# 2@https://hub.docker.com/\_/telegraf/

https://hub.docker.com/\_/telegraf/

docker pull telegraf

docker pull registry.alauda.cn/library/telegraf

Short Description

Telegraf is an agent for collecting metrics and writing them to InfluxDB or other outputs.

Full Description

**Supported tags and respective Dockerfile links**

* [1.1, 1.1.2 (telegraf/1.1/Dockerfile)](https://github.com/influxdata/influxdata-docker/blob/d3babf9a87cb217d99aeabcea369c917b81121e1/telegraf/1.1/Dockerfile)
* [1.1-alpine, 1.1.2-alpine (telegraf/1.1/alpine/Dockerfile)](https://github.com/influxdata/influxdata-docker/blob/d3babf9a87cb217d99aeabcea369c917b81121e1/telegraf/1.1/alpine/Dockerfile)
* [1.2, 1.2.1, latest (telegraf/1.2/Dockerfile)](https://github.com/influxdata/influxdata-docker/blob/d3babf9a87cb217d99aeabcea369c917b81121e1/telegraf/1.2/Dockerfile)
* [1.2-alpine, 1.2.1-alpine, alpine (telegraf/1.2/alpine/Dockerfile)](https://github.com/influxdata/influxdata-docker/blob/d3babf9a87cb217d99aeabcea369c917b81121e1/telegraf/1.2/alpine/Dockerfile)

For detailed information about the published artifacts of each of the above supported tags (image metadata, transfer size, etc), please see [the repos/telegraf directory](https://github.com/docker-library/repo-info/blob/master/repos/telegraf) in [the docker-library/repo-info GitHub repo](https://github.com/docker-library/repo-info).

For more information about this image and its history, please see [the relevant manifest file (library/telegraf)](https://github.com/docker-library/official-images/blob/master/library/telegraf). This image is updated via [pull requests to the docker-library/official-imagesGitHub repo](https://github.com/docker-library/official-images/pulls?q=label%3Alibrary%2Ftelegraf).

**Telegraf**

Telegraf is an open source agent written in Go for collecting metrics and data on the system it's running on or from other services. Telegraf writes data it collects to InfluxDB in the correct format.

[Telegraf Official Docs](https://docs.influxdata.com/telegraf/latest/introduction/getting-started-telegraf/)

https://raw.githubusercontent.com/docker-library/docs/43d87118415bb75d7bb107683e79cd6d69186f67/telegraf/logo.png

**Using this image**

**Exposed Ports**

* 8125 StatsD
* 8092 UDP
* 8094 TCP

**Using the default configuration**

The default configuration requires a running InfluxDB instance as an output plugin. Ensure that InfluxDB is running on port 8086 before starting the Telegraf container.

Minimal example to start an InfluxDB container:

$ docker run -d --name influxdb -p 8083:8083 -p 8086:8086 influxdb

Starting Telegraf using the default config, which connects to InfluxDB at http://localhost:8086/:

$ docker run --net=container:influxdb telegraf

**Using a custom config file**

First, generate a sample configuration and save it as telegraf.conf on the host:

$ docker run --rm telegraf -sample-config > telegraf.conf

Once you've customized telegraf.conf, you can run the Telegraf container with it mounted in the expected location:

$ docker run -v $PWD/telegraf.conf:/etc/telegraf/telegraf.conf:ro telegraf

Modify $PWD to the directory where you want to store the configuration file.

Read more about the Telegraf configuration [here](https://docs.influxdata.com/telegraf/latest/introduction/configuration/).

**Using the container with input plugins**

These examples assume you are using a custom configuration file that takes advantage of Docker's built-in service discovery capability. In order to do so, we'll first create a new network:

$ docker network create influxdb

Next, we'll start our InfluxDB container named influxdb:

$ docker run -d --name=influxdb \

--net=influxdb \

influxdb

The telegraf.conf configuration can now resolve the influxdb container by name:

[[outputs.influxdb]]

urls = ["http://influxdb:8086"]

Finally, we start our Telegraf container and verify functionality:

$ docker run -d --name=telegraf \

--net=influxdb \

-v $PWD/telegraf.conf:/etc/telegraf/telegraf.conf:ro \

telegraf

$ docker logs -f telegraf

**Aerospike**

Start an instance of aerospike:

$ docker run -d --name aerospike \

--net=influxdb \

-p 3000-3003:3000-3003 \

aerospike

Edit your Telegraf config file and set the correct connection parameter for Aerospike:

[[inputs.aerospike]]

servers = ["aerospike:3000"]

Restart your telegraf container to pick up the changes:

$ docker restart telegraf

**Nginx**

Create an nginx\_status.conf configuration file to expose metric data:

server {

listen 8090;

location /nginx\_status {

stub\_status on;

access\_log on;

}

}

Start an Nginx container utilizing it:

$ docker run -d --name=nginx \

--net=influxdb \

-p 8090:8090 -p 8080:80 \

-v $PWD/nginx\_status.conf:/etc/nginx/conf.d/nginx\_status.conf:ro \

nginx

Verify the status page: <http://localhost:8090/nginx_status>.

Configure the nginx input plugin in your Telegraf configuration file:

[[inputs.nginx]]

urls = ["http://nginx:8090/nginx\_status"]

Restart your telegraf container to pick up the changes:

$ docker restart telegraf

**StatsD**

Telegraf has a StatsD plugin, allowing Telegraf to run as a StatsD server that metrics can be sent to. In order for this to work, you must first configure the [StatsD plugin](https://github.com/influxdata/telegraf/tree/master/plugins/inputs/statsd) in your config file.

Run Telegraf with the UDP port 8125 exposed:

$ docker run -d --name=telegraf \

--net=influxdb \

-p 8125:8125/udp \

-v $PWD/telegraf.conf:/etc/telegraf/telegraf.conf:ro \

telegraf

Send Mock StatsD data:

$ for i in {1..50}; do echo $i;echo "foo:1|c" | nc -u -w0 127.0.0.1 8125; done

Check that the measurement foo is added in the DB.

**Supported Plugins Reference**

* [Input Plugins](https://docs.influxdata.com/telegraf/latest/inputs/)
* [Output Plugins](https://docs.influxdata.com/telegraf/latest/outputs/)

**Image Variants**

The telegraf images come in many flavors, each designed for a specific use case.

**telegraf:<version>**

This is the defacto image. If you are unsure about what your needs are, you probably want to use this one. It is designed to be used both as a throw away container (mount your source code and start the container to start your app), as well as the base to build other images off of. This tag is based off of [buildpack-deps](https://registry.hub.docker.com/_/buildpack-deps/). buildpack-deps is designed for the average user of docker who has many images on their system. It, by design, has a large number of extremely common Debian packages. This reduces the number of packages that images that derive from it need to install, thus reducing the overall size of all images on your system.

**telegraf:alpine**

This image is based on the popular [Alpine Linux project](http://alpinelinux.org/), available in [the alpine official image](https://hub.docker.com/_/alpine). Alpine Linux is much smaller than most distribution base images (~5MB), and thus leads to much slimmer images in general.

This variant is highly recommended when final image size being as small as possible is desired. The main caveat to note is that it does use [musl libc](http://www.musl-libc.org/) instead of [glibc and friends](http://www.etalabs.net/compare_libcs.html), so certain software might run into issues depending on the depth of their libc requirements. However, most software doesn't have an issue with this, so this variant is usually a very safe choice. See [this Hacker News comment thread](https://news.ycombinator.com/item?id=10782897) for more discussion of the issues that might arise and some pro/con comparisons of using Alpine-based images.

To minimize image size, it's uncommon for additional related tools (such as git or bash) to be included in Alpine-based images. Using this image as a base, add the things you need in your own Dockerfile (see the[alpine image description](https://hub.docker.com/_/alpine/) for examples of how to install packages if you are unfamiliar).

**License**

View [license information](https://github.com/influxdata/telegraf/blob/master/LICENSE) for the software contained in this image.

**Supported Docker versions**

This image is officially supported on Docker version 17.04.0-ce.

Support for older versions (down to 1.6) is provided on a best-effort basis.

Please see [the Docker installation documentation](https://docs.docker.com/installation/) for details on how to upgrade your Docker daemon.

**User Feedback**

**Issues**

If you have any problems with or questions about this image, please contact us through a [GitHub issue](https://github.com/influxdata/influxdata-docker/issues). If the issue is related to a CVE, please check for [a cve-tracker issue on the official-images repository first](https://github.com/docker-library/official-images/issues?q=label%3Acve-tracker).

You can also reach many of the official image maintainers via the #docker-library IRC channel on [Freenode](https://freenode.net/).

**Contributing**

You are invited to contribute new features, fixes, or updates, large or small; we are always thrilled to receive pull requests, and do our best to process them as fast as we can.

Before you start to code, we recommend discussing your plans through a [GitHub issue](https://github.com/influxdata/influxdata-docker/issues), especially for more ambitious contributions. This gives other contributors a chance to point you in the right direction, give you feedback on your design, and help you find out if someone else is working on the same thing.

**Documentation**

Documentation for this image is stored in the [telegraf/ directory](https://github.com/docker-library/docs/tree/master/telegraf) of the [docker-library/docs GitHub repo](https://github.com/docker-library/docs). Be sure to familiarize yourself with the [repository's README.md file](https://github.com/docker-library/docs/blob/master/README.md) before attempting a pull request.

# samuelebistoletti/docker-statsd-influxdb-grafana

Docker Image with Telegraf (StatsD), InfluxDB and Grafana

https://github.com/samuelebistoletti/docker-statsd-influxdb-grafana

Docker Image with Telegraf (StatsD), InfluxDB and Grafana

 Battle-tested

Versions

Warning, breaking change: upgrade from version 1.0.x of this image is not supported, all persisted data in volumes will be lost if you delete the container.

* Docker Image: 2.0.0
* Ubuntu: 16.04
* InfluxDB: 1.2
* Telegraf (StatsD): 1.2
* Grafana: 4.1.1

Quick Start

To start the container the first time launch:

docker run -d \

--name docker-statsd-influxdb-grafana \

-p 3003:3003 \

-p 3004:8083 \

-p 8086:8086 \

-p 22022:22 \

-p 8125:8125/udp \

samuelebistoletti/docker-statsd-influxdb-grafana:latest

You can replace latest with the desired version listed in changelog file.

To stop the container launch:

docker stop docker-statsd-influxdb-grafana

To start the container again launch:

docker start docker-statsd-influxdb-grafana

Mapped Ports

Host Container Service

3003 3003 grafana

3004 8083 influxdb-admin

8086 8086 influxdb

8125 8125 statsd

22022 22 sshd

SSH

ssh root@localhost -p 22022

Password: root

Grafana

Open [http://localhost:3003](http://localhost:3003/)

Username: root

Password: root

Add data source on Grafana

1. Using the wizard click on Add data source
2. Choose a name for the source and flag it as Default
3. Choose InfluxDB as type
4. Choose direct as access
5. Fill remaining fields as follows and click on Add without altering other fields

Url: http://localhost:8086

Database: telegraf

User: telegraf

Password: telegraf

Basic auth and credentials must be left unflagged. Proxy is not required.

Now you are ready to add your first dashboard and launch some query on database.

InfluxDB

Web Interface

Open [http://localhost:3004](http://localhost:3004/)

Username: root

Password: root

Port: 8086

InfluxDB Shell (CLI)

1. Establish a ssh connection with the container
2. Launch influx to open InfluxDB Shell (CLI)

# Docker监控方案(TIG)的研究与实践之Telegraf

- Andy-xu的个人空间 https://my.oschina.net/xxbAndy/blog/751330

*摘要: Docker监控方案之容器内部指标采集工具Telegraf的介绍和安装。Telegraf用纯go编写，通过插件化方式进行采集各种服务(system，docker，redis，nginx，kafka等)监控指标并且上报给相应的中间件，比如influxdb，opentsdb(商城docker监控使用这个)。Telegraf也是整个TICK(telegraf+influxdb+chronograf+kapacitor)生态栈的第一块组件也是最重要的组件。*

**前言**

Docker由于使用了基于namespace和cgroup的技术，因此监控docker容器和监控宿主机在某些性能指标和方式上有一些区别，而传统的监控方式可能无法满足docker容器内部的指标监控，本篇系列文章主要分享使用telegraf+influxdb+grafana去监控docker容器内部资源使用情况。目前主要关注的监控指标为：每个宿主机上的docker容器数量，每个docker容器的内存使用情况，CPU使用情况，网络使用情况以及磁盘使用情况。同时这套方案也能够监控到宿主机的一些基本资源使用情况。

**Telegraf简介与实践**

简介：

由influxdata公司开发的用于采集系统数据的服务，用纯go编写，通过插件化方式进行采集各种服务(system，docker，redis，nginx，kafka等)监控指标并且上报给相应的中间件，比如influxdb，opentsdb(商城docker监控使用这个)。Telegraf也是整个TICK(telegraf+influxdb+chronograf+kapacitor)生态栈的第一块组件也是最重要的组件。

特点：

纯go编写，不需要依赖其他组件；消耗相关系统资源比较小；plugins支持多种输入输出插件(采集和上报)；   
相关连接：

github：<https://github.com/influxdata/telegraf>  
官网文档：<https://docs.influxdata.com/telegraf/v1.0/>  
TICK生态栈:<https://www.influxdata.com/downloads/#telegraf>

