目錄

简介 1.1

基本组件 2.1

- InfluxDB 2.1.1
- InfluxQL 2.1.2
- Telegraf 2.1.3
- Grafana简介 2.1.4
- Grafana Tutorial 2.1.5
- Grafana 配置 2.1.6
- Grafana 异常 2.1.7

Linux监控 3.1

- Linux监控指标 3.1.1
- 基本信息监控 3.1.2
- CPU监控 3.1.3
- 内存监控 3.1.4
- 磁盘监控 3.1.5
- 网络监控 3.1.6

web server监控 4.1

- apache/httpd监控 4.1.1
- http响应时间监控 4.1.2

Docker监控 5.1

- cAdvisor 5.1.1
- cAdvisor+InfluxDB 5.1.2

监控系统实践

本书的gitbook地址是: https://frank6866.gitbooks.io/monitor/content/

主要内容

- 开源分布式监控与告警系统(Telegraf+InfluxDB+Grafana)的运维笔记
- Linux监控
- Docker监控

About me

本人运维开发一枚,专注于IaaS及Paas相关技术,欢迎大家拍砖,
frank6866@vip.sina.com

基本组件

监控系统一般如下几个模块：

- 数据采集
- 数据存储
- 数据展示
- 告警

本书介绍了开源的监控方案:

采集数据(Telegraf) -> 存储数据(InfluxDB) -> 显示数据(Grafana), 并使用Kapacitor做告警

InfluxDB

InfluxDB是一个开源的、分布式时序、事件和指标数据库。InfluxDB使用Go语言编写，着力于高性能地查询与存储时序型数据，无需外部依赖。

本文使用InfluxDB版本为V1.2，官网文档地址。

InfluxDB有以下特点:

- schemaless(无结构): 可以是任意数量的列
- scalable: 可扩展的
- 有很多聚合函数: min, max, sum, count, mean等

概念

名称	解释
user	在InfluxDB中有两种类型的user：Admin用户对所有的数据库都有读和写的权限，并且可以执行管理员才能执行的查询和用户管理命令；非admin用户可以对某个或者某几个数据库，具有读或者写或者读写的权限。
schema	在InfluxDB中数据是如何存储和组织的。InfluxDB的基础schema包括databases、retention policies、series、measurements、tag keys、tag values和field keys。
field	InfluxDB数据结构中的key-value pair(键值对)，用来记录元数据(metadata)和实际的数据值。InfluxDB中的数据结构需要field，并且field不能被索引。针对field的值的查询将会扫描满足特定时间需求的所有点。所以，和tag相比，field没有tag的查询效率高。
field key	field key是key-value pair中的key部分，filed key是字符串，field key存储的是元数据。
field value	field value是key-value pair中的value部分，field value是实际的数据，可以是字符串、浮点型、整型或者布尔型。一个field value总是和一个timestamp关联。field value不会被索引，对field value的查询将会扫描满足特定时间需求的所有点，是很不高效的。(提示：在查询中使用tag value而不是field value)
field set	一个point上很多key-value pair(field)的集合。
	点，series中很多field的集合，相当于关系数据库中的一行数据。point是由timestamp、fields和tags组成。每一个point通

point	过它的series和timestamp唯一标识。在一个series中，对于一个timestamp我们不能存储多个point。
function	InfluxQL中的aggregations、selectors和transformations。
continuous query(CQ，持续查询)	continuous query在数据库中自动和周期地运行。continuous query在SELECT中需要一个aggregation(聚合函数)，并且必须包含**GROUP BY time()**。
database(数据库)	数据库是一个逻辑的容器，包括：users、retention policies、continuous queries和time series data。
measurement	相当于关系数据库中的table,measurement是tag、field和time的容器。对于measurement来说,field是必须的,并且不能根据field来排序。
retention policy	保留策略,用于决定要保留多久的数据,保存几个备份,以及集群的策略。对于每一个database来说，只有一个retention policy。当我们创建一个database时，InfluxDB会自动创建一个叫**autogen**的retention policy，其duration为无穷，replication factor是1，shard group duration是7天。
duration(区间)	duration是retention policy的一个属性，用来决定InfluxDB将数据存储多久，比duration更老的数据将会被自动删掉。
replication factor(复制系数)	replication factor是retention policy的一个属性，用来决定InfluxDB集群将数据存储多少个副本。（Replication factor不支持单节点的集群）
series	一个series可以理解为一个曲线图，这个曲线图上的所有点的如下属性必须是一样的：measurement、tag set和retention policy，如果有一个不一样，就是另外一个series。举个例子，在一个measurement中，retention policy都是一样的，tag(k)的取值共有f(k)，那么在这个measurement中，共有f(1) *f(2)* ... * f(k)个series。注意，field set不是series的标识符，不能用来区别series。
tag	tag是InfluxDB中记录元数据的的key-value pair。tag是可选的,tag会被索引,所以通常tag被用来存储经常被查询的的元数据。tag是以字符串的形式存放的。
tag key	tag的key，使用字符串标识；tag的key被索引了，所以查询起来很高效。
tag value	tag的value，使用字符串标识；tag的value被索引了，所以查询起来很高效。
tag set	一个point上tag key和tag value的结合
timestamp	和一个point关联的日期和时间，在InfluxDB中，所有的时间都是UTC时间。
tsm(Time	

Structured Merge tree)	TSM允许更高的数据压缩率和更高的读写吞吐。
aggregation	聚合函数，聚合函数对一组point做聚合并返回一个聚合值(比如对一组point求平均值的函数mean)。
transformation	transformation是InfluxQL的一类函数，它可以计算一些特定的point，返回一个或者一组值；但是transformation函数不返回aggregation值。
batch	批处理，一组point通过换行符分隔，这组point可以通过HTTP请求被一起提交到数据库中。这样可以让HTTP API更加高效。InfluxDB推荐batch size在5000-10000之间；针对不同的使用场景也可以调整。
identifier	数据库、field key、measurement name、retention policy name、tag keys等的唯一标识符。
line protocol	将points写入到InfluxDB时，基于文本的协议。
server	一个运行InfluxDB的机器(物理机或者虚拟机)。对于每一个server来说，只能有一个InfluxDB进程。
node	一个独立的**influxd**进程。
metastore	包含InfluxDB运行状态的系统信息。metastore包含user、databases、retention policies、shard metadata、continuous queries和subscriptions
now()	服务器的时间戳，精确到nanosecond(纳秒)
values per second	一个InfluxDB服务的监控指标，表示数据被持久化到InfluxDB中的速率。values per second的计算方法为：每秒写入的point个数乘以每个point的field的个数。比如，某个point有4个field，每秒可以执行10次批处理操作，每次批处理操作的大小为5000；那么values per second = 4 *10* 5000 = 200,000
wal(Write Ahead Log)	对最近写入的数据的临时缓存。为了减少存储文件被访问的频率，InfluxDB将新的point缓存到WAL中，直到WAL的的缓存空间满了或者缓存的point老化而触发写到磁盘文件中去的操作。WAL中的point可以被查询，在系统重启后也存在；但重启后，InfluxDB进程启动了，WAL中的所有point必须被flush到磁盘上，InfluxDB才能接收新的写请求。

安装

CentOS7

在CentOS7上部署InfluxDB可以使用我写的ansible role,不过为了熟悉InfluxDB,建议还是手工安装一遍。

创建仓库文件

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdata.repo
[influxdata]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basea
rch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key
EOF
```

install, start and enable service:

```
# yum install influxdb
# systemctl start influxdb
# systemctl enable influxdb
```

> CentOS7上InfluxDB的配置文件路径是: **/etc/influxdb/influxdb.conf**

macOS

```
➜ brew install influxdb
➜ brew services start influxdb
```

> macOS上InfluxDB的配置文件路径是: **/usr/local/etc/influxdb.conf**

配置

常用端口

- TCP 8083: web ui使用的端口

- TCP 8086: HTTP API使用的端口
- TCP 8088: remote backups and restores

admin ui

在1.3版本中将会移除admin ui功能，不要再依赖这个功能。

HTTPS配置

生成证书

这里用的是在macOS上自建的CA，用来签发证书。(如何搭建CA，参考我的搭建CA这篇文章)

```
➜ mkdir /tmp/process_csr/influxdb/
➜ openssl genrsa -out /tmp/process_csr/influxdb/influxdb.key 204
8
➜ openssl req -new -batch -days 3650 -subj /CN=influxdb -key /tm
p/process_csr/influxdb/influxdb.key -reqexts SAN -config <(cat /
usr/local/etc/openssl/openssl.cnf <(printf "[SAN]\nsubjectAltNam
e=IP:192.168.168.201,IP:192.168.168.202,IP:192.168.168.203,DNS:l
ocalhost")) -out /tmp/process_csr/influxdb/influxdb.csr
➜ openssl ca -in /tmp/process_csr/influxdb/influxdb.csr  -extens
ions SAN -config <(cat /usr/local/etc/openssl/openssl.cnf <(prin
tf "[SAN]\nsubjectAltName=IP:192.168.168.201,IP:192.168.168.202,
IP:192.168.168.203, DNS:localhost"))  -out /tmp/process_csr/infl
uxdb/influxdb.crt
```

将生成的证书及key文件拷贝到InfluxDB服务器上。

修改配置

编辑/etc/influxdb/influxdb.conf文件的[http]部分，修改三个配置项:

- https-enabled: 表示是否启用https(默认为false)
- https-certificate: 证书文件路径

- https-private-key: 证书对应的密钥文件路径

修改完后，如下:

