc/prometheus

重启prometheus：

docker **stop** prometheus  
sudo rm -rf /**var**/prometheus/storage  
docker **start** prometheus

这样在页面上就可以选择节点的一些指标了。也可以访问 [***http://192.168.33.18:9100/***](http://192.168.33.18:9100/) 来直接查看Exporter的指标。

Push Gateway

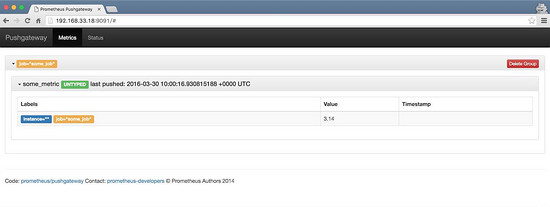
Prometheus采集数据是用的pull也就是拉模型，这从我们刚才设置的5秒参数就能看出来。但是有些数据并不适合采用这样的方式，对这样的数据可以使用Push Gateway服务。它就相当于一个缓存，当数据采集完成之后，就上传到这里，由Prometheus稍后再pull过来。我们来试一下，首先启动Push Gateway：

docker **run** -d \  
 --name=pg \  
 -p 9091:9091 \  
 prom/pushgateway

可以访问 [***http://192.168.33.18:9091/***](http://192.168.33.18:9091/) 来查看它的页面。下个命令将会往Push Gateway上传数据：

echo "some\_metric 3.14" | curl --data-binary @- http://192.168.33.18:9091/metrics/job/some\_job

效果是酱紫滴：



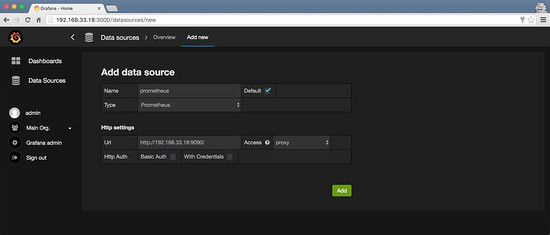
而在Prometheus的配置文件里，只要把端口换成 9100 便能采集到Push Gateway的数据了。

Grafana

[***Grafana***](https://prometheus.io/docs/visualization/grafana/) 是目前比较流行的监控可视化UI，它从2.5.0版开始直接支持Prometheus的数据。我们来试一下。首先启动grafana：

docker **run** -d \  
 --name grafana \  
 -p 3000:3000 \  
 grafana/grafana:2.6.0

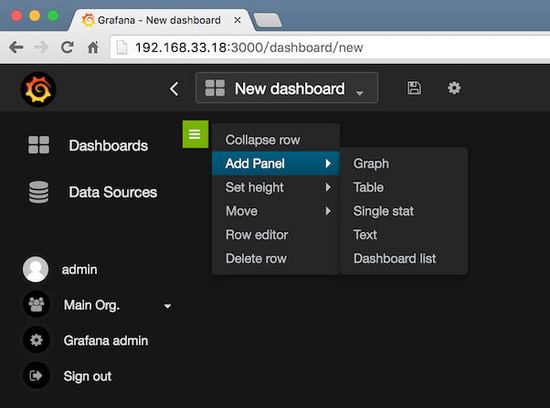
打开 [***http://192.168.33.18:3000/***](http://192.168.33.18:3000/) ，就能看到grafana的登录页面了。输入默认的admin/admin登录grafana。选择左侧的 **Data Sources** ，然后点击上面的 **Add new** 按钮，便可以把prometheus作为数据源导入grafana：



输入下面的值：

* Name：prometheus
* Default：true
* Type：Prometheus
* Url： [***http://192.168.33.18:9090/***](http://192.168.33.18:9090/)

然后点击 **Add** 按钮。之后会出来一个 **Test Connection** 的按钮，点击它便可以收到 **Data source is working** 的消息。点击左边的 **Dashboards** 回到主页，点击上面的 **Home** ，选择 **+ New** ，会出来一个绿色的小竖条，点击它便会弹出来一个菜单：



选择 **Add Panel** 和 **Graph** ，便会出来一个图。然后就可以在 **Query** 里输入prometheus支持的查询了：