安装：

所有的安装以及部署都是在linux下的，所以不知道linux下安装基础软件包的，请自觉绕路！   
Centos系列可以配置yum源或者直接下载包，并安装。个人建议直接下载包，由于不需要其他系统依赖，可以直接在集群环境进行共享。   
wget<https://dl.influxdata.com/telegraf/releases/telegraf-1.0.0.x86_64.rpm> && rpm -ivh telegraf-1.0.0.x86\_64.rpm   
其他环境安装指南:

Ubuntu && Debin：   
ubuntu repo：   
curl -sL<https://repos.influxdata.com/influxdb.key> | sudo apt-key add -   
source /etc/lsb-release   
echo “deb [https://repos.influxdata.com/](https://repos.influxdata.com/%3Cspan%20class='mathjax-replacement'%20rel='227521499c481b5de95e53b2237f2e80'%3E$%7BDISTRIB_ID,,%7D%20$%3C/span%3E){DISTRIB\_CODENAME} stable” | sudo tee /etc/apt/sources.list.d/influxdb.list   
Debin repo：   
curl -sL<https://repos.influxdata.com/influxdb.key> | sudo apt-key add -   
source /etc/os-release   
test $VERSION\_ID = “7” && echo “deb<https://repos.influxdata.com/debian>wheezy stable” | sudo tee /etc/apt/sources.list.d/influxdb.list   
test $VERSION\_ID = “8” && echo “deb<https://repos.influxdata.com/debian> jessie stable” | sudo tee /etc/apt/sources.list.d/influxdb.list   
配置完ubuntu系列的repo之后，就可以执行sudo apt-get update && sudo apt-get install telegraf进行安装了   
直接下载deb包方式：   
wget<https://dl.influxdata.com/telegraf/releases/telegraf_1.0.0_amd64.deb>&& sudo dpkg -i telegraf\_1.0.0\_amd64.deb

启动：

telegraf可以支持多种服务管理方式，安装之后默认可用使用service和systemd进行管理，因此在centos6-7中都可以使用系统自带的服务管理进行维护(init.d和systemctl)

/etc/init.d/telegraf start 或者systemctl restart telegraf

配置：

配置可以说是telegraf运用中最核心的一个环节，因为配置的细节决定你采集数据的指标。telegraf的配置可以说是比较千变万化，因为可以支持多种输出、输入组件，并且每种组件的配置支持不通的过滤规则，能够让配置管理和维护者正确的采集自己需要的信息。   
默认配置文件存放路径：/etc/telegraf/telegraf.conf ，额外配置路径/etc/telegraf/telegraf.d/。   
在生产环境中建议自定生成配置并存放在/etc/telegraf/telegraf.d/中。   
自定义生成配置文件：

#telegraf -sample-config > telegraf.conf 这样生成的配置文件将包含每一个插件，但是大部分会被注释掉，可以根据实际的业务场景进行定义

配置文件示例以及详细讲解：

#cat telegraf.conf

########################################全局配置############################################################

#全局tag配置，采用key = "values"方式，这样在本机采集到的所有数据将都有这个标签

[global\_tags]

dc = "docker-test"

#agent配置

[agent]

#默认的数据(input)采集间隔时间

interval = "10s"

#采用轮询时间间隔。默认是使用interval里面的值进行轮询，比如interval = "10s",那采集时间将是:00, :10, :20, 等

round\_interval = true

#每次发送到output的度量大小不能超过metric\_batch\_size的值

metric\_batch\_size = 1000

#telegraf会为每一个output去缓存一份度量值，metric\_buffer\_limit为缓存的限制，并且刷新buffer以确定成功写入。如果达到这个限制了，老的数据会被第一时间丢弃

#当然了，增加这个值能够容忍更多的数据连接，但是这也将会增加telegraf潜在的内存占用。这个值可以大于metric\_batch\_size但是必须小于它的两倍

metric\_buffer\_limit = 10000

#通过随机度量来对采集时间进行抖动。每个插件在采集数据之前将会有一个随机时间的休眠，但是这个时间应小于collection\_jitter

#这个设置是为了防止多个采集源数据同一时间都在队列

collection\_jitter = "0s"

#默认所有数据flush到outputs的时间(在数据被flush到output之前，最大能到flush\_interval + flush\_jitter)。不能低于interval

flush\_interval = "10s"

# 通过随机数来对flush间隔进行抖动。这个主要是为了避免当运行一个大的telegraf实例的时候有比较大的写入。(jitter=5s,flush\_interval=10s意味着每10-15s会发生一次flush操作)

flush\_jitter = "0s"

#默认这个值被设置相同的时间戳通过采集间隔排序。最大值为1s。这个指标一般不会用在service input(比如logparser和statsd)。单位(ns,us,ms,s)

precision = ""

#以debug模式运行

debug = false

#以安静模式运行

quiet = false

#这个将会覆盖默认的hostname，如果为空的话，将会采用os.Hostname()

hostname = ""

#如果设置为true，就不允许在telegraf agent里面设置"host"标签了

omit\_hostname = false

##############################################度量值过滤#######################################################

#过滤可以被配置在每一个输入和输出值

namepass：一个数组字符串可以被用来过滤由当前input生成的度量值，在数组中的每一个字符串和全局匹配到的测量值名字进行对比，如果匹配了，值被采用

namedrop:pass的反向含义，如果匹配，则不使用

fieldpass：在namepass满足的条件下，output的fieldpass不可用

fielddrop：pass的反向含义，如果field名字匹配，将不被采用。output的fielddrop不可用

tagpass：tag names和数组中的字符串都被用来过滤当前input的值，数组中的每一个每一个字符串和tag name对比，匹配则则采用

tagdrop:tagpass的反向含义，如果tag匹配，该度量值不被采用

tagexclude:被用来从度量值(measurements)中执行一个tag。作为tagdrop的对立面，它将丢弃所有依赖于tag的相关度量值，tagexclude只是单纯的从度量值中给tag一个key

这个可以被用作input和output中，但是强烈建议用在input中，他会在同一个采集时间点更加有效的过滤out tags

taginclude:tagexclude的反向含义。在最终的度量值中，也将包含tag keys

注意：tagpass和tagdrop参数必须等一在plugin函数的底部，不然对应的子plugin配置可能被tagpass/tagdrop映射中的内容截断

#################################################(OUTPUT)输出配置##############################################

#输出插件，我们使用的是influxdb，得先进行安装配置

[[outputs.influxdb]]

## The full HTTP or UDP endpoint URL for your InfluxDB instance.

#如果有多个urls，可以指定为相同集群的一部分。意味着urls中的一个将被写到每一个间隔

# urls = ["udp://localhost:8089"] # UDP endpoint example

urls = ["http://172.25.46.7:8086"] # required

#默认需要连接的telegraf库，没有则自己创建

database = "telegraf" # required

precision = "s"

#修改保留策略

retention\_policy = ""

#持续写入，仅支持集群模式, can be: "any", "one", "quorum", "all"

write\_consistency = "any"

#作为influxdb客户端，设置写超时时间，如果为空默认为5s超时，0s表示不设置超时时间(不建议)

timeout = "5s"

#设置telegraf的库的用户名和密码

# username = "telegraf"

# password = "metricsmetricsmetricsmetrics"

###############################################(INPUT)输入配置###################################################

inputs插件全局参数:

#每一个input都可以配置的全局配置项

#name\_override:覆盖默认的度量值名字(默认是input的名字)

#name\_prefix:指定一个前缀并附加到度量值的名字(measuerments name)

#name\_suffix:指定后缀

#tags：一个标签映射到指定的input度量值

#interval：多久采集一次数据，默认可用使用全局配置中的参数

配置示例：

[[inputs.cpu]]

#采集每个cpu的指标

percpu = true

#采集总的cpu指标

totalcpu = true

#会丢弃掉time开头的。如果想要采集原始的cpu相关指标，请注释

fielddrop = ["time\_\*"]

[[inputs.disk]]

#默认的telegraf将手机所有挂载点的信息

#下面这个参数可以指定挂载点

mount\_points = ["/"]

#仅存储磁盘inode相关的度量值

fieldpass = ["inodes\*"]

#通过文件系统类型来忽略一些挂载点，比如tmpfs

ignore\_fs = ["tmpfs", "devtmpfs"]

#仅存储tagpass相关的信息

[inputs.disk.tagpass]

fstype = [ "ext4", "xfs" ]

path = [ "/export", "/home\*" ]

#默认telegraf将采集所有存储设备的信息，devices参数可以指定

# devices = ["sda", "sdb"]

#如果需要磁盘的串行号可以将下面注释打开

# skip\_serial\_number = false

[[inputs.mem]]

#采集docker和redis的插件

[[inputs.docker]]

#指定docker启动的api接口，并指定需要采集那些容器指标

endpoint = "tcp://10.0.0.2:5256"

container\_names = []

[[inputs.redis]]

#指定redis的相关接口

servers = ["tcp://10.0.0.1:6379"]

测试插件是否正常工作：

使用以下命令会将telegraf采集的数据默认输出到终端，依次来检验配置的监控项是否是自己所期望的指标。

#telegraf -config /etc/telegraf/telegraf.conf -input-filter docker -test 会输出docker相关的监控信息说明配置正确(当然也可以去测试其他inputs plugins)

**注意：上面的配置文件中使用的output plugins是influxdb，因此在没有成功配置influxdb的前提下，此配置文件是不能正常让telegraf正常启动的！下一节将会讲到influxdb的相关知识influxdb相关：**

重启服务：

centos6.x:

#/etc/init.d/telegraf restart （service telegraf restart ）

centos7.x:

#systemctl restart telegraf

此时，可以查看相关日志，确保telegraf正常启动，启动之后去influxdb就可以查询相关采集到的数据。

附：

telegraf常用的input plugins:

收集docker相关的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/docker>主要是通过docker API调用相关监控   
收集相关redis的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/redis>  
收集相关mesos的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mesos>  
收集相关nginx的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/nginx>  
收集相关mysql的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mysql>  
收集ping相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/ping>  
收集influxdb相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/influxdb>  
收集系统相关的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/system>  
收集haproxy相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/haproxy>  
收集cgroup相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/cgroup>

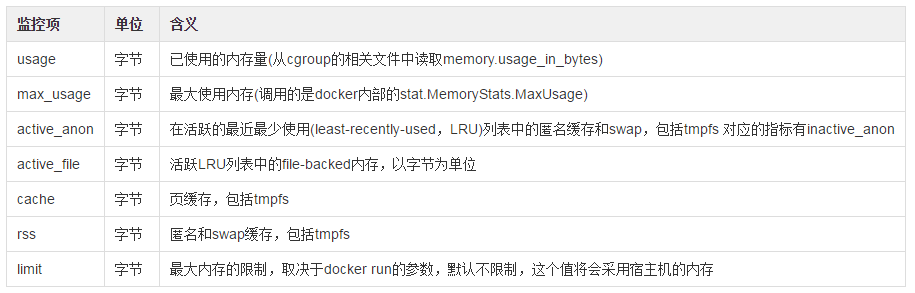
核心plugins 监控指标的采集原理(system，docker)

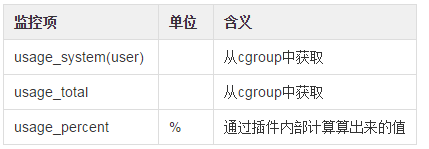
* **system plugin：**主要监控项包含CPU,DISK,KERNEL,KERNEL\_VMSTAT,NETSTAT,PROCESS,SYSTEM

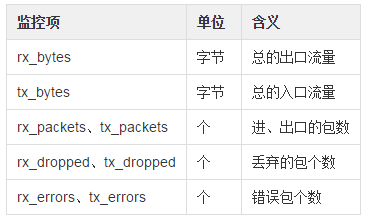
CPU中有两个参数，分别为totalcpu和percpu，如果为true经分别采集相关cpu的指标。主要指标有：user，nice，system，idle，iowait，cpu\_usage等   
DISK：主要指标有free，total；used(单位字节);uesd\_percent;inode\_free;inode\_total;inode\_used.需要注意的是，used\_percent指标通过使用used/(used+free)计算得出。   
MEM：主要指标total;available(/proc/meminfo原生值)；available\_percent(available / total \* 100)；used\_percent(used / total \* 100)   
NET：通过lsof采集tcp连接状态和udp相关信息。指标：established syn\_sent syn\_recv fin\_wait1 time\_wait close listen closing   
PROCESS：收集进程总数个状态组（zombie,sleeping,running),也是通过采集/proc中的数据   
SYSTEM：系统负载,load1;load15;load5

* **docker plugin:**主要监控项包括ocker\_container\_mem,docker\_container\_cpu,docker\_container\_net,docker\_container\_nlkio,docker\_,docker\_data,docker\_metadata。基本上是通过docker api进行采集docker容器相关的监控指标的（<https://docs.docker.com/engine/reference/api/docker_remote_api_v1.25/#/inspect-a-container>）   
  具体的监控项可以在源码中进行查看：(<https://github.com/influxdata/telegraf/blob/master/plugins/inputs/docker/docker.go>)
* **几个主要关心的指标:**

docker\_memory:

  
docker\_cpu:

  
docker\_net:

  
docker相关:



注意：原创著作，转载请联系作者！

# 2@telegraf/plugins/inputs/Jolokia

at master · influxdata/telegraf

https://github.com/influxdata/telegraf/tree/master/plugins/inputs/jolokia

Telegraf plugin: Jolokia

Configuration

# Read JMX metrics through Jolokia

[[inputs.jolokia]]

## This is the context root used to compose the jolokia url

## NOTE that Jolokia requires a trailing slash at the end of the context root

context = "/jolokia/"

## This specifies the mode used

# mode = "proxy"

#

## When in proxy mode this section is used to specify further

## proxy address configurations.

## Remember to change host address to fit your environment.

# [inputs.jolokia.proxy]

# host = "127.0.0.1"

# port = "8080"

## Optional http timeouts

##

## response\_header\_timeout, if non-zero, specifies the amount of time to wait

## for a server's response headers after fully writing the request.

# response\_header\_timeout = "3s"

##

## client\_timeout specifies a time limit for requests made by this client.

## Includes connection time, any redirects, and reading the response body.

# client\_timeout = "4s"

## List of servers exposing jolokia read service

[[inputs.jolokia.servers]]

name = "as-server-01"

host = "127.0.0.1"

port = "8080"

# username = "myuser"

# password = "mypassword"

## List of metrics collected on above servers

## Each metric consists in a name, a jmx path and either

## a pass or drop slice attribute.

## This collect all heap memory usage metrics.

[[inputs.jolokia.metrics]]

name = "heap\_memory\_usage"

mbean = "java.lang:type=Memory"

attribute = "HeapMemoryUsage"

## This collect thread counts metrics.

[[inputs.jolokia.metrics]]

name = "thread\_count"

mbean = "java.lang:type=Threading"

attribute = "TotalStartedThreadCount,ThreadCount,DaemonThreadCount,PeakThreadCount"

## This collect number of class loaded/unloaded counts metrics.

[[inputs.jolokia.metrics]]

name = "class\_count"

mbean = "java.lang:type=ClassLoading"

attribute = "LoadedClassCount,UnloadedClassCount,TotalLoadedClassCount"

Description

The Jolokia plugin collects JVM metrics exposed as MBean's attributes through jolokia REST endpoint. All metrics are collected for each server configured.

See: <https://jolokia.org/>

Measurements:

Jolokia plugin produces one measure for each metric configured, adding Server's jolokia\_name, jolokia\_host andjolokia\_port as tags.

# 2@telegraf/plugins/outputs/influxdb

at master · influxdata/telegraf

https://github.com/influxdata/telegraf/tree/master/plugins/outputs/influxdb

InfluxDB Output Plugin

This plugin writes to [InfluxDB](https://www.influxdb.com/) via HTTP or UDP.

Configuration:

# Configuration for influxdb server to send metrics to

[[outputs.influxdb]]

## The HTTP or UDP URL for your InfluxDB instance. Each item should be

## of the form:

## scheme "://" host [ ":" port]

##

## Multiple urls can be specified as part of the same cluster,

## this means that only ONE of the urls will be written to each interval.

# urls = ["udp://localhost:8089"] # UDP endpoint example

urls = ["http://localhost:8086"] # required

## The target database for metrics (telegraf will create it if not exists).

database = "telegraf" # required

## Retention policy to write to. Empty string writes to the default rp.

retention\_policy = ""

## Write consistency (clusters only), can be: "any", "one", "quorum", "all"

write\_consistency = "any"

## Write timeout (for the InfluxDB client), formatted as a string.

## If not provided, will default to 5s. 0s means no timeout (not recommended).

timeout = "5s"

# username = "telegraf"

# password = "metricsmetricsmetricsmetrics"

## Set the user agent for HTTP POSTs (can be useful for log differentiation)

# user\_agent = "telegraf"

## Set UDP payload size, defaults to InfluxDB UDP Client default (512 bytes)

# udp\_payload = 512

## Optional SSL Config

# ssl\_ca = "/etc/telegraf/ca.pem"

# ssl\_cert = "/etc/telegraf/cert.pem"

# ssl\_key = "/etc/telegraf/key.pem"

## Use SSL but skip chain & host verification

# insecure\_skip\_verify = false

Required parameters:

* urls: List of strings, this is for InfluxDB clustering support. On each flush interval, Telegraf will randomly choose one of the urls to write to. Each URL should start with either http:// or udp://
* database: The name of the database to write to.

Optional parameters:

* write\_consistency: Write consistency (clusters only), can be: "any", "one", "quorum", "all".
* retention\_policy: Retention policy to write to.
* timeout: Write timeout (for the InfluxDB client), formatted as a string. If not provided, will default to 5s. 0s means no timeout (not recommended).
* username: Username for influxdb
* password: Password for influxdb
* user\_agent: Set the user agent for HTTP POSTs (can be useful for log differentiation)
* udp\_payload: Set UDP payload size, defaults to InfluxDB UDP Client default (512 bytes)
* ssl\_ca: SSL CA
* ssl\_cert: SSL CERT
* ssl\_key: SSL key
* insecure\_skip\_verify: Use SSL but skip chain & host verification (default: false)

# 2@telegraf+influxdb+grafana+spring boot构建监控平台

- Soongp的博客 - 博客频道 - CSDN.NET

http://blog.csdn.net/soongp/article/details/66974529

一、数据采集

Telegraf 是一个用 [**Go**](http://lib.csdn.net/base/go) 编写的代理程序，可收集系统和服务的统计数据，并写入到 InfluxDB [**数据库**](http://lib.csdn.net/base/mysql)，Telegraf 具有内存占用小的特点，通过插件系统开发人员可轻松添加支持其他

服务的扩展。

Telegraf是完全配置驱动，所有的数据都通过声明的数据输入源(inputs)收集，并发送给声明的数据输出源(outputs)。

所以，我们通过配置inputs来实现对[**spring**](http://lib.csdn.net/base/javaee)-boot应用的监控。

下载地址：https://cloud.influxdata.com/downloads

[**Linux**](http://lib.csdn.net/base/linux)下安装：

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. ①cd /usr/local
2. ②wget https://dl.influxdata.com/telegraf/releases/telegraf-1.2.1.x86\_64.rpm
3. ③sudo yum localinstall telegraf-1.2.1.x86\_64.rpm

配置：

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. vi /etc/telegraf/telegraf.conf

①配置数据输入源，释放一些基本的服务：

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. [[inputs.cpu]]
2. ## Whether to report per-cpu stats or not
3. percpu = **true**
4. ## Whether to report total system cpu stats or not
5. totalcpu = **true**
6. ## If **true**, collect raw CPU time metrics.
7. collect\_cpu\_time = **false**
9. [[inputs.mem]]
10. # no configuration
12. [[inputs.zookeeper]]
13. #   ## An array of address to gather stats about. Specify an ip or hostname
14. #   ## with port. ie localhost:2181, 10.0.0.1:2181, etc.
15. #
16. #   ## If no servers are specified, then localhost is used as the host.
17. #   ## If no port is specified, 2181 is used
18. #   servers = [":2181"]
19. servers = ["172.21.121.53:2181","172.21.121.54:2181","172.21.121.55:2181"]

还支持：kafka、MQTT、redis等服务，

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. # # Read JMX metrics through Jolokia
2. [[inputs.jolokia]]
3. #   ## This is the context root used to compose the jolokia url
4. #   ## NOTE that Jolokia requires a trailing slash at the end of the context root
5. #   ## NOTE that your jolokia security policy must allow **for** POST requests.
6. context = "/jolokia/"
7. #
8. #   ## This specifies the mode used
9. #   # mode = "proxy"
10. #   #
11. #   ## When in proxy mode **this** section is used to specify further
12. #   ## proxy address configurations.
13. #   ## Remember to change host address to fit your environment.
14. #   # [inputs.jolokia.proxy]
15. #   #   host = "127.0.0.1"
16. #   #   port = "8080"
17. #
18. #   ## Optional http timeouts
19. #   ##
20. #   ## response\_header\_timeout, **if** non-zero, specifies the amount of time to wait
21. #   ## **for** a server's response headers after fully writing the request.
22. #   # response\_header\_timeout = "3s"
23. #   ##
24. #   ## client\_timeout specifies a time limit **for** requests made by **this** client.
25. #   ## Includes connection time, any redirects, and reading the response body.
26. #   # client\_timeout = "4s"
27. #
28. #   ## List of servers exposing jolokia read service
29. [[inputs.jolokia.servers]]#这里指向的就是jolokia所监控的应用，技术这东西融会贯通，构建spring boot的helloWorld
30. #http://projects.spring.io/spring-boot/#quick-start，具体的一些框架整合的配置请查阅官方文档
31. name = "server-001"
32. host = "127.0.0.1"
33. port = "8089"
34. #     # username = "myuser"
35. #     # password = "mypassword"
36. #
37. #   ## List of metrics collected on above servers
38. #   ## Each metric consists in a name, a jmx path and either
39. #   ## a pass or drop slice attribute.
40. #   ##?This collect all heap memory usage metrics.
41. [[inputs.jolokia.metrics]]#这里就是一些监控的参数了：内存使用、类加载，线程等
42. name = "heap\_memory\_usage"
43. mbean  = "java.lang:type=Memory"
44. attribute = "HeapMemoryUsage"
45. #
46. #   ##?This collect thread counts metrics.
47. [[inputs.jolokia.metrics]]
48. name = "thread\_count"
49. mbean  = "java.lang:type=Threading"
50. attribute = "TotalStartedThreadCount,ThreadCount,DaemonThreadCount,PeakThreadCount"
51. #
52. #   ##?This collect number of **class** loaded/unloaded counts metrics.
53. [[inputs.jolokia.metrics]]
54. name = "class\_count"
55. mbean  = "java.lang:type=ClassLoading"
56. attribute = "LoadedClassCount,UnloadedClassCount,TotalLoadedClassCount"

②配置数据输出源：

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. #这里的配置指向了数据存储：influxdb数据库
2. [[outputs.influxdb]]
3. ## The full HTTP or UDP endpoint URL **for** your InfluxDB instance.
4. ## Multiple urls can be specified as part of the same cluster,
5. ## **this** means that only ONE of the urls will be written to each interval.
6. # urls = ["udp://localhost:8089"] # UDP endpoint example
7. urls = ["http://172.21.121.54:8086"] # required
8. ## The target database **for** metrics (telegraf will create it **if** not exists).
9. database = "telegraf" # required
11. ## Retention policy to write to. Empty string writes to the **default** rp.
12. retention\_policy = ""
13. ## Write consistency (clusters only), can be: "any", "one", "quorum", "all"
14. write\_consistency = "any"
16. ## Write timeout (**for** the InfluxDB client), formatted as a string.
17. ## If not provided, will **default** to 5s. 0s means no timeout (not recommended).
18. timeout = "5s"
19. username = "admin"
20. password = "admin"
21. # username = "telegraf"
22. # password = "metricsmetricsmetricsmetrics"
23. ## Set the user agent **for** HTTP POSTs (can be useful **for** log differentiation)
24. # user\_agent = "telegraf"
25. ## Set UDP payload size, defaults to InfluxDB UDP Client **default** (512 bytes)
26. # udp\_payload = 512
28. ## Optional SSL Config
29. # ssl\_ca = "/etc/telegraf/ca.pem"
30. # ssl\_cert = "/etc/telegraf/cert.pem"
31. # ssl\_key = "/etc/telegraf/key.pem"
32. ## Use SSL but skip chain & host verification
33. # insecure\_skip\_verify = **false**

这里的配置信息是最后调试成功的配置信息，正常来讲，telegraf是最后来配置。

启动：

sudo service telegraf start [stop停止、status、restart]

二、数据存储：influxdb

InfluxDB 是一个开源分布式时序、事件和指标数据库。使用 Go 语言编写，无需外部依赖。其设计目标是实现分布式和水平伸缩扩展。

它有三大特性：

1. Time Series （时间序列）：你可以使用与时间有关的相关函数（如最大，最小，求和等）

2. Metrics（度量）：你可以实时对大量数据进行计算

3. Eevents（事件）：它支持任意的事件数据

下载地址：https://cloud.influxdata.com/downloads

linux下安装：

注意：influxdb的安装依赖3个端口：8083、8086、8088，所以安装之前，先检查端口是否被占用：netstat -apn | grep 8080

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. ①cd /usr/local
2. ②wget https://dl.influxdata.com/influxdb/releases/influxdb-0.13.0.x86\_64.rpm
3. （这个版本有后台管理，1.1版本后。删除了后台管理功能，你也可以下载其他版本）
4. ③sudo yum localinstall influxdb-0.13.0.x86\_64.rpm
5. ④vi /etc/influxdb/influxdb.conf
6. 可以配置一下端口(8083和8086)

启动：

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. sudo service influxdb start     [stop停止、status、restart]

访问：

访问http://localhost:8083来访问influxdb的web页面；

三、配置spring boot应用

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. 引入spring-boot-admin依赖
2. <dependency>
3. <groupId>de.codecentric</groupId>
4. <artifactId>spring-boot-admin-server</artifactId>
5. <version>1.4.0</version>
6. </dependency>
8. <dependency>
9. <groupId>de.codecentric</groupId>
10. <artifactId>spring-boot-admin-server-ui</artifactId>
11. <version>1.4.0</version>
12. </dependency>