```
[http]
...
  # Determines whether HTTPS is enabled.
    https-enabled = true

  # The SSL certificate to use when HTTPS is enabled.
    https-certificate = "/etc/pki/influxdb/influxdb.crt"

  # Use a separate private key location.
    https-private-key = "/etc/pki/influxdb/private/influxdb.key"
...
```

修改完后重启InfluxDB:

```
# systemctl restart influxdb
```

测试HTTPS

```
# influx -ssl
Connected to https://localhost:8086 version 1.2.2
InfluxDB shell version: 1.2.2
>
```

-ssl选项表示Use https for requests.

可能出现的问题及解决方法:

- 如果出现"x509: certificate is valid for xxx, not localhost"的问题，确认是否在证书的SAN中添加了DNS:localhost
- 如果出现"x509: certificate signed by unknown authority"，确认CA的证书被系统信任((CentOS7或macOS中如何添加对CA证书的信任，参考我的搭建CA这篇文章)).

authentication

参考官网文档

编辑/etc/influxdb/influxdb.conf文件的[http]部分，修改如下配置项:

- auth-enabled: 是否启用认证(默认是false)

修改完后，如下:

```
......
[http]
......
  # Determines whether HTTP authentication is enabled.
    auth-enabled = true
......
```

修改完后重启InfluxDB:

```
# systemctl restart influxdb
```

启用认证后，如果没有admin用户，执行操作时会报错，如下:

```
# influx -ssl
Connected to https://localhost:8086 version 1.2.2
InfluxDB shell version: 1.2.2
> show databases;
ERR: error authorizing query: create admin user first or disable
 authentication
Warning: It is possible this error is due to not setting a datab
ase.
Please set a database with the command "use <database>".
```

首先创建一个有admin权限用户，比如名为admin的用户(名为admin的用户不是必须的，具有admin权限的用户可以是其他名字):

```
> CREATE USER admin WITH PASSWORD 'change_me' WITH ALL PRIVILEGE
S
```

在CLI中使用认证

可以在influx命令行中使用auth命令认证(比较麻烦):

```
> auth
username: admin
password:
> show databases
name: databases
name
----
_internal
```

可以在启动influx时使用认证(通过history可以看到密码，不安全):

```
# influx -ssl -username admin -password change_me
Connected to https://localhost:8086 version 1.2.2
InfluxDB shell version: 1.2.2
> show databases
name: databases
name
----
_internal
```

通过环境变量的方式(方便，推荐的方式，也便于管理多套influxdb环境，比如开发环境和测试环境可以各写一个rc文件，用的时候source一下rc文件就可以了，用登录到机器上就可以操作；生产环境建议还是老老实实登录到机器上去玩，因为可能会以为操作的是测试环境的库，而实际上操作的是生产的库，避免误操作):

```
# vi ~/influxdb.rc
export INFLUX_USERNAME=admin
export INFLUX_PASSWORD=change_me
```

可以不用输入账号和密码，如下:

```
# influx -ssl
Connected to https://localhost:8086 version 1.2.2
InfluxDB shell version: 1.2.2
> show databases;
name: databases
name
----
_internal
```

example

在mac上使用influx客户端连接192.168.168.201上的influxdb(配置了ssl):

```
# vi 192.168.168.201.rc
export INFLUX_HOST=192.168.168.201
export INFLUX_USERNAME=admin
export INFLUX_PASSWORD=change_me
```

测试了一下，想把host也放到rc文件中，这样切换不同环境就更方便了:

```
➜ influx
Failed to connect to http://localhost:8086: Get http://localhost
:8086/ping: dial tcp [::1]:8086: getsockopt: connection refused
Please check your connection settings and ensure 'influxd' is ru
nning.
```

但是好像不支持，不过在命令行里面使用-host选项还是可以的，如下:

```
➜ influx -ssl -host 192.168.168.201
Connected to https://192.168.168.201:8086 version 1.2.2
InfluxDB shell version: v1.2.0
> SHOW GRANTS FOR frank6866
database privilege
-------- ---------
telegraf READ
```

刚开始这里有一个疑问的地方，在192.168.168.201的influxdb上，我配置了HTTPS，在influx的命令行选项里面我没有看到ca相关的选项，但是上面的命令连接成功了。想到在192.168.168.201的influxdb上的证书是用我在macOS上自建的CA签发的，CA的根证书我已经信任过了，所以上面的命令可以连接到192.168.168.201的influxdb上。如果在macOS取消对自建CA根证书的信任，在运行上面的命令，如下:

```
➜ influx -ssl -host 192.168.168.201
Failed to connect to https://192.168.168.201:8086: Get https://1
92.168.168.201:8086/ping: x509: certificate signed by unknown au
thority
Please check your connection settings and ensure 'influxd' is ru
nning.
```

所以如果碰到"x509: certificate signed by unknown authority"类似的问题，解决思路在系统中添加对证书的信任。(CentOS7或macOS中如何添加对CA证书的信任，参考我的搭建CA这篇文章)

InfluxQL

使用下面的命令进入InfluxDB交互模式

```
# influx
```

元数据管理

用户管理

首先创建一个有admin权限用户(加上WITH ALL PRIVILEGES)，比如名为admin的用户(名为admin的用户不是必须的，具有admin权限的用户可以是其他名字):

```
> CREATE USER admin WITH PASSWORD 'change_me' WITH ALL PRIVILEGE
S
```

创建一个普通用户

```
> CREATE USER frank6866 WITH PASSWORD 'change_me'
```

查看所有用户

```
> show users
user      admin
----      -----
admin     true
frank6866 false
```

给frank6866用户赋予admin权限(GRANT ALL PRIVILEGES):

```
> GRANT ALL PRIVILEGES TO frank6866
> show users
user      admin
----      -----
admin     true
frank6866 true
```

撤销frank6888用户的管理员权限(REVOKE ALL PRIVILEGES):

```
> REVOKE ALL PRIVILEGES from frank6866
> show users
user      admin
----      -----
admin     true
frank6866 false
```

给frank6866在telegraf库中所有权限(所有权限是ALL，读是READ，写是WRITE):

```
> GRANT ALL ON telegraf TO frank6866
> SHOW GRANTS FOR frank6866
database privilege
-------- ---------
telegraf ALL PRIVILEGES
```

撤销frank6866在telegraf库中的写权限

```
> REVOKE WRITE ON telegraf FROM frank6866
> SHOW GRANTS FOR frank6866
database privilege
-------- ---------
telegraf READ
```

可以看到，如果frank6866用户在某个库中，原本是ALL的权限，被收回WRITE权限后，还剩下READ权限。

修改frank6866用户的密码

```
> SET PASSWORD FOR "frank6866" = 'change@me'
```

数据库操作

显示所有数据库

```
> show databases
name: databases
name
----
_internal
telegraf
```

创建一个名为db_test的数据库

```
> create database db_test
```

删除名为db_test的数据库

```
> drop database db_test
```

切换到名为telegraf的数据库

```
> use telegraf
```

表操作

show measurements

显示telegraf库中的所有表

```
> show measurements
name: measurements
name
----
apache
ceph
cpu
......
rabbitmq_node
rabbitmq_overview
rabbitmq_queue
swap
system
```

show field keys

查看表的字段

查看telegraf库中所有表的所有字段(如果当前use的库是telegraf，可以省略on条件):

```
> show field keys on telegraf
name: rabbitmq_overview
fieldKey           fieldType
--------           ---------
channels           integer
connections        integer
consumers          integer
exchanges          integer
messages           integer
messages_acked     integer
messages_delivered integer
messages_published integer
messages_ready     integer
messages_unacked   integer
queues             integer

name: rabbitmq_queue
fieldKey                  fieldType
--------                  ---------
consumer_utilisation      float
consumers                 integer
idle_since                string
memory                    integer
message_bytes             integer
message_bytes_persist     integer
......
```

查看telegraf库中ceph表的所有字段(如果当前use的库是telegraf，可以省略on条件):

```
> show field keys on telegraf from ceph
name: ceph
fieldKey                                    fieldType
--------                                    ---------
accept_timeout                              float
activating_latency.avgcount                 float
activating_latency.sum                      float
active_latency.avgcount                     float
active_latency.sum                          float
agent_evict                                 float
agent_flush                                 float
agent_skip                                  float
agent_wake                                  float
apply_latency.avgcount                      float
apply_latency.sum                           float
backfilling_latency.avgcount                float
backfilling_latency.sum                     float
begin                                       float
begin_bytes.avgcount                        float
begin_bytes.sum                             float
begin_keys.avgcount                         float
begin_keys.sum                              float
begin_latency.avgcount                      float
begin_latency.sum                           float
buffer_bytes                                float
bytes                                       float
bytes_dirtied                               float
bytes_wb                                    float
...
```

show tag keys

显示tag keys。命令格式为:

```
show_tag_keys_stmt = "SHOW TAG KEYS" [on_clause] [ from_clause ]
 [ where_clause ]
                     [ limit_clause ] [ offset_clause ] .
```

显示当前数据库的所有表上的tag key(如果命令操作的是当前库，on条件可以省略；如果要显示db_test数据库上所有的tag的命令是show tag keys on db_test)

```
> show tag keys
name: apache
tagKey
------
host
port
server

name: ceph
tagKey
------
collection
host
id
type

......
```

显示当前数据库上ceph表中的所有tags:

```
> show tag keys from ceph
name: ceph
tagKey
------
collection
host
id
type
```

显示当前数据库上ceph表中host这个tag为ceph-1的所有tag:

```
> show tag keys from ceph where host="ceph-1"
name: ceph
tagKey
------
collection
host
id
type
```

SHOW TAG VALUES

显示tag values，命令格式为：

```
show_tag_values_stmt = "SHOW TAG VALUES" [on_clause] [ from_clau
se ] with_tag_clause [ where_clause ]
                       [ limit_clause ] [ offset_clause ] .
```

show tag values必须至少指定一个key的名称，可以查询某个或某几个key的数据库所有表中的tag value，也可以查询某个表中的tag values。

显示tag为host的所有值(如果不指定表，会列出所有表中tag的key为host的所有value)

```
> SHOW TAG VALUES WITH KEY = "host"
name: apache
key  value
---  -----
host openstack-1

name: ceph
key  value
---  -----
host ceph-1

name: cpu
key  value
---  -----
host 10.12.10.43
host ceph-1
......
```

删除measure

```
DROP MEASUREMENT <measurement_name>
```

数据管理

ref

- https://docs.influxdata.com/influxdb/v1.2/concepts/crosswalk/
- https://docs.influxdata.com/influxdb/v1.2/query_language/data_exploration/

注意事项

- 不支持JOIN操作
- 如果标识符中包含除了[A-z,0-9,_]之外的字符串或者它们以数字开头，或者它们是InfluxDB的关键字，我们必须使用双引号括起来

- WHERE条件中的值不能使用双引号
- 每一个表中都有一个名为time的字段，可以在where中使用，比如**WHERE time > now() - 7d**
- group by后面可以使用tag key或者time interval，不能使用filed key
- select查询出来的都是epoch类型的时间戳，如果想看到的是字符串类型的时间，比如，可以在cli中使用**precision rfc3339**命令，切换时间显示格式。常用的格式包括(rfc3339, h, m, s, ms, u or ns).Unix时间戳的长度是10位整数(**date +%s** 命令可以查看当前时间戳)，InfluxDB默认的显示格式是ns，也就是19位整数.

SELECT

用法:

```
> SELECT <field_key>[,<field_key>,<tag_key>] FROM <measurement_name>[,<measurement_name>]
```

注意：

- SELECT语句中可以使用field和tag，但是如果有tag，必须至少包含一个field
- 如果表中field和tag的名称相同，可以使用 **"filed_name"::field** 或者 **"tag_name"::tag** 来区分
- 如果只是在SELECT中指定tag key，不会报错，会返回空的数据

WHERE

用法:

```
> SELECT_clause FROM_clause WHERE <conditional_expression> [(AND|OR) <conditional_expression> [...]]
```

注意事项:

- where条件中的字符串类型的值必须用单引号括起来；如果使用双引号，不会报错，也不会得到任何值。
- where条件中float类型的值一定不能使用单引号，否则查不到数据。

WHERE条件中支持对filed、tag和timestamp进行操作。

field

格式:

```
field_key <operator> ['string' | boolean | float | integer]
```

支持的操作符:

```
=   equal to
<> not equal to
!= not equal to
>   greater than
>= greater than or equal to
<   less than
<= less than or equal to
```

tag

格式:

```
tag_key <operator> ['tag_value']
```

支持的操作符:

```
=   equal to
<> not equal to
!= not equal to
```

timestamps

InfluxDB中的所有时间都是UTC时间。

注意：

- 对于SELECT语句，默认的时间范围是 1677-09-21 00:12:43.145224194 到 2262-04-11T23:47:16.854775806Z。
- 对于带有GROUP BY time()的SELECT语句，默认的时间范围是1677-09-21 00:12:43.145224194 UTC 到 now()

demo

查询CPU的load15大于10的数据(其中host是一个tag，代表主机名):

```
> select load15, host from system where "load15" > 10
name: system
time                load15 host
----                ------ ----
1489474580000000000 10.05  nl-cloud-openstack-1
1489474590000000000 10.07  nl-cloud-openstack-1
1489474600000000000 10.09  nl-cloud-openstack-1
1489474610000000000 10.14  nl-cloud-openstack-1
1489474620000000000 10.08  nl-cloud-openstack-1
```

GROUP BY

- group by需要和聚合函数结合使用
- group by后面可以使用tag key或者time interval，不能使用filed key.

group by tag key

格式:

```
> SELECT_clause FROM_clause [WHERE_clause] GROUP BY [* | <tag_ke
y>[,<tag_key]]
```

如果想对所有的tag key做group by，可以使用GROUP BY *

比如想按照主机分组，查询CPU的load15的平均值

```
> select mean(load15) from system group by host
name: system
tags: host=nl-cloud-ceph-1
time mean
---- ----
0    0.09120040939667792

name: system
tags: host=nl-cloud-ceph-2
time mean
---- ----
0    0.11674878345502718
......
```

- 可以看到，group by之后，time的值是0.在InfluxDB中， epoch(时间点) 0 (1970-01-01T00:00:00Z) 一般表示空的时间戳。

group by time intervals

格式:

```
> SELECT <function>(<field_key>) FROM_clause WHERE <time_range>
GROUP BY time(<time_interval>),[tag_key] [fill(<fill_option>)]
```

查询'2017-03-23T00:00:00Z'到'2017-03-23T00:05:00Z'之间的数据:

```
> select load15 from system where host='nl-cloud-ceph-1' and tim
e >= '2017-03-23T00:00:00Z' and time <= '2017-03-23T00:05:00Z'
name: system
time                 load15
----                 ------
2017-03-23T00:00:00Z 0.05
2017-03-23T00:00:10Z 0.05
2017-03-23T00:00:20Z 0.05
2017-03-23T00:00:30Z 0.05
2017-03-23T00:00:40Z 0.05
2017-03-23T00:00:50Z 0.05
2017-03-23T00:01:00Z 0.05
2017-03-23T00:01:10Z 0.05
2017-03-23T00:01:20Z 0.05
2017-03-23T00:01:30Z 0.05
2017-03-23T00:01:40Z 0.05
2017-03-23T00:01:50Z 0.05
2017-03-23T00:02:00Z 0.05
2017-03-23T00:02:10Z 0.05
2017-03-23T00:02:20Z 0.05
2017-03-23T00:02:30Z 0.05
2017-03-23T00:02:40Z 0.05
2017-03-23T00:02:50Z 0.05
2017-03-23T00:03:00Z 0.05
2017-03-23T00:03:10Z 0.05
2017-03-23T00:03:20Z 0.05
2017-03-23T00:03:30Z 0.05
2017-03-23T00:03:40Z 0.05
2017-03-23T00:03:50Z 0.05
2017-03-23T00:04:00Z 0.05
2017-03-23T00:04:10Z 0.05
2017-03-23T00:04:20Z 0.05
2017-03-23T00:04:30Z 0.05
2017-03-23T00:04:40Z 0.05
2017-03-23T00:04:50Z 0.05
2017-03-23T00:05:00Z 0.05
```

查询'2017-03-23T00:00:00Z'到'2017-03-23T00:05:00Z'之间的数据，按照1m进行分组，对每个分组求平均值

```
> select mean(load15) from system where host='nl-cloud-ceph-1' a
nd time >= '2017-03-23T00:00:00Z' and time <= '2017-03-23T00:05:
00Z' group by time(1m)
name: system
time                  mean
----                  ----
2017-03-23T00:00:00Z 0.049999999999999996
2017-03-23T00:01:00Z 0.049999999999999996
2017-03-23T00:02:00Z 0.049999999999999996
2017-03-23T00:03:00Z 0.049999999999999996
2017-03-23T00:04:00Z 0.049999999999999996
2017-03-23T00:05:00Z 0.05
```

fill

fill是可选的，如果使用group by time(...)没有返回数据，使用什么方式来替换这个值。fill的选项包括:

- linear: 线性插入这个值
- none: 没有数据也没有time interval
- null: 没有数据，有time interval。这是默认的行为
- previous: 如果有time interval没有数据，使用上一个数据

> 注意: fill必须在group by子句的最后面

函数

https://docs.influxdata.com/influxdb/v1.2/query_language/functions/#aggregations

Aggregations(聚合函数)

聚合函数只能使用field key，而不能使用tag key作为参数。

COUNT

COUNT函数的功能是返回某个字段非空值的个数。

比如我想查看system表中，load1这个字段一共有多少个非空的值:

```
> select count(load1) from system;
name: system
time count
---- -----
0    2490856
```

可以看到一共有2490856个非空值

比如我想查看system表中，load1这个字段一共有多少个非空的值，并且按照host这个tag进行分组:

```
> select count(load1) from system group by host;
name: system
tags: host=nl-cloud-ceph-1
time count
---- -----
0    81126

name: system
tags: host=nl-cloud-ceph-2
time count
---- -----
0    81254

name: system
tags: host=nl-cloud-ceph-3
time count
---- -----
0    51619
......
```

DISTINCT

DISTINCT返回一个字段不重复的值的列表

INTEGRAL

1.2版本中未实现。

MEAN

返回某个字段的平均值，字段必须是int64或float64的。

格式:

```
> SELECT MEAN(<field_key>) FROM <measurement_name> [WHERE <stuff
>] [GROUP BY <stuff>]
```

返回过去一天内nl-cloud-ceph-1主机load15的平均值

```
> select mean(load15) from system where host='nl-cloud-ceph-1' a
nd time >= now() - 1d
name: system
time                           mean
----                           ----
2017-03-26T07:06:24.945676036Z 0.07806597222222564
```

MEDIAN

median是中位数。

定义引用自wiki: 对于一组有限个数的数据来说，它们的中位数是这样的一种数：这群数据里的一半的数据比它大，而另外一半数据比它小。 计算有限个数的数据的中位数的方法是：把所有的同类数据按照大小的顺序排列。 如果数据的个数是奇数，则中间那个数据就是这群数据的中位数；如果数据的个数是偶数，则中间那2个数据的算术平均值就是这群数据的中位数。

```
> select median(load15) from system where host='nl-cloud-ceph-1'
 and time >= now() - 10d
name: system
time                           median
----                           ------
2017-03-17T07:25:54.355346846Z 0.06
```

MODE

返回某个key出现频率最高的value