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. 引入actuator依赖
2. <dependency>
3. <groupId>org.springframework.boot</groupId>
4. <artifactId>spring-boot-starter-actuator</artifactId>
5. <version>1.4.3.RELEASE</version>
6. </dependency>
8. 引入jolokia依赖
9. <dependency>
10. <groupId>org.jolokia</groupId>
11. <artifactId>jolokia-core</artifactId>
12. <version>1.3.5</version>
13. </dependency>

配置application.properties：

增加

**[java]** [view plain](http://blog.csdn.net/soongp/article/details/66974529) [copy](http://blog.csdn.net/soongp/article/details/66974529)

1. jolokia.config.debug=**true**
2. endpoints.jolokia.enabled=**true**
3. #endpoints.jolokia.sensitive=**true**
4. endpoints.jolokia.path=/jolokia
5. endpoints: enabled: **true** jmx: enabled: **true** jolokia: enabled: **true** management: security: enabled: **false**

最后启动，并访问http://172.21.121.54:8089/jolokia/read/org.springframework.boot:name=metricsEndpoint,type=Endpoint/Data

若有数据返回，则整合成功；

四、数据展示：grafana

Grafana是一个纯粹的html/js应用，访问InfluxDB时不会有跨域访问的限制。

只要配置好数据源为InfluxDB之后就可以，剩下的工作就是配置图表。Grafana 功能非常强大。

这个我安装的是windows版。

下载地址：https://grafana.com/grafana/download

配置：

复制conf文件夹下sqmple.int，命名为custom.ini

修改custom.ini文件

①配置日志文件保存路径：logs = /var/log/grafana 这个路径自定义

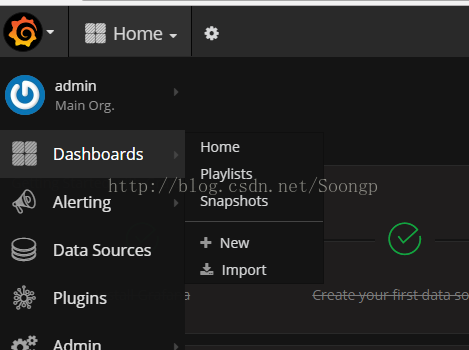
②配置端口号：http\_port = 8091

③配置数据库：三种选择，[**MySQL**](http://lib.csdn.net/base/mysql)，postgres，sqlite3

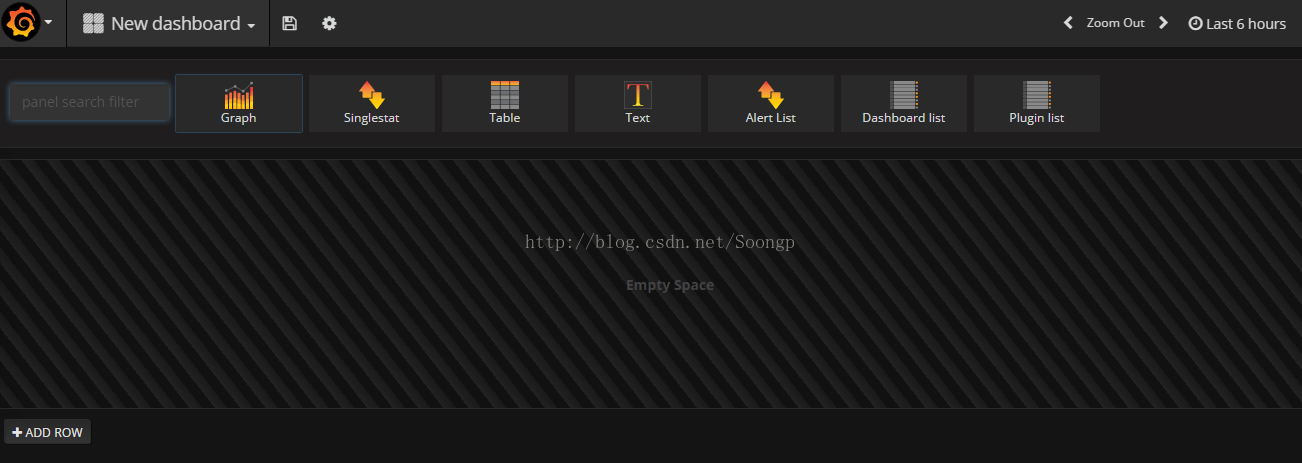
url = mysql://root:root@172.21.121.52:3306/test