```
> select mode(tcp_currestab) from net where host='nl-cloud-ceph-
1' and time >= now() - 10d
name: net
time                           mode
----                           ----
2017-03-17T07:30:18.948334757Z 242
```

表示242在tcp_currestab这个filed中出现的频率最高

SPREAD

计算最大值和最小值之间的差距

比如计算一天内最大连接数和最小连接数之间的差异:

```
> select spread(tcp_currestab) from net where host='nl-cloud-cep
h-1' and time >= now() - 1d
name: net
time                           spread
----                           ------
2017-03-26T07:42:07.905201326Z 6
```

通过下面的max可以看到最大是192，min可以看到最小是186，相减正好是6.

也可以按照6小时做一次分组，那么可以得到24/6+1=5组数据:

```
> select spread(tcp_currestab) from net where host='nl-cloud-cep
h-1' and time >= now() - 1d group by time(6h)
name: net
time                 spread
----                 ------
2017-03-26T06:00:00Z 2
2017-03-26T12:00:00Z 1
2017-03-26T18:00:00Z 1
2017-03-27T00:00:00Z 4
2017-03-27T06:00:00Z 5
```

STDDEV

标准差是方差(平方差)的平方根，反应的是数据离散的程度，越大数据之间的差异越大。

```
> select stddev(tcp_currestab) from net where host='nl-cloud-cep
h-1' and time >= now() - 1d group by time(6h)
name: net
time                 stddev
----                 ------
2017-03-26T06:00:00Z 0.23537934230121352
2017-03-26T12:00:00Z 0.08838576993946781
2017-03-26T18:00:00Z 0.23267329099713924
2017-03-27T00:00:00Z 0.18383302430318563
2017-03-27T06:00:00Z 0.6003860353212642
```

SUM

求和

Selectors(选择函数)

FIRST

获取第一个值

返回过去一天内nl-cloud-ceph-1主机已建立网络连接数的第一个值

```
> select first(tcp_currestab) from net where host='nl-cloud-ceph
-1' and time >= now() - 1d
name: net
time                  first
----                  -----
2017-03-26T07:35:30Z 187
```

LAST

获取最后一个值

返回过去一天内nl-cloud-ceph-1主机已建立网络连接数的最后一个值

```
> select last(tcp_currestab) from net where host='nl-cloud-ceph-
1' and time >= now() - 1d
name: net
time                  last
----                  ----
2017-03-27T07:33:40Z 187
```

MAX

获取最大值

返回过去一天内nl-cloud-ceph-1主机已建立网络连接数的最大值

```
> select max(tcp_currestab) from net where host='nl-cloud-ceph-1
' and time >= now() - 1d
name: net
time                  max
----                  ---
2017-03-27T06:50:30Z 192
```

MIN

获取最大值

返回过去一天内nl-cloud-ceph-1主机已建立网络连接数的最小值

```
> select min(tcp_currestab) from net where host='nl-cloud-ceph-1
' and time >= now() - 1d
name: net
time                  min
----                  ---
2017-03-26T07:47:20Z 186
```

TOP

获取最大的N个值

返回过去一天内nl-cloud-ceph-1主机已建立网络连接数的的最大的3个值

```
> select top(tcp_currestab, 3) from net where host='nl-cloud-cep
h-1' and time >= now() - 1d
name: net
time                  top
----                  ---
2017-03-27T06:50:30Z 192
2017-03-27T06:50:40Z 192
2017-03-27T06:50:50Z 192
```

BOTTOM

获取最小的N个值

返回过去一天内nl-cloud-ceph-1主机已建立网络连接数的的最小的3个值

```
> select bottom(tcp_currestab, 3) from net where host='nl-cloud-
ceph-1' and time >= now() - 1d
name: net
time                 bottom
----                 ------
2017-03-26T07:47:20Z 186
2017-03-26T07:47:30Z 186
2017-03-26T07:47:40Z 186
```

PERCENTILE

格式: PERCENTILE(key, N)

PERCENTILE可以返回key对应value前N%的值(N的取值范围是0-100):

- PERCENTILE(,100) 就是MAX()
- PERCENTILE(,0) 就是MIN()
- PERCENTILE(,50) 就是MEDIAN()

比如，查询CPU的load1中大于80%其他值的值是多少。

返回过去一天内nl-cloud-ceph-1主机已建立网络连接数中大于80%其他值的值是多少。

```
> select percentile(tcp_currestab, 80) from net where host='nl-c
loud-ceph-1' and time >= now() - 1d
name: net
time                 percentile
----                 ----------
2017-03-27T02:50:30Z 188
```

查询出来的结果是188，表示188会大于查询的数据集中80%的值。

SAMPLE

随机返回一些数据点

> SELECT SAMPLE(,) FROM_clause [WHERE_clause] [GROUP_BY_clause]

```
> select sample(load1, 3) from system where host='nl-cloud-opens
tack-1' and time >= now() - 1d
name: system
time                 sample
----                 ------
2017-03-26T10:53:10Z 0.45
2017-03-26T13:52:00Z 0.32
2017-03-26T16:27:30Z 0.07
```

Transformations(转换函数)

CEILING

1.2版本中暂未实现。

CUMULATIVE_SUM

累加和。

求平均值：

```
> select mean(tcp_currestab) from net where host='nl-cloud-opens
tack-1' and time >= now() - 1d group by time(12h)
name: net
time                 mean
----                 ----
2017-03-26T00:00:00Z 143.7172131147541
2017-03-26T12:00:00Z 143.41736111111112
2017-03-27T00:00:00Z 143.2752100840336
```

对平均值做累加(自己的值加上个组的值)

```
> select cumulative_sum(mean(tcp_currestab)) from net where host
='nl-cloud-openstack-1' and time >= now() - 1d group by time(12h
)
name: net
time                  cumulative_sum
----                  --------------
2017-03-26T00:00:00Z 143.71692940370116
2017-03-26T12:00:00Z 287.1342905148123
2017-03-27T00:00:00Z 430.40832057423205
```

DERIVATIVE

衍生，返回数据的变化率。(可以用来计算pps、网速、磁盘读写速率).

> SELECT DERIVATIVE(, []) FROM [WHERE]

derivative函数计算一个字段上两个数据之间的差值，然后除以unit(如果不指定，默认是1s)。

查询网卡的包量:

```
> SELECT packets_recv FROM "net" where host='nl-cloud-openstack-
1' and time >= now() - 1m
name: net
time                  packets_recv
----                  ------------
2017-03-27T08:26:50Z 82409624
2017-03-27T08:27:00Z 82409767
2017-03-27T08:27:10Z 82409923
2017-03-27T08:27:20Z 82410066
2017-03-27T08:27:30Z 82410212
2017-03-27T08:27:40Z 82410352
```

我们用data_interval表示两个数据之间的时间差，比如上面的时间差是10s

进行derivative操作:

```
> SELECT derivative("packets_recv", 1s) FROM "net" where host='n
l-cloud-openstack-1' and time >= now() - 1m
name: net
time                  derivative
----                  ----------
2017-03-27T08:27:00Z 14.3
2017-03-27T08:27:10Z 15.6
2017-03-27T08:27:20Z 14.3
2017-03-27T08:27:30Z 14.6
2017-03-27T08:27:40Z 14
```

我们用derivative_unit表示derivative中的单位(默认是1s)，当前时间derivative值的计算公式是:

> 当前时间derivative值=(当前时间的值 - 上一个时间的值)/(data_interval/derivative_unit)。

我们以2017-03-27T08:27:00Z这个时间的数据为例，当前的值是82409767，上一个时间2017-03-27T08:26:50Z的值是82409624，data_interval为两个点之间的时间差(2017-03-27T08:27:00Z - 2017-03-27T08:26:50Z = 10s)，derivative_unit为1s，根据计算公式:

> (82409767 - 82409624)/(10s / 1s) = 14.3

如果使用group by time(interval)

```
> SELECT derivative("packets_recv", 10s) FROM "net" where host='
nl-cloud-openstack-1' and time >= now() - 1m
name: net
time                  derivative
----                  ----------
2017-03-27T08:27:10Z 156
2017-03-27T08:27:20Z 143
2017-03-27T08:27:30Z 146
2017-03-27T08:27:40Z 140
2017-03-27T08:27:50Z 149
```

DIFFERENCE

求差。

计算每天的最小连接数值：

```
> select min(tcp_currestab) from net where host='nl-cloud-openst
ack-1' and time >= now() - 10d group by time(1d)
name: net
time                  min
----                  ---
2017-03-17T00:00:00Z 158
2017-03-18T00:00:00Z 158
2017-03-19T00:00:00Z 158
2017-03-20T00:00:00Z 156
2017-03-21T00:00:00Z 137
2017-03-22T00:00:00Z 137
2017-03-23T00:00:00Z 137
2017-03-24T00:00:00Z 141
2017-03-25T00:00:00Z 141
2017-03-26T00:00:00Z 141
2017-03-27T00:00:00Z 141
```

计算每天最小连接数的变化:

```
> select difference(min(tcp_currestab)) from net where host='nl-
cloud-openstack-1' and time >= now() - 10d group by time(1d)
name: net
time                  difference
----                  ----------
2017-03-18T00:00:00Z 0
2017-03-19T00:00:00Z 0
2017-03-20T00:00:00Z -2
2017-03-21T00:00:00Z -19
2017-03-22T00:00:00Z 0
2017-03-23T00:00:00Z 0
2017-03-24T00:00:00Z 4
2017-03-25T00:00:00Z 0
2017-03-26T00:00:00Z 0
2017-03-27T00:00:00Z 0
```

ELAPSED

单位:

```
> SELECT ELAPSED(<field_key>, <unit>) FROM <measurement_name> [W
HERE <stuff>]
```

显示两个数据点之间的经历的时间差的个数，默认单位为ns(纳秒)。比如:

```
> select elapsed(tcp_currestab) from net where host='nl-cloud-op
enstack-1' and time >= now() - 1m
name: net
time                 elapsed
----                 -------
2017-03-27T08:14:30Z 10000000000
2017-03-27T08:14:40Z 10000000000
2017-03-27T08:14:50Z 10000000000
2017-03-27T08:15:00Z 10000000000
2017-03-27T08:15:10Z 10000000000
```

以5s为单位显示：

```
> select elapsed(tcp_currestab, 5s) from net where host='nl-clou
d-openstack-1' and time >= now() - 1m
name: net
time                 elapsed
----                 -------
2017-03-27T08:20:20Z 2
2017-03-27T08:20:30Z 2
2017-03-27T08:20:40Z 2
2017-03-27T08:20:50Z 2
2017-03-27T08:21:00Z 2
```

由于两个数据之间的时间间隔是10s，单位是5s，所以计算出来的是2。

注意，如果unit大于数据之间的时间间隔，会显示为0，比如以20s为单位显示：

```
> select elapsed(tcp_currestab, 20s) from net where host='nl-clo
ud-openstack-1' and time >= now() - 1m
name: net
time                  elapsed
----                  -------
2017-03-27T08:17:50Z 0
2017-03-27T08:18:00Z 0
2017-03-27T08:18:10Z 0
2017-03-27T08:18:20Z 0
```

FLOOR

1.2版本中暂未实现。

HISTOGRAM

1.2版本中暂未实现。

MOVING_AVERAGE

NON_NEGATIVE_DERIVATIVE

Predictors(预测函数)

HOLT_WINTERS

Telegraf

Telegraf是一个用Golang编写的代理程序，可以收集系统和服务的统计数据，并写入到InfluxDB。Telegraf具有内存占用小的特点，通过开发插件我们可以添加我们需要的服务。

本文使用的telegraf版本为V1.2,官网文档地址。

安装

CentOS7

在CentOS7上部署Telegraf可以使用我写的ansible role,不过为了熟悉Telegraf,建议还是手工安装一遍。

创建仓库文件

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdata.repo
[influxdata]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basea
rch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key
EOF
```

install, start and enable service:

```
yum install telegraf
systemctl start telegraf
systemctl enable telegraf
```

配置文件路径: **/etc/telegraf/telegraf.conf**

macOS

```
brew install telegraf
brew services start telegraf
```

配置文件路径: **/usr/local/etc/telegraf.conf**

数据采集频率

默认的采集频率是10s,有点长,可以修改为5s

```
[agent]
  ## Default data collection interval for all inputs
  interval = "10s"
```

和InfluxDB集成

以CentOS7为例。

和InfluxDB集成的配置主要在/etc/telegraf/telegraf.conf文件的**[[outputs.influxdb]]**块中

urls配置

配置InfluxDB的API的endpoint，比如:

```
  urls = ["https://192.168.168.201:8086"]
```

auth配置

如果InfluxDB启用了auth，在telegraf中需要做如下配置:

```
  username = "frank6866"
  password = "change_me"
```

注意，如果InfluxDB中没有启用认证，可以不用事先在InfluxDB中创建telegraf使用的数据库，telegraf启动后会去创建；如果InfluxDB中启用了认证，并且telegraf配置的用户在InfluxDB中没有创建数据库的权限，telegraf会报无法创建数据库的错误，如下:

```
# journalctl -fl -u telegraf
......
InfluxDB Output Error: {"error":"database not found: \"telegraf\
""}
......
```

解决: 在InfluxDB中使用具有admin权限的账户创建数据库。

```
> CREATE DATABASE telegraf
```

https配置

如果InfluxDB启用了https，在telegraf中需要做如下配置:

```
  ssl_ca = "/etc/pki/influxdb/cacert.crt"
  ssl_cert = "/etc/pki/influxdb/influxdb.crt"
  ssl_key = "/etc/pki/influxdb/private/influxdb.key"
```

验证

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

查看telegraf抓取的所有数据

```
# telegraf -test
* Plugin: inputs.system, Collection 1
> system,host=nl-cloud-dyc-k8s-1 load1=0.19,load5=0.34,load15=0.
25,n_users=2i,n_cpus=2i 1494408513000000000
> system,host=nl-cloud-dyc-k8s-1 uptime=1902973i,uptime_format="
22 days,  0:36" 1494408513000000000
* Plugin: inputs.net, Collection 1
......
```

查看telegraf抓取的内存和网络数据(-input-filter的值为配置文件中inputs.xxx中的xxx,可以设置多个值,使用冒号分隔):

```
# telegraf -test -input-filter mem:diskio
* Plugin: inputs.mem, Collection 1
> mem,host=nl-cloud-dyc-k8s-1 available=1193779200i,used=2950987
776i,active=3256971264i,total=4144766976i,free=143593472i,cached
=1392947200i,buffered=0i,inactive=581951488i,used_percent=71.197
91759313613,available_percent=28.802082406863878 149440864300000
0000
* Plugin: inputs.diskio, Collection 1
> diskio,name=vda,host=nl-cloud-dyc-k8s-1 writes=11154527i,read_
bytes=2805115392i,write_bytes=40910338048i,read_time=512738i,wri
te_time=49621289i,io_time=35475332i,iops_in_progress=0i,reads=43
120i 1494408643000000000
> diskio,name=vda1,host=nl-cloud-dyc-k8s-1 read_time=512595i,wri
te_time=49525882i,io_time=35426178i,iops_in_progress=0i,reads=42
936i,writes=10720821i,read_bytes=2804361728i,write_bytes=4091033
8048i 1494408643000000000
> diskio,name=dm-0,host=nl-cloud-dyc-k8s-1 io_time=101894i,iops_
in_progress=0i,reads=176074i,writes=123313i,read_bytes=315287347
2i,write_bytes=4582692352i,read_time=135021i,write_time=1289091i
 1494408643000000000
```

Input

telegraf是input驱动的，它从配置文件中指定的input中收集metrics。

使用某些input生成配置文件:

```
# telegraf -sample-config -input-filter cpu:mem:disk -output-fil
ter influxdb > telegraf.conf
```

Grafana

Grafana是一个开源的监控数据展示和分析工具,界面美观有科技感,功能强大,可以从多种数据源取值并进行展示,比如InfluxDB.

安装

Grafana的默认端口是3000，默认账号和密码都是admin。

CentOS7

在CentOS7上部署Grafana可以使用我写的ansible role,不过为了熟悉Grafana,建议还是手工安装一遍。

yum仓库文件

```
[grafana]
name=grafana
baseurl=https://packagecloud.io/grafana/stable/el/7/$basearch
repo_gpgcheck=0
enabled=1
gpgcheck=0
gpgkey=https://packagecloud.io/gpg.key https://grafanarel.s3.ama
zonaws.com/RPM-GPG-KEY-grafana
sslverify=0
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
```

安装grafana:

```
# yum install grafana
```

启动Grafana并设置为开机启动:

```
# systemctl start grafana-server
# systemctl enable grafana-server
```

- CentOS7中Grafana配置文件位置是: **/etc/grafana/grafana.ini**
- CentOS7中Grafana日志文件位置是: **/var/log/grafana/grafana.log**

macOS

```
brew install grafana
brew services start grafana
```

基本概念

参考: http://docs.grafana.org/guides/basic_concepts/

Data Source

数据源，Grafana支持支持很多种时间序列的数据源，每一种数据源都有一个特定的定制的查询编辑器。

官方支持的数据源包括： Graphite、InfluxDB、OpenTSDB、Prometheus、Elasticsearch、CloudWatch。

Organization

Grafana支持多种组织，因此它可以支持多种部署模型，包括只使用一个Grafana为多个潜在的不被信任的组织提供服务。

在很多场景下，Grafana只有一个组织，每个组织都有一个或者多个数据源。

所有的Dashboard只被一个特定的组织拥有。

User

用户是Grafana中的一个有名称的账号，一个用户可以属于一个或者多个组织，在不同的组织中可以有不同的权限。

Grafana支持多种内部和外部认证的方式，比如从Grafana自己的数据库中，或者从一个外部的SQL数据库，或者从一个外部的LDAP服务器。

Row

行在Dashboard内部是一个逻辑分割条，用来把Panels组合在一起。

Panel

面板是Grafana中可视化的基本单元，每个面板都配有一个查询编辑器。面板的类型如下：

- Graph：可以根据metrics制作图表
- Signlestate
- Table
- Dashlist：没有连接到任何数据源
- Text：没有连接到任何数据源

Query Editor

查询编辑器向外暴露了数据源，使用查询编辑器我们可以查询数据源包含的metrics。

我们可以在查询编辑器中使用模板变量。

Dashboard

控制台把所有的东西聚合在一起，我们可以把控制台当成面板或者行的集合。

Grafana Tutorial

本文的主要内容:

- Grafana添加InfluxDB数据源(InfluxDB启用了auth并使用HTTPS)的相关配置
- 在Grafana中添加一个Dashboard用来展示Linux的基础监控(比如CPU、内存、网络、磁盘等)
- 在Dashboard中添加Graph，展示某个具体的监控想，比如CPU监控图表
- 通过模板变量在一个Dashboard切换不同主机的监控图，或多个主机的监控图一起显示

添加InfluxDB数据源

添加InfluxDB数据源相关的配置:

- Name: 数据源的名称，自定义的值，比如influxdb-192.168.168.201
- Type: 选择InfluxDB
- Default: 将该数据源设置为默认的数据源，这样有一个好处是，在添加panel的时候，会自动使用这个数据源。