启动：运行grafana-server

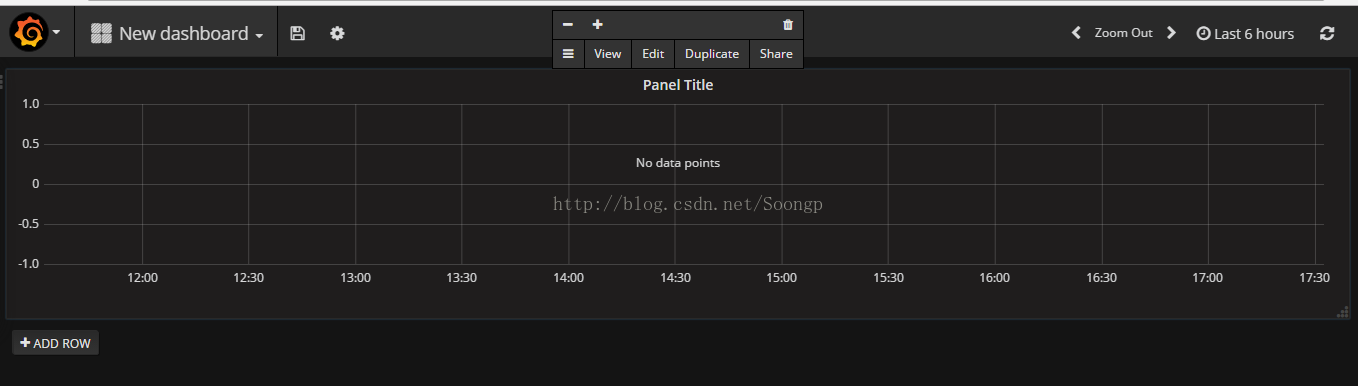
访问：http://localhost:8091，进行展示数据配置



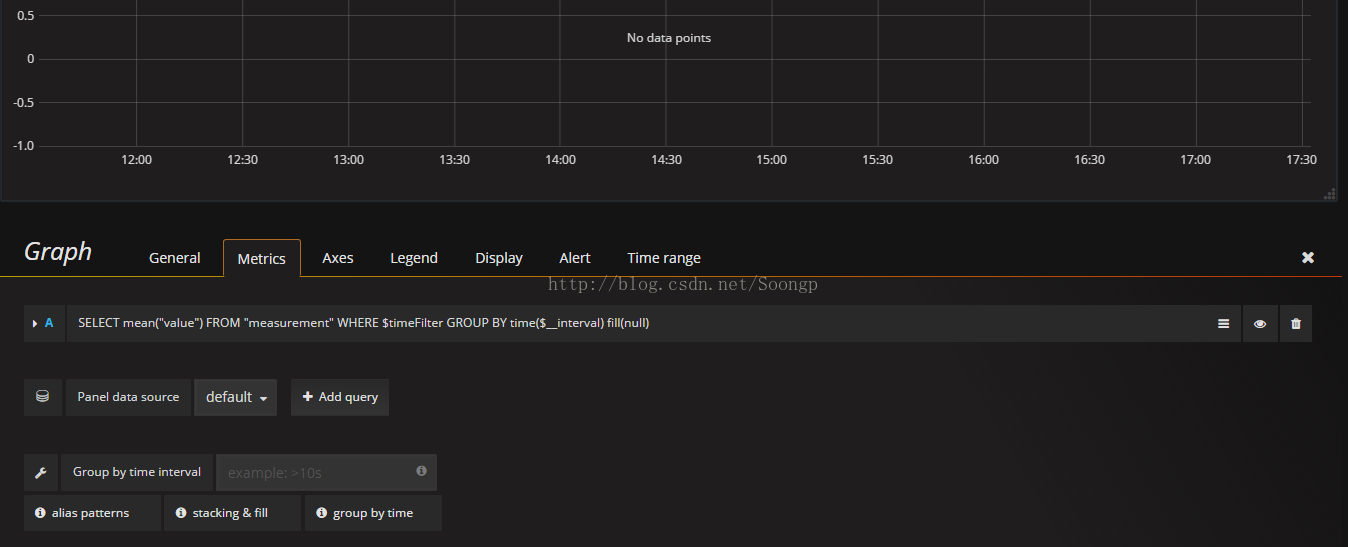
点击new创建一个Dashboards



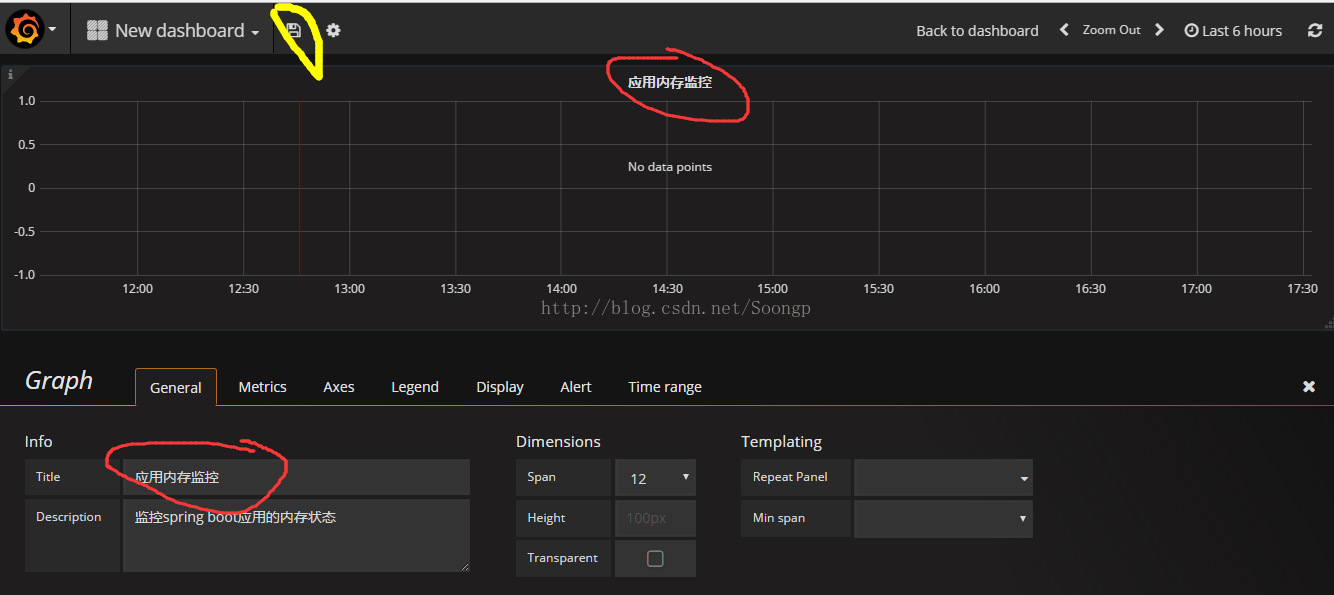
选择Graph



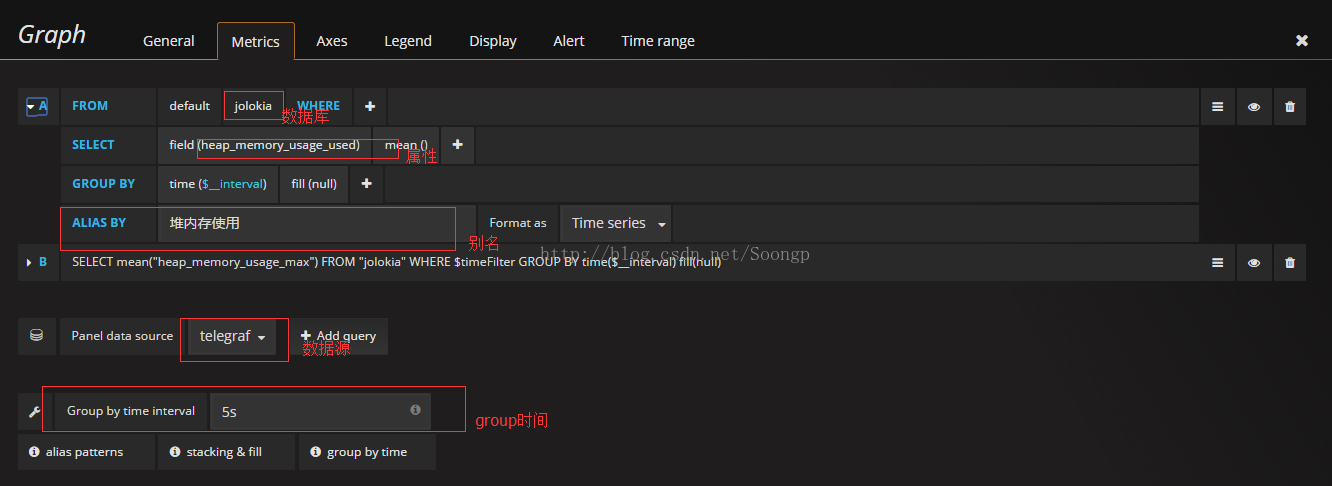
点击标题Panel Title,点击弹出框中的Edit,对这个图表进行配置



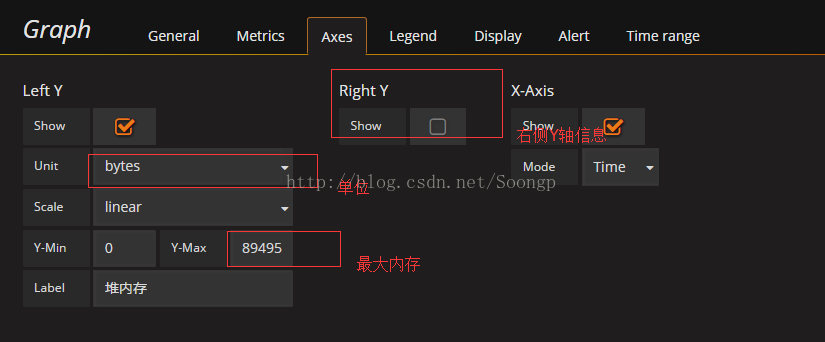
点击General



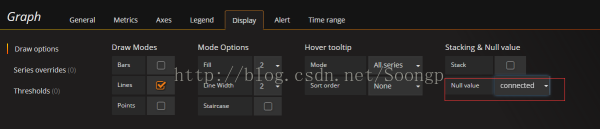
配置图表基本信息，然后点击保存。点击Metrics



点击Axes



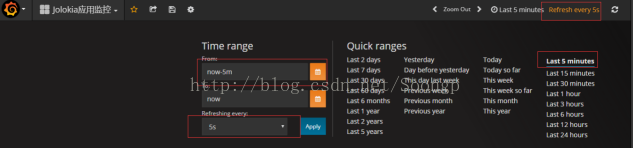
点击display



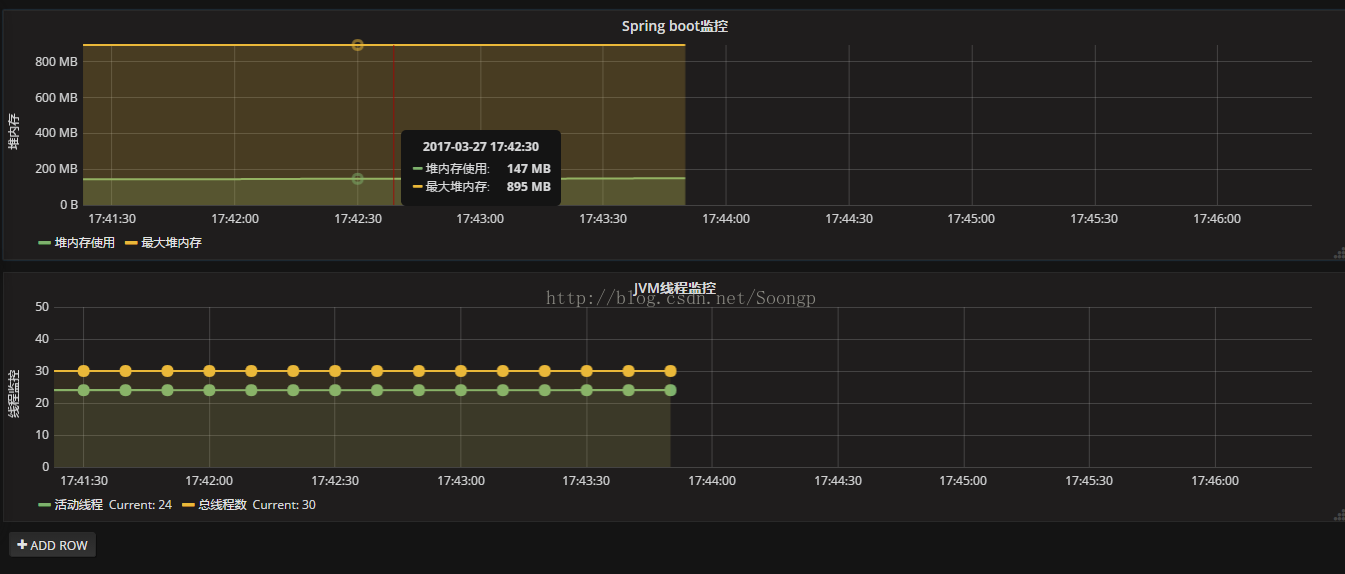
选择connected

最后保存；

点击右上角时间：



时间范围是5分钟，每5秒刷新一次数据，最终结果：



# Docker Metrics with Telegraf · benfb

https://benbailey.me/2015/06/docker-telegraf-metrics/

With the [release of InfluxDB v0.9](https://influxdb.com/blog/2015/06/11/InfluxDB-v0_9_0-released-with-developer-and-production-support.html), I was eager to start using Google’s [cAdvisor](https://github.com/google/cadvisor) to begin collecting metrics from Docker containers. Unfortunately, the new InfluxDB version comes with a new breaking API that cAdvisor still isn’t compatible with. Not only does cAdvisor not support the new API, it’s currently impossible to successfully run go get github.com/google/cadvisor because of this issue. After struggling with cAdvisor for a month, I learned that InfluxDB [recently rolled out](https://influxdb.com/blog/2015/06/19/Announcing-Telegraf-a-metrics-collector-for-InfluxDB.html) their own metrics collector, [Telegraf](https://github.com/influxdb/telegraf), which is pretty much guaranteed to have the best InfluxDB integration possible.

The new version of InfluxDB also includes alpha support for clustering, which is key when working with large infrastructures. In InfluxDB, each node is a broker node, a data node, or both. Data nodes host the data, while brokers are members of a raft consensus group.[1](https://benbailey.me/2015/06/docker-telegraf-metrics/#fn:1) In this Docker cluster, I chose to run a data node on every machine in order to reduce network throughput at the cost of slightly increased disk usage. This decision also makes Telegraf easier to set up, as with the right network configuration it can just report to localhost.

Thus, the docker command to start up an InfluxDB cluster look something like this:

docker run -e FORCE\_HOSTNAME=auto -e PRE\_CREATE\_DB="telegraf" -e REPLI\_FACTOR="3" --volume=/influxdb:/data --publish=8083 --publish=8086 --expose 8090 --expose=8099 -d tutum/influxdb:latest

docker run -e FORCE\_HOSTNAME=auto -e SEEDS="master:8090" --volume=/influxdb:/data --expose 8090 --expose=8099 -d tutum/influxdb:latest

docker run -e FORCE\_HOSTNAME=auto -e SEEDS="master:8090" --volume=/influxdb:/data --expose 8090 --expose=8099 -d tutum/influxdb:latest

Currently Telegraf only supports Vagrant officially. I made a Docker repository atbb/telegraf that will suffice for now. You can start it up with

docker run -d bbailey/telegraf

and it will automatically use localhost:8086 as the InfluxDB URL.

After running these containers, you should start to see data appearing in InfluxDB. All that’s left is to access it. Luckily, it’s very simple to get important data from the InfluxDB API using the [native Go client](https://godoc.org/github.com/influxdb/influxdb/client):

q := "SELECT percentile(value, 95) FROM docker\_system WHERE name='telegraf' ORDER BY asc"

res, \_ := queryDB(con, q)

fmt.Println(res[0].Series[0].Values[0][1])

This gets you the 90th percentile of the CPU usage of the docker container named “telegraf.”

In a very basic benchmark test involving top, Telegraf used less than half the CPU that cAdvisor did. While Telegraf doesn’t have the same strong focus on Docker and the documentation is incredibly sparse, the metrics it provides are useful and serve enough of the same purposes, and its native InfluxDB integration makes it a welcome change from other metrics reporters.

1. InfluxDB allows for a maximum of three brokers in the current version, but that still allows for one failure which should be plenty [[return]](https://benbailey.me/2015/06/docker-telegraf-metrics/#fnref:1)

# influxdata/telegraf:

The plugin-driven server agent for collecting & reporting metrics.

https://github.com/influxdata/telegraf

Telegraf

Telegraf is an agent written in Go for collecting, processing, aggregating, and writing metrics.

Design goals are to have a minimal memory footprint with a plugin system so that developers in the community can easily add support for collecting metrics from well known services (like Hadoop, Postgres, or Redis) and third party APIs (like Mailchimp, AWS CloudWatch, or Google Analytics).

Telegraf is plugin-driven and has the concept of 4 distinct plugins:

1. [Input Plugins](https://github.com/influxdata/telegraf#input-plugins) collect metrics from the system, services, or 3rd party APIs
2. [Processor Plugins](https://github.com/influxdata/telegraf#processor-plugins) transform, decorate, and/or filter metrics
3. [Aggregator Plugins](https://github.com/influxdata/telegraf#aggregator-plugins) create aggregate metrics (e.g. mean, min, max, quantiles, etc.)
4. [Output Plugins](https://github.com/influxdata/telegraf#output-plugins) write metrics to various destinations

For more information on Processor and Aggregator plugins please [read this](https://github.com/influxdata/telegraf/blob/master/docs/AGGREGATORS_AND_PROCESSORS.md).

New plugins are designed to be easy to contribute, we'll eagerly accept pull requests and will manage the set of plugins that Telegraf supports. See the [contributing guide](https://github.com/influxdata/telegraf/blob/master/CONTRIBUTING.md) for instructions on writing new plugins.

Installation:

You can either download the binaries directly from the [downloads](https://www.influxdata.com/downloads) page.

A few alternate installs are available here as well:

FreeBSD tarball:

Latest:

* <https://dl.influxdata.com/telegraf/releases/telegraf-VERSION_freebsd_amd64.tar.gz>

Ansible Role:

Ansible role: <https://github.com/rossmcdonald/telegraf>

From Source:

Telegraf manages dependencies via [gdm](https://github.com/sparrc/gdm), which gets installed via the Makefile if you don't have it already. You also must build with golang version 1.8+.

1. [Install Go](https://golang.org/doc/install)
2. [Setup your GOPATH](https://golang.org/doc/code.html#GOPATH)
3. Run go get github.com/influxdata/telegraf
4. Run cd $GOPATH/src/github.com/influxdata/telegraf
5. Run make

How to use it:

See usage with:

telegraf --help

Generate a telegraf config file:

telegraf config > telegraf.conf

Generate config with only cpu input & influxdb output plugins defined

telegraf --input-filter cpu --output-filter influxdb config

Run a single telegraf collection, outputing metrics to stdout

telegraf --config telegraf.conf -test

Run telegraf with all plugins defined in config file

telegraf --config telegraf.conf

Run telegraf, enabling the cpu & memory input, and influxdb output plugins

telegraf --config telegraf.conf -input-filter cpu:mem -output-filter influxdb

Configuration

See the [configuration guide](https://github.com/influxdata/telegraf/blob/master/docs/CONFIGURATION.md) for a rundown of the more advanced configuration options.

Input Plugins

* [aerospike](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/aerospike)
* [amqp\_consumer](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/amqp_consumer) (rabbitmq)
* [apache](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/apache)
* [aws cloudwatch](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/cloudwatch)
* [bcache](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/bcache)
* [cassandra](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/cassandra)
* [ceph](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/ceph)
* [cgroup](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/cgroup)
* [chrony](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/chrony)
* [consul](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/consul)
* [conntrack](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/conntrack)
* [couchbase](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/couchbase)
* [couchdb](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/couchdb)
* [disque](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/disque)
* [dns query time](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/dns_query)
* [docker](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/docker)
* [dovecot](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/dovecot)
* [elasticsearch](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/elasticsearch)
* [exec](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/exec) (generic executable plugin, support JSON, influx, graphite and nagios)
* [filestat](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/filestat)
* [haproxy](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/haproxy)
* [hddtemp](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/hddtemp)
* [http\_response](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/http_response)
* [httpjson](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/httpjson) (generic JSON-emitting http service plugin)
* [internal](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/internal)
* [influxdb](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/influxdb)
* [ipmi\_sensor](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/ipmi_sensor)
* [iptables](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/iptables)
* [jolokia](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/jolokia)
* [kubernetes](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/kubernetes)
* [leofs](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/leofs)
* [lustre2](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/lustre2)
* [mailchimp](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/mailchimp)
* [memcached](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/memcached)
* [mesos](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/mesos)
* [mongodb](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/mongodb)
* [mysql](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/mysql)
* [net\_response](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/net_response)
* [nginx](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/nginx)
* [nsq](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/nsq)
* [nstat](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/nstat)
* [ntpq](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/ntpq)
* [phpfpm](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/phpfpm)
* [phusion passenger](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/passenger)
* [ping](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/ping)
* [postgresql](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/postgresql)
* [postgresql\_extensible](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/postgresql_extensible)
* [powerdns](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/powerdns)
* [procstat](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/procstat)
* [prometheus](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/prometheus)
* [puppetagent](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/puppetagent)
* [rabbitmq](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/rabbitmq)
* [raindrops](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/raindrops)
* [redis](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/redis)
* [rethinkdb](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/rethinkdb)
* [riak](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/riak)
* [sensors](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/sensors)
* [snmp](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/snmp)
* [snmp\_legacy](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/snmp_legacy)
* [sql server](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/sqlserver) (microsoft)
* [twemproxy](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/twemproxy)
* [varnish](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/varnish)
* [zfs](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/zfs)
* [zookeeper](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/zookeeper)
* [win\_perf\_counters](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/win_perf_counters)(windows performance counters)
* [sysstat](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/sysstat)
* [system](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/system)
  + cpu
  + mem
  + net
  + netstat
  + disk
  + diskio
  + swap
  + processes
  + kernel (/proc/stat)
  + kernel (/proc/vmstat)
  + linux\_sysctl\_fs (/proc/sys/fs)