Http settings:

- Url: InfluxDB的HTTP endpoint，比如: https://192.168.168.201:8086
- Access: direct表示url is used directly from browser；proxy表示Grafana backend will proxy the request。这里选择proxy。

Http Auth:

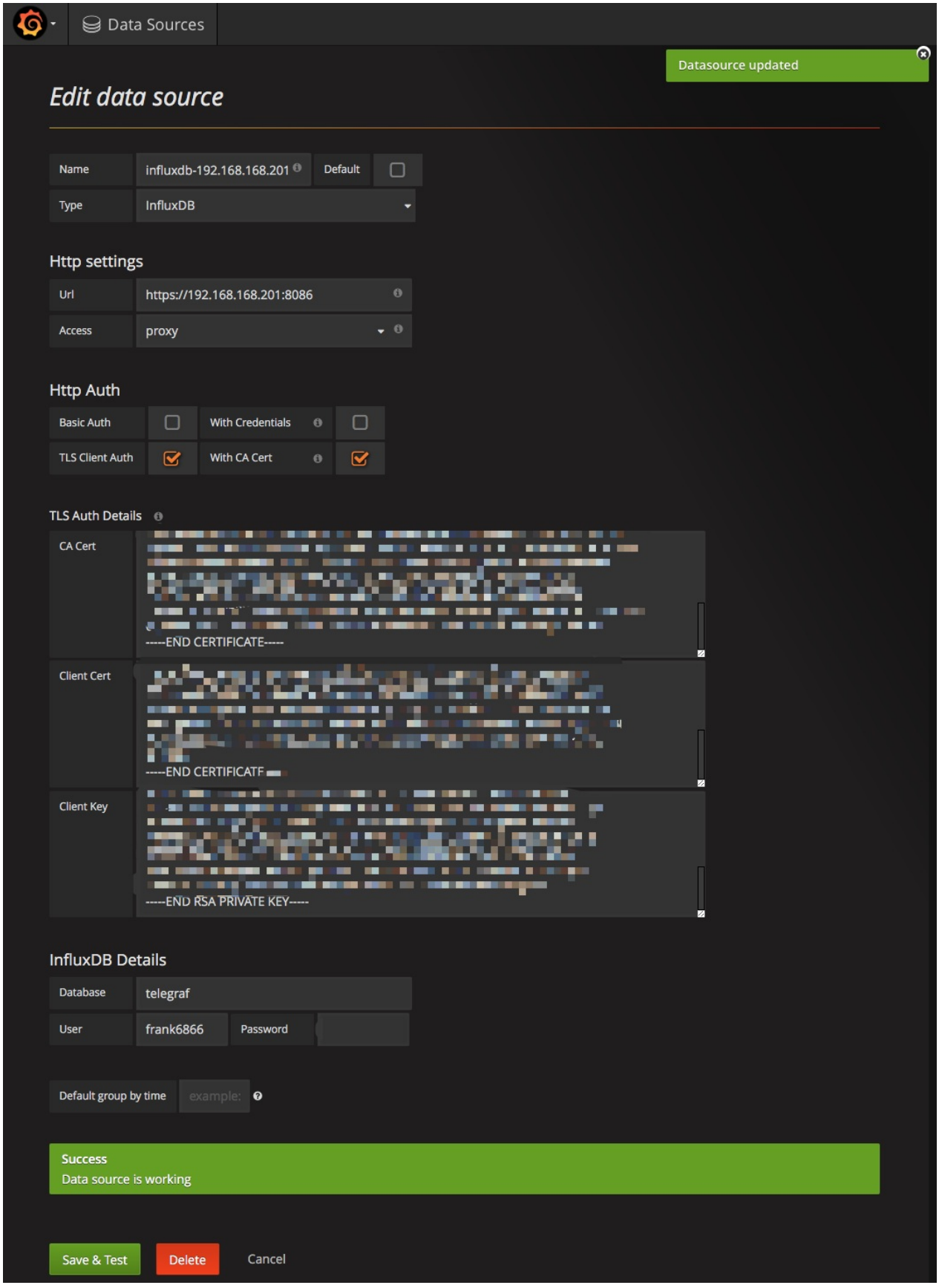
- Basic Auth: 是否使用HTTP basic认证，我们没有用到，就不勾选了。
- TLS Client Auth: 如果InfluxDB启用了https，需要证书内容(PEM格式)粘贴到TLS Auth Details的Client Cert中，将密钥内容(PEM格式)粘贴到TLS Auth Details的Client Key中
- With CA Cert: 如果证书是是自建CA签发或者自签的，需要将CA的证书内容(PEM格式)粘贴到TLS Auth Details的CA Cert中

InfluxDB Details:

- Database: InfluxDB中的数据库名称，比如telegraf

- User: InfluxDB中的用户名
- Password: InfluxDB中用户对应的密码

相关配置如下图:



添加Dashboard

找到新建Dashboard的菜单，Command+S然后输入名称就新建了一个Dashboard。在这里新建一个名为"Linux basic"的Dashboard。

添加row

row是panel的容器

添加Graph

添加一个类型为Graph的panel。

Graph配置

点击Graph的标题，再点击"Edit"，可以打开Graph的配置页面:

General标签

- Title: 表示图表的名称，比如, 设置为"CPU util"
- Description: 图表的描述信息

Metrics标签

首先设置"Panel data source"，如果上面添加的数据源被设置为Default，在Panel中就不需要设置了；如果上面的数据源没有被设置为Default，就需要手动设置数据源。

"Group by time interval"设置的是根据时间分组的秒数，这个值要大于telegraf数据采集的时间间隔(10s)，比如一般设置为">10s".

query

在Metrics中，每一个query都表示一条线。

query的组成:

- FROM: "select measurement"表示选择InfluxDB中的哪个measurement，这里我们选择cpu
- WHERE: "WHERE"是根据tag key和tag value过滤的，这里我们选择tag key为host，tag value随便选一个，这里我选择的是vagrant
- SELECT: field(value)表示显示哪一个cpu表中哪个指标，这里我们选择"usage_user"，后面的mean表示group by表示的这段时间内的平均值。
- GROUP BY: 表示分组，这里按照时间分组，设置为time($interval)和fill(null)
- ALIAS BY: 这条线的legend名称，比如设置为cpu.usr

帮助信息

在Metrics标签的最下面有三个帮助信息可以参考:

- alias patterns
- stacking & fill
- group by time

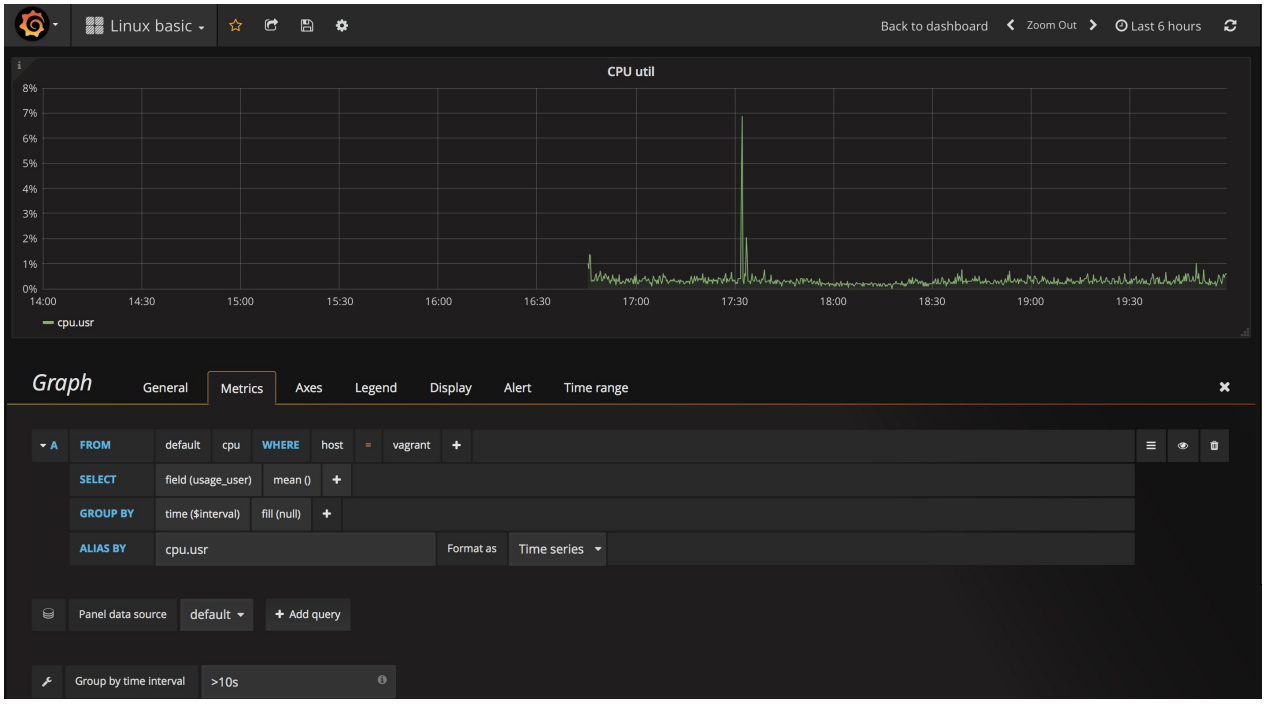
Axes标签

在Axes中可以设置坐标轴的单位，cpu利用率的单位是%，设置方法如下:

LeffY - Uint - none - percent(0-100)

> Grafana中,Unit表示的是查询出来的数据的单位,设置好后,Grafana会自动显示成合适的单位。比如原始数据的单位是bytes,数字是1024,在Grafana中会根据值自动在坐标轴上显示成1K,方便阅读。
>
> 还有一点注意的是,对于百分数,如果查询出来的是0-1之间的值,将percent设置为(0-1);如果查询出来的是0-100之间的值,将percent设置为(0-100)。对于百分比,Grafana在坐标轴中总是显示0-100%,所以将原始数据的单位设置好才能够正确地显示。

设置完后，效果图如下:



使用模板

定义模板变量

这里以InfluxDB(使用Telegraf采集)数据源为例，配置主机名变量 点击Dashboard上方的小齿轮('Manage dashboard' -> 'Templating' -> 'New')

Variable部分

- Name: 变量的名称(在查询条件中显示,这里设置为server_name)
- Label: 变量的标签名(在页面上过滤时显示,这里设置为主机名)
- Type: 设置为Query
- "Query Options" -> "Data Source"：选择我们配置的数据源
- "Query Options" -> "Refresh"：选择"On Time Range Change"
- "Query Options" -> "Query"： 查询主机名，输入 SHOW TAG VALUES ON "telegraf" WITH KEY = "host"
- "Selection Options" -> "Multi-value": 勾选中

使用模板变量

以CPU为例

```
FROM default cpu WHERE host =~ /^$server_name$/
SELECT field(usage_user) mean()
GROUP BY time($interval) tag(host)
cpu.usage($tag_host)
```

Grafana 配置

配置

http://docs.grafana.org/installation/configuration/

SMTP配置

http://docs.grafana.org/installation/configuration/#smtp

smtp部分配置如下:

```
[smtp]
enabled = true
host = smtp.vip.sina.com:25
user = dyc_abc
password = ...
;cert_file =
;key_file =
;skip_verify = false
from_address = dyc_abc@vip.sina.com
```

修改完配置文件后需要重启grafana

> systemctl restart grafana

异常

测试发送邮件时,发现发送失败,查看/var/log/grafana/grafana.log文件存在以下问题

> Notification service failed to initialize" logger=server erro="Invalid email address for smpt from_adress config

解决: 我的from_address配置的是**admin@grafana.10.12.10.8**,不是一个正常的email地址,修改为admin@grafana_10_12_10_8.com

> error="gomail: could not send email 1: 553 Envolope sender mismatch with login user.."

解决: user字段必须和from_address字段中@前面的一样

通知

"Alerting" - "Notifications" - "New Notification"

Send on all alerts表示对所有的告警都使用这个通知。

设置完后,可以使用"Send Test"测试一下。

email

smtp的配置参见上面。

Type设置为email,如果有多个地址,使用分号";"分隔。

slack

申请incoming webhook

1. 点击左上角账户,点击 team - "Apps & integrations",搜索webhook(这个搜索功能很好用,还可以搜索token)
2. 找到"incoming webhook"然后点击进去
3. 点击左边的Request, 批注后配置一下就会生成一个url

配置

Type设置为slack。

- Url: 设置为上面申请的incoming webhook
- Recipient(接受者): 如果是某个人,使用@user;如果是某个chanel,使用#chanel
- Mention: 如果Recipient设置的是chanel,这个选项表示在这个chanel里面艾特某个具体的人,使用@user。

Grafana 异常

没有数据也没有报错

检查Group By time interval是否设置了值,如果group by time interval的值小于数据的收集周期,数据会显示不出来。

解决: 设置Group By time interval,大于数据的抓取周期,比如">10s"

Invalid dimensions for plot, width = 1867.625, height = 0

在编辑图表时,不小心调整了图表了大小,返回Dashboard后提示这个异常。

看了一下正常图表中"General"标签中的Dimensions的Span是5,我误操作图表的这个是是12;直接修改为5还是不行,后来将图表的大小调整了一下就可以了。

Linux监控

本文主要介绍了Telegraf中Linux相关的监控指标。

查看监控项分类(查询telegraf表中的measurement):

```
> use telegraf
> show measurements
name: measurements
name
----
cpu
disk
diskio
kernel
mem
net
processes
swap
system
```

cpu

查看cpu表中的metrics:

```
> show field keys on telegraf from cpu
name: cpu
fieldKey         fieldType
--------         ---------
usage_guest      float
usage_guest_nice float
usage_idle       float
usage_iowait     float
usage_irq        float
usage_nice       float
usage_softirq    float
usage_steal      float
usage_system     float
usage_user       float
```

如下:

- usage_user: 用户态占用的CPU时间百分比
- usage_system: 内核态占用的CPU时间百分比
- usage_iowait: io等待占用的CPU时间百分比
- usage_idle: CPU空闲时间百分比
- usage_irq: 硬中断占用的CPU时间百分比
- usage_softirq: 软中断占用的CPU时间百分比
- usage_nice: 被nice的进程占用的CPU时间百分比
- usage_steal: 虚拟机占用的CPU时间百分比(如果OS作为Hypervisor)

processes

查看processes表中的metrics:

```
> show field keys on telegraf from processes
name: processes
fieldKey      fieldType
--------      ---------
blocked       integer
paging        integer
running       integer
sleeping      integer
stopped       integer
total         integer
total_threads integer
unknown       integer
zombies       integer
```

system

查看system表中的metrics:

```
> show field keys on telegraf from system
name: system
fieldKey      fieldType
--------      ---------
load1         float
load15        float
load5         float
n_cpus        integer
n_users       integer
uptime        integer
uptime_format string
```

kernel

查看kernel表中的metrics:

```
> show field keys on telegraf from kernel
name: kernel
fieldKey         fieldType
--------         ---------
boot_time        integer
context_switches integer
interrupts       integer
processes_forked integer
```

mem

查看mem表中的metrics:

```
> show field keys on telegraf from mem
name: mem
fieldKey          fieldType
--------          ---------
active            integer
available         integer
available_percent float
buffered          integer
cached            integer
free              integer
inactive          integer
total             integer
used              integer
used_percent      float
```

swap

查看swap表中的metrics:

```
> show field keys on telegraf from swap
name: swap
fieldKey     fieldType
--------     ---------
free         integer
in           integer
out          integer
total        integer
used         integer
used_percent float
```

disk

查看disk表中的metrics:

```
> show field keys on telegraf from disk
name: disk
fieldKey     fieldType
--------     ---------
free         integer
inodes_free  integer
inodes_total integer
inodes_used  integer
total        integer
used         integer
used_percent float
```

- free:
- inodes_free:
- inodes_total:
- inodes_used:
- total:
- used:
- used_percent:

diskio

查看diskio表中的metrics:

```
> show field keys on telegraf from diskio
name: diskio
fieldKey         fieldType
--------         ---------
io_time          integer
iops_in_progress integer
read_bytes       integer
read_time        integer
reads            integer
write_bytes      integer
write_time       integer
writes           integer
```

net

查看net表中的metrics:

```
> show field keys on telegraf from net
name: net
fieldKey              fieldType
--------              ---------
bytes_recv            integer
bytes_sent            integer
drop_in               integer
drop_out              integer
err_in                integer
err_out               integer
icmp_inaddrmaskreps   integer
icmp_inaddrmasks      integer
icmp_incsumerrors     integer
icmp_indestunreachs   integer
icmp_inechoreps       integer
icmp_inechos          integer
icmp_inerrors         integer
icmp_inmsgs           integer
icmp_inparmprobs      integer
```

```
icmp_inredirects      integer
icmp_insrcquenchs     integer
icmp_intimeexcds      integer
icmp_intimestampreps  integer
icmp_intimestamps     integer
icmp_outaddrmaskreps  integer
icmp_outaddrmasks     integer
icmp_outdestunreachs  integer
icmp_outechoreps      integer
icmp_outechos         integer
icmp_outerrors        integer
icmp_outmsgs          integer
icmp_outparmprobs     integer
icmp_outredirects     integer
icmp_outsrcquenchs    integer
icmp_outtimeexcds     integer
icmp_outtimestampreps integer
icmp_outtimestamps    integer
icmpmsg_intype0       integer
icmpmsg_outtype3      integer
icmpmsg_outtype8      integer
ip_defaultttl         integer
ip_forwarding         integer
ip_forwdatagrams      integer
ip_fragcreates        integer
ip_fragfails          integer
ip_fragoks            integer
ip_inaddrerrors       integer
ip_indelivers         integer
ip_indiscards         integer
ip_inhdrerrors        integer
ip_inreceives         integer
ip_inunknownprotos    integer
ip_outdiscards        integer
ip_outnoroutes        integer
ip_outrequests        integer
ip_reasmfails         integer
ip_reasmoks           integer
ip_reasmreqds         integer
ip_reasmtimeout       integer
```

```
packets_recv          integer
packets_sent          integer
tcp_activeopens       integer
tcp_attemptfails      integer
tcp_currestab         integer
tcp_estabresets       integer
tcp_incsumerrors      integer
tcp_inerrs            integer
tcp_insegs            integer
tcp_maxconn           integer
tcp_outrsts           integer
tcp_outsegs           integer
tcp_passiveopens      integer
tcp_retranssegs       integer
tcp_rtoalgorithm      integer
tcp_rtomax            integer
tcp_rtomin            integer
udp_incsumerrors      integer
udp_indatagrams       integer
udp_inerrors          integer
udp_noports           integer
udp_outdatagrams      integer
udp_rcvbuferrors      integer
udp_sndbuferrors      integer
udplite_incsumerrors  integer
udplite_indatagrams   integer
udplite_inerrors      integer
udplite_noports       integer
udplite_outdatagrams  integer
udplite_rcvbuferrors  integer
udplite_sndbuferrors  integer
```