Telegraf can also collect metrics via the following service plugins:

* [http\_listener](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/http_listener)
* [kafka\_consumer](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/kafka_consumer)
* [mqtt\_consumer](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/mqtt_consumer)
* [nats\_consumer](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/nats_consumer)
* [nsq\_consumer](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/nsq_consumer)
* [logparser](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/logparser)
* [statsd](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/statsd)
* [socket\_listener](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/socket_listener)
* [tail](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/tail)
* [tcp\_listener](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/socket_listener)
* [udp\_listener](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/socket_listener)
* [webhooks](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/webhooks)
  + [filestack](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/webhooks/filestack)
  + [github](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/webhooks/github)
  + [mandrill](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/webhooks/mandrill)
  + [rollbar](https://github.com/influxdata/telegraf/blob/master/plugins/inputs/webhooks/rollbar)

Processor Plugins

* [printer](https://github.com/influxdata/telegraf/blob/master/plugins/processors/printer)

Aggregator Plugins

* [minmax](https://github.com/influxdata/telegraf/blob/master/plugins/aggregators/minmax)

Output Plugins

* [influxdb](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/influxdb)
* [amon](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/amon)
* [amqp](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/amqp) (rabbitmq)
* [aws kinesis](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/kinesis)
* [aws cloudwatch](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/cloudwatch)
* [datadog](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/datadog)
* [discard](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/discard)
* [elasticsearch](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/elasticsearch)
* [file](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/file)
* [graphite](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/graphite)
* [graylog](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/graylog)
* [instrumental](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/instrumental)
* [kafka](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/kafka)
* [librato](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/librato)
* [mqtt](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/mqtt)
* [nats](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/nats)
* [nsq](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/nsq)
* [opentsdb](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/opentsdb)
* [prometheus](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/prometheus_client)
* [riemann](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/riemann)
* [riemann\_legacy](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/riemann_legacy)
* [socket\_writer](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/socket_writer)
* [tcp](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/socket_writer)
* [udp](https://github.com/influxdata/telegraf/blob/master/plugins/outputs/socket_writer)

Contributing

Please see the [contributing guide](https://github.com/influxdata/telegraf/blob/master/CONTRIBUTING.md) for details on contributing a plugin to Telegraf.

# telegraf/plugins/inputs/docker at master · influxdata/telegraf

https://github.com/influxdata/telegraf/tree/master/plugins/inputs/docker

Docker Input Plugin

The docker plugin uses the docker remote API to gather metrics on running docker containers. You can read Docker's documentation for their remote API [here](https://docs.docker.com/engine/reference/api/docker_remote_api_v1.20/#get-container-stats-based-on-resource-usage)

The docker plugin uses the excellent [docker engine-api](https://github.com/docker/engine-api) library to gather stats. Documentation for the library can be found [here](https://godoc.org/github.com/docker/engine-api)and documentation for the stat structure can be found [here](https://godoc.org/github.com/docker/engine-api/types#Stats)

Configuration:

# Read metrics about docker containers

[[inputs.docker]]

## Docker Endpoint

## To use TCP, set endpoint = "tcp://[ip]:[port]"

## To use environment variables (ie, docker-machine), set endpoint = "ENV"

endpoint = "unix:///var/run/docker.sock"

## Only collect metrics for these containers, collect all if empty

container\_names = []

## Timeout for docker list, info, and stats commands

timeout = "5s"

## Whether to report for each container per-device blkio (8:0, 8:1...) and

## network (eth0, eth1, ...) stats or not

perdevice = true

## Whether to report for each container total blkio and network stats or not

total = false

## docker labels to include and exclude as tags. Globs accepted.

## Note that an empty array for both will include all labels as tags

docker\_label\_include = []

docker\_label\_exclude = []

Measurements & Fields:

Every effort was made to preserve the names based on the JSON response from the docker API.

Note that the docker\_container\_cpu metric may appear multiple times per collection, based on the availability of per-cpu stats on your system.

* docker\_container\_mem
  + total\_pgmafault
  + cache
  + mapped\_file
  + total\_inactive\_file
  + pgpgout
  + rss
  + total\_mapped\_file
  + writeback
  + unevictable
  + pgpgin
  + total\_unevictable
  + pgmajfault
  + total\_rss
  + total\_rss\_huge
  + total\_writeback
  + total\_inactive\_anon
  + rss\_huge
  + hierarchical\_memory\_limit
  + total\_pgfault
  + total\_active\_file
  + active\_anon
  + total\_active\_anon
  + total\_pgpgout
  + total\_cache
  + inactive\_anon
  + active\_file
  + pgfault
  + inactive\_file
  + total\_pgpgin
  + max\_usage
  + usage
  + failcnt
  + limit
  + container\_id
* docker\_container\_cpu
  + throttling\_periods
  + throttling\_throttled\_periods
  + throttling\_throttled\_time
  + usage\_in\_kernelmode
  + usage\_in\_usermode
  + usage\_system
  + usage\_total
  + usage\_percent
  + container\_id
* docker\_container\_net
  + rx\_dropped
  + rx\_bytes
  + rx\_errors
  + tx\_packets
  + tx\_dropped
  + rx\_packets
  + tx\_errors
  + tx\_bytes
  + container\_id
* docker\_container\_blkio
  + io\_service\_bytes\_recursive\_async
  + io\_service\_bytes\_recursive\_read
  + io\_service\_bytes\_recursive\_sync
  + io\_service\_bytes\_recursive\_total
  + io\_service\_bytes\_recursive\_write
  + io\_serviced\_recursive\_async
  + io\_serviced\_recursive\_read
  + io\_serviced\_recursive\_sync
  + io\_serviced\_recursive\_total
  + io\_serviced\_recursive\_write
  + container\_id
* docker\_
  + n\_used\_file\_descriptors
  + n\_cpus
  + n\_containers
  + n\_containers\_running
  + n\_containers\_stopped
  + n\_containers\_paused
  + n\_images
  + n\_goroutines
  + n\_listener\_events
  + memory\_total
  + pool\_blocksize
* docker\_data
  + available
  + total
  + used
* docker\_metadata
  + available
  + total
  + used

Tags:

Docker Engine tags

* docker (memory\_total)
  + unit=bytes
  + engine\_host
* docker (pool\_blocksize)
  + unit=bytes
  + engine\_host
* docker\_data
  + unit=bytes
  + engine\_host
* docker\_metadata
  + unit=bytes
  + engine\_host

Docker Container tags

* Tags on all containers:
  + engine\_host
  + container\_image
  + container\_name
  + container\_version
* docker\_container\_mem specific:
* docker\_container\_cpu specific:
  + cpu
* docker\_container\_net specific:
  + network
* docker\_container\_blkio specific:
  + device

Example Output:

% ./telegraf -config ~/ws/telegraf.conf -input-filter docker -test

\* Plugin: docker, Collection 1

> docker n\_cpus=8i 1456926671065383978

> docker n\_used\_file\_descriptors=15i 1456926671065383978

> docker n\_containers=7i 1456926671065383978

> docker n\_containers\_running=7i 1456926671065383978

> docker n\_containers\_stopped=3i 1456926671065383978

> docker n\_containers\_paused=0i 1456926671065383978

> docker n\_images=152i 1456926671065383978

> docker n\_goroutines=36i 1456926671065383978

> docker n\_listener\_events=0i 1456926671065383978

> docker,unit=bytes memory\_total=18935443456i 1456926671065383978

> docker,unit=bytes pool\_blocksize=65540i 1456926671065383978

> docker\_data,unit=bytes available=24340000000i,total=107400000000i,used=14820000000i 1456926671065383978

> docker\_metadata,unit=bytes available=2126999999i,total=2146999999i,used=20420000i 145692667106538

> docker\_container\_mem,

container\_image=spotify/kafka,container\_name=kafka \

active\_anon=52568064i,active\_file=6926336i,cache=12038144i,fail\_count=0i,\

hierarchical\_memory\_limit=9223372036854771712i,inactive\_anon=52707328i,\

inactive\_file=5111808i,limit=1044578304i,mapped\_file=10301440i,\

max\_usage=140656640i,pgfault=63762i,pgmajfault=2837i,pgpgin=73355i,\

pgpgout=45736i,rss=105275392i,rss\_huge=4194304i,total\_active\_anon=52568064i,\

total\_active\_file=6926336i,total\_cache=12038144i,total\_inactive\_anon=52707328i,\

total\_inactive\_file=5111808i,total\_mapped\_file=10301440i,total\_pgfault=63762i,\

total\_pgmafault=0i,total\_pgpgin=73355i,total\_pgpgout=45736i,\

total\_rss=105275392i,total\_rss\_huge=4194304i,total\_unevictable=0i,\

total\_writeback=0i,unevictable=0i,usage=117440512i,writeback=0i 1453409536840126713

> docker\_container\_cpu,

container\_image=spotify/kafka,container\_name=kafka,cpu=cpu-total \

throttling\_periods=0i,throttling\_throttled\_periods=0i,\

throttling\_throttled\_time=0i,usage\_in\_kernelmode=440000000i,\

usage\_in\_usermode=2290000000i,usage\_system=84795360000000i,\

usage\_total=6628208865i 1453409536840126713

> docker\_container\_cpu,

container\_image=spotify/kafka,container\_name=kafka,cpu=cpu0 \

usage\_total=6628208865i 1453409536840126713

> docker\_container\_net,\

container\_image=spotify/kafka,container\_name=kafka,network=eth0 \

rx\_bytes=7468i,rx\_dropped=0i,rx\_errors=0i,rx\_packets=94i,tx\_bytes=946i,\

tx\_dropped=0i,tx\_errors=0i,tx\_packets=13i 1453409536840126713

> docker\_container\_blkio,

container\_image=spotify/kafka,container\_name=kafka,device=8:0 \

io\_service\_bytes\_recursive\_async=80216064i,io\_service\_bytes\_recursive\_read=79925248i,\

io\_service\_bytes\_recursive\_sync=77824i,io\_service\_bytes\_recursive\_total=80293888i,\

io\_service\_bytes\_recursive\_write=368640i,io\_serviced\_recursive\_async=6562i,\

io\_serviced\_recursive\_read=6492i,io\_serviced\_recursive\_sync=37i,\

io\_serviced\_recursive\_total=6599i,io\_serviced\_recursive\_write=107i 1453409536840126713

# Docker监控方案(TIG)的研究与实践之Telegraf - Andy-xu的个人空间

https://my.oschina.net/xxbAndy/blog/751330

*摘要: Docker监控方案之容器内部指标采集工具Telegraf的介绍和安装。Telegraf用纯go编写，通过插件化方式进行采集各种服务(system，docker，redis，nginx，kafka等)监控指标并且上报给相应的中间件，比如influxdb，opentsdb(商城docker监控使用这个)。Telegraf也是整个TICK(telegraf+influxdb+chronograf+kapacitor)生态栈的第一块组件也是最重要的组件。*

**前言**

Docker由于使用了基于namespace和cgroup的技术，因此监控docker容器和监控宿主机在某些性能指标和方式上有一些区别，而传统的监控方式可能无法满足docker容器内部的指标监控，本篇系列文章主要分享使用telegraf+influxdb+grafana去监控docker容器内部资源使用情况。目前主要关注的监控指标为：每个宿主机上的docker容器数量，每个docker容器的内存使用情况，CPU使用情况，网络使用情况以及磁盘使用情况。同时这套方案也能够监控到宿主机的一些基本资源使用情况。

**Telegraf简介与实践**

简介：

由influxdata公司开发的用于采集系统数据的服务，用纯go编写，通过插件化方式进行采集各种服务(system，docker，redis，nginx，kafka等)监控指标并且上报给相应的中间件，比如influxdb，opentsdb(商城docker监控使用这个)。Telegraf也是整个TICK(telegraf+influxdb+chronograf+kapacitor)生态栈的第一块组件也是最重要的组件。

特点：

纯go编写，不需要依赖其他组件；消耗相关系统资源比较小；plugins支持多种输入输出插件(采集和上报)；   
相关连接：

github：<https://github.com/influxdata/telegraf>  
官网文档：<https://docs.influxdata.com/telegraf/v1.0/>  
TICK生态栈:<https://www.influxdata.com/downloads/#telegraf>

安装：

所有的安装以及部署都是在linux下的，所以不知道linux下安装基础软件包的，请自觉绕路！   
Centos系列可以配置yum源或者直接下载包，并安装。个人建议直接下载包，由于不需要其他系统依赖，可以直接在集群环境进行共享。   
wget<https://dl.influxdata.com/telegraf/releases/telegraf-1.0.0.x86_64.rpm> && rpm -ivh telegraf-1.0.0.x86\_64.rpm   
其他环境安装指南:

Ubuntu && Debin：   
ubuntu repo：   
curl -sL<https://repos.influxdata.com/influxdb.key> | sudo apt-key add -   
source /etc/lsb-release   
echo “deb [https://repos.influxdata.com/](https://repos.influxdata.com/%3Cspan%20class='mathjax-replacement'%20rel='227521499c481b5de95e53b2237f2e80'%3E$%7BDISTRIB_ID,,%7D%20$%3C/span%3E){DISTRIB\_CODENAME} stable” | sudo tee /etc/apt/sources.list.d/influxdb.list   
Debin repo：   
curl -sL<https://repos.influxdata.com/influxdb.key> | sudo apt-key add -   
source /etc/os-release   
test $VERSION\_ID = “7” && echo “deb<https://repos.influxdata.com/debian>wheezy stable” | sudo tee /etc/apt/sources.list.d/influxdb.list   
test $VERSION\_ID = “8” && echo “deb<https://repos.influxdata.com/debian> jessie stable” | sudo tee /etc/apt/sources.list.d/influxdb.list   
配置完ubuntu系列的repo之后，就可以执行sudo apt-get update && sudo apt-get install telegraf进行安装了   
直接下载deb包方式：   
wget<https://dl.influxdata.com/telegraf/releases/telegraf_1.0.0_amd64.deb>&& sudo dpkg -i telegraf\_1.0.0\_amd64.deb

启动：

telegraf可以支持多种服务管理方式，安装之后默认可用使用service和systemd进行管理，因此在centos6-7中都可以使用系统自带的服务管理进行维护(init.d和systemctl)

/etc/init.d/telegraf start 或者systemctl restart telegraf

配置：

配置可以说是telegraf运用中最核心的一个环节，因为配置的细节决定你采集数据的指标。telegraf的配置可以说是比较千变万化，因为可以支持多种输出、输入组件，并且每种组件的配置支持不通的过滤规则，能够让配置管理和维护者正确的采集自己需要的信息。   
默认配置文件存放路径：/etc/telegraf/telegraf.conf ，额外配置路径/etc/telegraf/telegraf.d/。   
在生产环境中建议自定生成配置并存放在/etc/telegraf/telegraf.d/中。   
自定义生成配置文件：

#telegraf -sample-config > telegraf.conf 这样生成的配置文件将包含每一个插件，但是大部分会被注释掉，可以根据实际的业务场景进行定义

配置文件示例以及详细讲解：

#cat telegraf.conf

########################################全局配置############################################################

#全局tag配置，采用key = "values"方式，这样在本机采集到的所有数据将都有这个标签

[global\_tags]

dc = "docker-test"

#agent配置

[agent]

#默认的数据(input)采集间隔时间

interval = "10s"

#采用轮询时间间隔。默认是使用interval里面的值进行轮询，比如interval = "10s",那采集时间将是:00, :10, :20, 等

round\_interval = true

#每次发送到output的度量大小不能超过metric\_batch\_size的值

metric\_batch\_size = 1000

#telegraf会为每一个output去缓存一份度量值，metric\_buffer\_limit为缓存的限制，并且刷新buffer以确定成功写入。如果达到这个限制了，老的数据会被第一时间丢弃

#当然了，增加这个值能够容忍更多的数据连接，但是这也将会增加telegraf潜在的内存占用。这个值可以大于metric\_batch\_size但是必须小于它的两倍

metric\_buffer\_limit = 10000

#通过随机度量来对采集时间进行抖动。每个插件在采集数据之前将会有一个随机时间的休眠，但是这个时间应小于collection\_jitter

#这个设置是为了防止多个采集源数据同一时间都在队列

collection\_jitter = "0s"

#默认所有数据flush到outputs的时间(在数据被flush到output之前，最大能到flush\_interval + flush\_jitter)。不能低于interval

flush\_interval = "10s"

# 通过随机数来对flush间隔进行抖动。这个主要是为了避免当运行一个大的telegraf实例的时候有比较大的写入。(jitter=5s,flush\_interval=10s意味着每10-15s会发生一次flush操作)

flush\_jitter = "0s"

#默认这个值被设置相同的时间戳通过采集间隔排序。最大值为1s。这个指标一般不会用在service input(比如logparser和statsd)。单位(ns,us,ms,s)

precision = ""

#以debug模式运行

debug = false

#以安静模式运行

quiet = false

#这个将会覆盖默认的hostname，如果为空的话，将会采用os.Hostname()

hostname = ""

#如果设置为true，就不允许在telegraf agent里面设置"host"标签了

omit\_hostname = false

##############################################度量值过滤#######################################################

#过滤可以被配置在每一个输入和输出值

namepass：一个数组字符串可以被用来过滤由当前input生成的度量值，在数组中的每一个字符串和全局匹配到的测量值名字进行对比，如果匹配了，值被采用

namedrop:pass的反向含义，如果匹配，则不使用

fieldpass：在namepass满足的条件下，output的fieldpass不可用

fielddrop：pass的反向含义，如果field名字匹配，将不被采用。output的fielddrop不可用

tagpass：tag names和数组中的字符串都被用来过滤当前input的值，数组中的每一个每一个字符串和tag name对比，匹配则则采用

tagdrop:tagpass的反向含义，如果tag匹配，该度量值不被采用

tagexclude:被用来从度量值(measurements)中执行一个tag。作为tagdrop的对立面，它将丢弃所有依赖于tag的相关度量值，tagexclude只是单纯的从度量值中给tag一个key

这个可以被用作input和output中，但是强烈建议用在input中，他会在同一个采集时间点更加有效的过滤out tags

taginclude:tagexclude的反向含义。在最终的度量值中，也将包含tag keys

注意：tagpass和tagdrop参数必须等一在plugin函数的底部，不然对应的子plugin配置可能被tagpass/tagdrop映射中的内容截断

#################################################(OUTPUT)输出配置##############################################

#输出插件，我们使用的是influxdb，得先进行安装配置

[[outputs.influxdb]]

## The full HTTP or UDP endpoint URL for your InfluxDB instance.

#如果有多个urls，可以指定为相同集群的一部分。意味着urls中的一个将被写到每一个间隔

# urls = ["udp://localhost:8089"] # UDP endpoint example

urls = ["http://172.25.46.7:8086"] # required

#默认需要连接的telegraf库，没有则自己创建

database = "telegraf" # required

precision = "s"

#修改保留策略

retention\_policy = ""

#持续写入，仅支持集群模式, can be: "any", "one", "quorum", "all"

write\_consistency = "any"

#作为influxdb客户端，设置写超时时间，如果为空默认为5s超时，0s表示不设置超时时间(不建议)

timeout = "5s"

#设置telegraf的库的用户名和密码

# username = "telegraf"

# password = "metricsmetricsmetricsmetrics"

###############################################(INPUT)输入配置###################################################

inputs插件全局参数:

#每一个input都可以配置的全局配置项

#name\_override:覆盖默认的度量值名字(默认是input的名字)

#name\_prefix:指定一个前缀并附加到度量值的名字(measuerments name)

#name\_suffix:指定后缀

#tags：一个标签映射到指定的input度量值

#interval：多久采集一次数据，默认可用使用全局配置中的参数

配置示例：

[[inputs.cpu]]

#采集每个cpu的指标

percpu = true

#采集总的cpu指标

totalcpu = true

#会丢弃掉time开头的。如果想要采集原始的cpu相关指标，请注释

fielddrop = ["time\_\*"]

[[inputs.disk]]

#默认的telegraf将手机所有挂载点的信息

#下面这个参数可以指定挂载点

mount\_points = ["/"]

#仅存储磁盘inode相关的度量值

fieldpass = ["inodes\*"]

#通过文件系统类型来忽略一些挂载点，比如tmpfs

ignore\_fs = ["tmpfs", "devtmpfs"]

#仅存储tagpass相关的信息

[inputs.disk.tagpass]

fstype = [ "ext4", "xfs" ]

path = [ "/export", "/home\*" ]

#默认telegraf将采集所有存储设备的信息，devices参数可以指定

# devices = ["sda", "sdb"]

#如果需要磁盘的串行号可以将下面注释打开

# skip\_serial\_number = false

[[inputs.mem]]

#采集docker和redis的插件

[[inputs.docker]]

#指定docker启动的api接口，并指定需要采集那些容器指标

endpoint = "tcp://10.0.0.2:5256"

container\_names = []

[[inputs.redis]]

#指定redis的相关接口

servers = ["tcp://10.0.0.1:6379"]

测试插件是否正常工作：

使用以下命令会将telegraf采集的数据默认输出到终端，依次来检验配置的监控项是否是自己所期望的指标。

#telegraf -config /etc/telegraf/telegraf.conf -input-filter docker -test 会输出docker相关的监控信息说明配置正确(当然也可以去测试其他inputs plugins)

**注意：上面的配置文件中使用的output plugins是influxdb，因此在没有成功配置influxdb的前提下，此配置文件是不能正常让telegraf正常启动的！下一节将会讲到influxdb的相关知识influxdb相关：**

重启服务：

centos6.x:

#/etc/init.d/telegraf restart （service telegraf restart ）

centos7.x:

#systemctl restart telegraf

此时，可以查看相关日志，确保telegraf正常启动，启动之后去influxdb就可以查询相关采集到的数据。

附：

telegraf常用的input plugins:

收集docker相关的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/docker>主要是通过docker API调用相关监控   
收集相关redis的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/redis>  
收集相关mesos的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mesos>  
收集相关nginx的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/nginx>  
收集相关mysql的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/mysql>  
收集ping相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/ping>  
收集influxdb相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/influxdb>  
收集系统相关的信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/system>  
收集haproxy相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/haproxy>  
收集cgroup相关信息：<https://github.com/influxdata/telegraf/tree/master/plugins/inputs/cgroup>

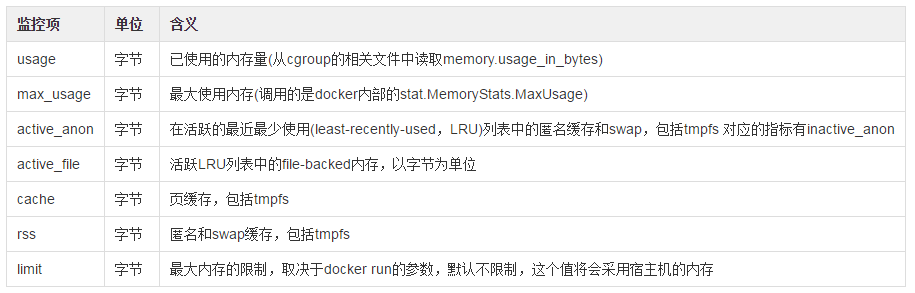
核心plugins 监控指标的采集原理(system，docker)

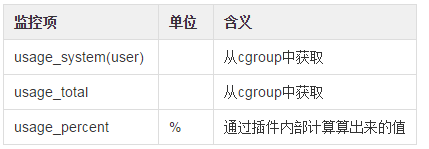
* **system plugin：**主要监控项包含CPU,DISK,KERNEL,KERNEL\_VMSTAT,NETSTAT,PROCESS,SYSTEM

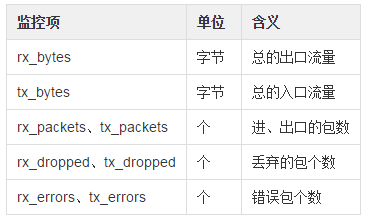
CPU中有两个参数，分别为totalcpu和percpu，如果为true经分别采集相关cpu的指标。主要指标有：user，nice，system，idle，iowait，cpu\_usage等   
DISK：主要指标有free，total；used(单位字节);uesd\_percent;inode\_free;inode\_total;inode\_used.需要注意的是，used\_percent指标通过使用used/(used+free)计算得出。   
MEM：主要指标total;available(/proc/meminfo原生值)；available\_percent(available / total \* 100)；used\_percent(used / total \* 100)   
NET：通过lsof采集tcp连接状态和udp相关信息。指标：established syn\_sent syn\_recv fin\_wait1 time\_wait close listen closing   
PROCESS：收集进程总数个状态组（zombie,sleeping,running),也是通过采集/proc中的数据   
SYSTEM：系统负载,load1;load15;load5

* **docker plugin:**主要监控项包括ocker\_container\_mem,docker\_container\_cpu,docker\_container\_net,docker\_container\_nlkio,docker\_,docker\_data,docker\_metadata。基本上是通过docker api进行采集docker容器相关的监控指标的（<https://docs.docker.com/engine/reference/api/docker_remote_api_v1.25/#/inspect-a-container>）   
  具体的监控项可以在源码中进行查看：(<https://github.com/influxdata/telegraf/blob/master/plugins/inputs/docker/docker.go>)
* **几个主要关心的指标:**

docker\_memory:

  
docker\_cpu:

  
docker\_net:

  
docker相关:

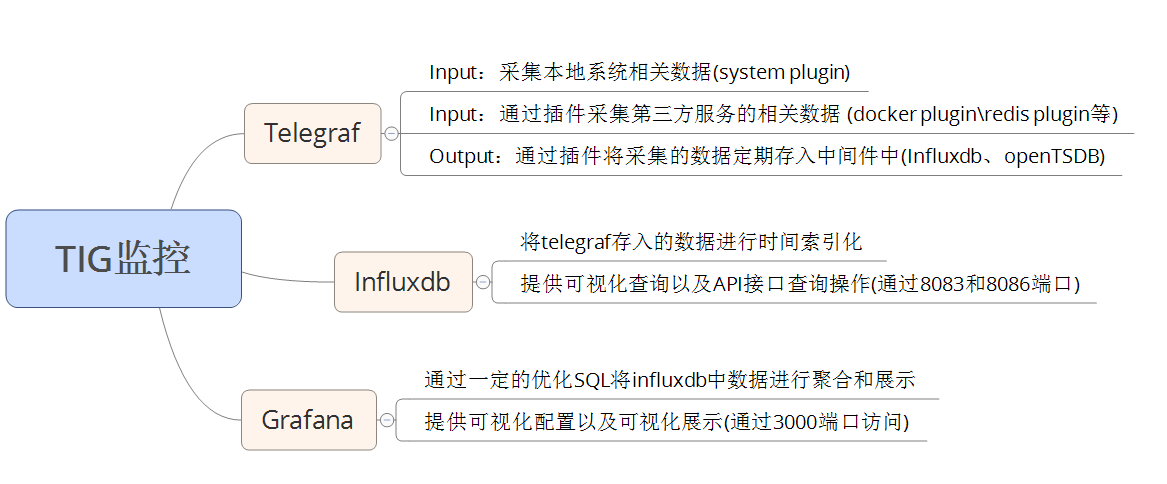


注意：原创著作，转载请联系作者！

# Docker监控套件(Telegraf+Influxdb+Grafana)研究与实践 - Andy-xu的个人空间

https://my.oschina.net/xxbAndy/blog/751325

*摘要: Docker监控方案之TIG方案的介绍以及各个组件的安装部署。Docker底层使用了namespace和cgroup技术，因此需要采集宿主机中各个容器的监控指标，从容器内部去采集相应的数据是不合理并且不建议的，所以必须通过在宿主机中尽情采集本机所有承载的容器的相应监控数据。本文采用了开源的Docker容器监控方案，Telegraf+Influxdb+Grafana去获取各个容器内部的运行情况。*

1.[Telegraf相关介绍以及实践](https://my.oschina.net/xxbAndy/blog/751330)   
2.[Influxdb相关介绍以及实践](https://my.oschina.net/xxbAndy/blog/751329)   
3.[Grafana相关介绍和实践](https://my.oschina.net/xxbAndy/blog/751328)   
4.TIG架构组成以及原理介绍  
telegraf为数据采集器，通过监控指标的配置将相应的数据采集并存储到influxdb中，进而进行时间序列化，而grafana最终通过influxdb中提供的源数据进行聚合分析和展示。   
telegraf的难点在于理解每个plugin的配置以及采集指标的原理和含义；influxdb的难点在于类SQL语言的优化使用；grafana的难点在于对监控需求以及指标的分析和提取，并通过可视化配置将图标展现。   
5.监控效果图



注意:原创著作，转载请联系作者！

# 2@利用 Telegraf 进行简单的系统监控 - 推酷

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

时间 2017-03-14 23:09:26  [伪架构师](http://www.tuicool.com/sites/EjEFJfN)

原文  [http://blog.fleeto.us/content/li-yong-telegraf-jin-xing-jian-dan-de-xi-tong-jian-kong](http://blog.fleeto.us/content/li-yong-telegraf-jin-xing-jian-dan-de-xi-tong-jian-kong?utm_source=tuicool&utm_medium=referral)

主题 [系统监控](http://www.tuicool.com/topics/11000134)[InfluxDB](http://www.tuicool.com/topics/11030128)

InfluxData 除了广为人知的 InfluxDB 之外，还有几个其他的产品，合称 TICK：

* Telegraf：数据采集
* InfluxDB：数据存储
* Chronograf：数据展现
* Kapacitor：数据分析、告警

在翻看 InfluxDB 的时候偶然发现了这个东西，虽然 Tick 四兄弟捆起来也不够看，不过 Telegraf 足够小巧，而且自动化的可能性更大，更符合目前的做事风格，所以就学习一下。

官宣： The plugin-driven server agent for collecting & reporting metrics.

所以 Telegraf 主要是一个框架，由数据输入、处理、输出三大类插件完成各种功能。Github 的 README.md 中列出了主要插件： **https://github.com/influxdata/telegraf** 。总的来说还是比较丰富的，下面的操作将利用简单的输入插件结合 InfluxDB 输出插件完成一个初步的指标收集过程。

安装

CentOS

生成如下的 repo 文件：

**[influxdb]**

name = InfluxDB Repository - RHEL $releasever

baseurl = https://repos.influxdata.com/rhel/$releasever/$basearch/stable

enabled = 1

gpgcheck = 0

**yum install -y telegraf** 即可完成安装。

Docker

docker pull telegraf

配置

yum 安装后在 **/etc/telegraf** 下会生成一个 **telegraf.conf** 文件。

配置文件中可以使用 "$ENV\_ITEM" 的形式使用环境变量。

global\_tags

这里记录的内容将作为 Tags 保存到 InfluxDB 的每个 Item 中。

agent

这一节内容是数据搜集服务的行为定义。这里暂时无需进行改动

outputs.influxdb

这里用于定义写入的 InfluxDB。

urls = ["http://localhost:8086"]

database = "telegraf"

timeout = "5s"

username = "telegraf"

password = "abcde!@#$%"

urls 参数是一个数组，代表一个集群，如果其中包含多个服务，则每次只会选择其中一台进行写入。

而在 inputs 一节中，缺省启用了很多系统属性，例如磁盘，网络等，这里我们添加一点 http 监控内容：

**[[inputs.http\_response]]**

address = "http://163.com"

response\_timeout = "5s"

method = "GET"

**[[inputs.http\_response]]**

address = "http://sina.com.cn"

response\_timeout = "5s"

method = "GET"

小窍门：可以用 **telegraf -config telegraf.conf -input-filter http\_response -test** 命令，来检查配置的正确性。

配置文件编写完成之后，就可以利用 **systemctl start telegraf** ，启动 telegraf 服务了。

启动之后，Telegraf 会在一定的时间间隔里向 InfluxDB 汇报数据。我们可以在 InfluxDB UI 中利用

**select** \* **from** cpu

这样的语句来查询数据，或者接入 Grafana 等进行展现。

# Telegraf & Kapacitor, 来自Influxdata的套路 - 微店技术团队 – SegmentFault

https://segmentfault.com/a/1190000007513519

InfluxDB推出了的正式版V1.0版本(最新版本为1.1)，随之而来还有Telegraf、Chronograf、Kapacitor等多个产品。InfluxDB也推出了企业版，并推出了InfluxCloud的云服务，这阵势，是要承包指标采集、分析、画图等时序数据库上下游的生意，有点模仿ELK套件的意思，今天我们就来说一下这里面的套路。

**Telegraf**

Telegraf是一个数据采集套件，使用起来跟Collectd、Statsd、Logstash等软件很像。通过plugin来实现数据的input和output。

看着下面长长的一串plugin，感觉非常强大有没有。

**Input Plugins**

* aws cloudwatch
* aerospike
* apache
* bcache
* cassandra
* ceph
* chrony
* consul
* conntrack
* couchbase
* couchdb
* disque
* dns query time
* docker
* dovecot
* elasticsearch
* exec (generic executable plugin, support JSON, influx, graphite and nagios)
* filestat
* haproxy
* hddtemp
* http\_response
* httpjson (generic JSON-emitting http service plugin)
* influxdb
* ipmi\_sensor
* iptables
* jolokia
* leofs
* lustre2
* mailchimp
* memcached
* mesos
* mongodb
* mysql
* net\_response
* nginx
* nsq
* nstat
* ntpq
* phpfpm
* phusion passenger
* ping
* postgresql
* postgresql\_extensible
* powerdns
* procstat
* prometheus
* puppetagent
* rabbitmq
* raindrops
* redis
* rethinkdb
* riak
* sensors
* snmp
* snmp\_legacy
* sql server (microsoft)
* twemproxy
* varnish
* zfs
* zookeeper
* win\_perf\_counters (windows performance counters)
* sysstat
* system
* cpu
* mem
* net
* netstat
* disk
* diskio
* swap
* processes
* kernel (/proc/stat)
* kernel (/proc/vmstat)

**Service plugins:**

* http\_listener
* kafka\_consumer
* mqtt\_consumer
* nats\_consumer
* nsq\_consumer
* logparser
* statsd
* tail
* tcp\_listener
* udp\_listener
* webhooks
* filestack
* github
* mandrill
* rollbar

**Processor Plugins**

* printer

**Aggregator Plugins**

* minmax

**Output Plugins**

* influxdb
* amon
* amqp
* aws kinesis
* aws cloudwatch
* datadog
* file
* graphite
* graylog
* instrumental
* kafka
* librato
* mqtt
* nats
* nsq
* opentsdb
* prometheus
* riemann

**安装与使用**

**安装**

笔者使用了最简单的RPM方式安装

**curl** -LO https://dl.influxdata.com/telegraf/releases/telegraf-1.1.1.x86\_64.rpm

rpm ivh telegraf-1.1.1.x86\_64.rpm

**启动**

standalone方式启动，可以指定需要的plugin，是不是有点像logstash呢？

telegraf --config telegraf.conf -input-filter cpu:mem -output-filter influxdb

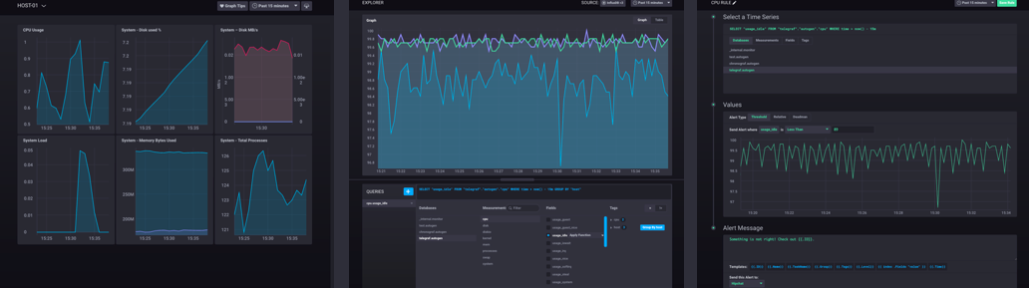
或者以后台Deamon方式启动。首先编辑默认的配置文件

**vim** /etc/telegraf/telegraf.conf

设置好plugin之后，就可以启动服务了

service telegraf **start**

**Chronograf**

Chronograf是一款画图软件，整体风格与Grafana十分相似，来贴几张界面的图  
当然比起已经比较成熟的Grafana来说，还差一些。

**Kapacitor**

Kapacitor是一款时序数据分析、处理的软件。可以周期性将InfluxDB中的数据汇总、处理后再输出到InfluxDB当中，或者告警（支持Email、HTTP、TCP、 HipChat, OpsGenie, Alerta, Sensu, PagerDuty, Slack等多种方式）

Kapacitor的配置可以使用一种叫做 TICKscript 的DSL语言来书写，下面是一个配置的例子

stream

|**from**()

.measurement('cpu\_usage\_idle')

.groupBy('host')

|window()

.period(1m)

.every(1m)

|mean('value')

|eval(lambda: 100.0 - "mean")

.**as**('used')

|alert()

.message('{{ .Level}}: {{ .Name }}/{{ index .Tags "host" }} has high cpu usage: {{ index .Fields "used" }}')

.warn(lambda: "used" > 70.0)

.crit(lambda: "used" > 85.0)

// Send alert to hander of choice.

// Slack

.slack()

.channel('#alerts')

// VictorOps

.victorOps()

.routingKey('team\_rocket')

// PagerDuty

.pagerDuty()

我们就以上例来讲解配置

**stream**

stream,一段配置的开始，相当于function,也可以用来赋值

**var** errors = stream

|**from**()

.measurement('errors')

**from**

from, 定义数据的来源，也可用QUERY来直接书写SQL

|query('''

SELECT mean("value")

FROM "telegraf"."default".cpu\_usage\_idle

WHERE "host" = 'serverA'

''')

**windows**

window, 定义时序数据的时间范围

|window()

.period(10m)

.every(1m)

表示每分钟执行一次，取10分钟内的指标，也可以使用cron来指定执行时间

**mean**

mean表示取中位数，当然还有Derivative(增值)、Difference（差值）等很多方法，用过Grafana的同学应该比较熟悉

**eval**

eval可以通过自定义的函数对数据进行加工，这里将100 - "mean"的结果定义为used

|eval(**lambda**: 100.0 - "mean")

.**as**('used')

**alert**

alert就是告警的方法了。

id、message分别是告警的标题和内容

可以设置多个级别的告警阀值（OK、INFO、WARNING、CRITICAL）

**stream**

.groupBy('service')

|alert()

.id('kapacitor/{{ index .Tags "service" }}')

.message('{{ .ID }} is {{ .Level }} value:{{ index .Fields "value" }}')

.info(lambda: "value" > 10)

.warn(lambda: "value" > 20)

.crit(lambda: "value" > 30)

.post("http://example.com/api/alert")

.post("http://another.example.com/api/alert")

.tcp("exampleendpoint.com:5678")

.email('oncall@example.com')

最后就是设置报警的方式了，部分参数可以配置在启动配置文件中

**启动**

写完TICKscript后，将之保存到文件，例如cpu\_alert.tick  
之后用以下命令启动程序

# Define the task (assumes cpu data is in db 'telegraf')

kapacitor define \

cpu\_alert \

-type stream \

-dbrp telegraf.default \

-tick ./cpu\_alert.tick

# Start the task

kapacitor enable cpu\_alert