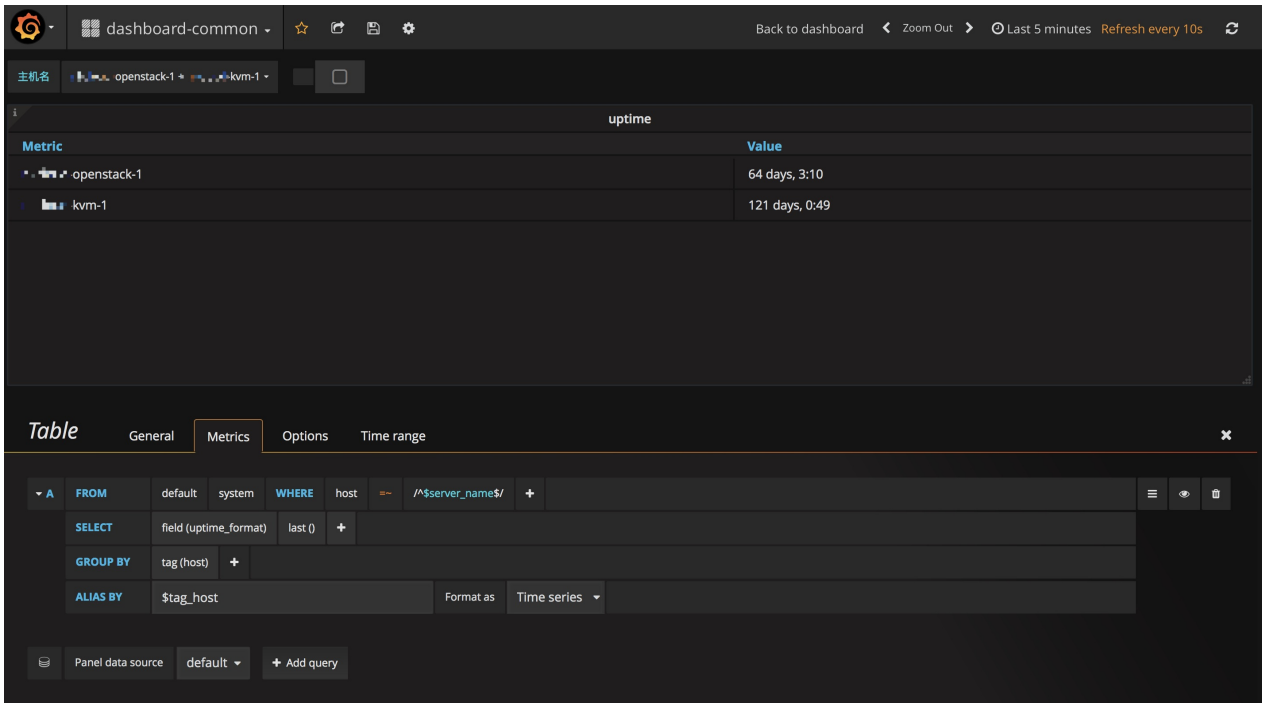
基本信息

系统运行时间

以表格的形式显示系统自启动以来的运行时间,当选择多个主机时,表格显示每个主机的系统运行时间。

数据源: system表中的uptime_format是以字符串表示的系统启动时间

如下图



配置，添加panel时，选择Table类型。

General标签

- Title: 设为"uptime"
- Description: 设为"system uptime"

Metrics标签

query editor设置:

- FROM: 选择从system表拉取数据
- WHERE: 为了支持多个主机，设置为host等于主机名模板变量的值(模板变量的设置参考Grafana Tutorial)
- SELECT: 选择"uptime_format"字段，并使用last这个选择函数，因为我们只关注最后一个值
- GROUP BY: 按照主机分组，设为tag(host)
- ALIAS BY: 设置为$tag_host，这样表格中的Metri列就会显示主机名

Options标签

- Data: 将"Table Transform"设置为"Time series to rows"
- Column Styles: 将Time列设置为Hidden，就不会把时间显示出来；如果需要显示时间这一列，可以将Type设置为Date

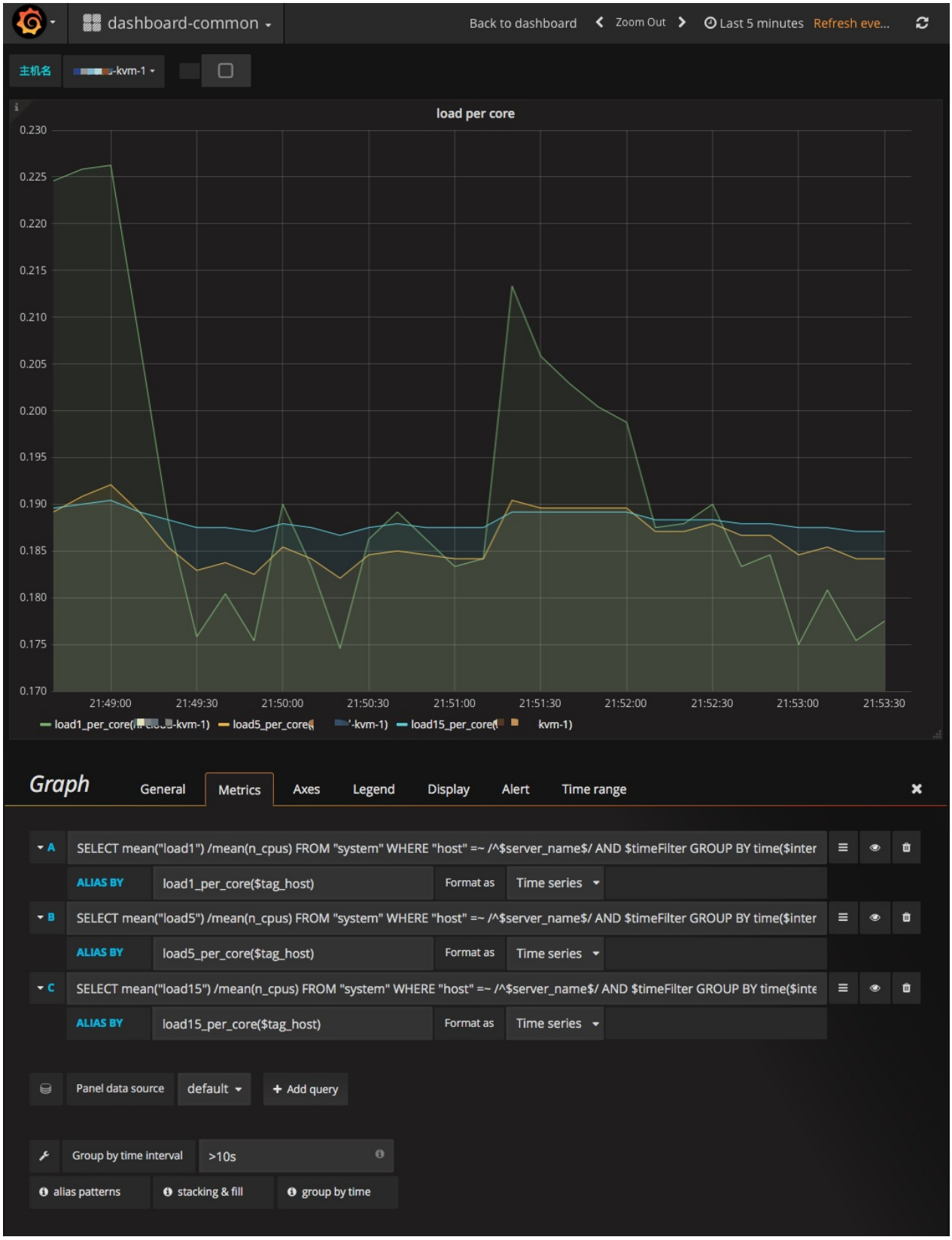
CPU监控

每个核的平均负载

对于多核系统，load显示的是所有核的总负载，1个核的系统load为2时负载就有点高了，但是24核的系统load为12时系统的负载都不高，所以我们使用每个核的平均负载反应系统的负载情况。当每个核的平均负载大于1时，就需要注意了。

数据源: system表中的load1表示的是系统的所有核1分钟的平均负载，n_cpus表示系统逻辑cpu的个数(查看系统的cpu信息参考这里)。使用load1/n_cpus就可以得到系统1分钟内每个核的平均负载，5分钟和15分钟计算方法是一样的。

如下图:



添加panel时，选择Graph类型。

General标签

- Title: 设为"load per core"

- Description: 设为"system load average per core"

Metrics标签

query editor设置(load1,load5和load15的配置类似，以load1为例):

由于在SELECT语句中使用了两个字段相除，选择模式不支持，所以点击"Toggle Edit Mode"进入输入模式，查询语句如下:

```
SELECT mean("load1") /mean(n_cpus) FROM "system" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
```

ALIAS BY设置为$tag_host

CPU利用率监控

数据源: cpu表中的相关使用率信息.

配置如下图:


cpu util
3.0%
2.5%
2.0%
1.5%
1.0%
0.5%
0%
19:05
19:10
19:15
19:20
19:25
19:30
19:35
19:40
19:45
19:50
19:55
20:00
20:05
20:10
usage_user(flannel-1)
usage_system(flannel-1)
usage_iowait(flannel-1)
usage_irq(flannel-1)
usage_softirq(flannel-1)
usage_total(flannel-1)
Graph
General
Metrics
Axes
Legend
Display
Alert
Time range
A SELECT mean("usage_user") FROM "cpu" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
B SELECT mean("usage_system") FROM "cpu" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
C SELECT mean("usage_iowait") FROM "cpu" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
D SELECT mean("usage_irq") FROM "cpu" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
E SELECT mean("usage_softirq") FROM "cpu" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
F SELECT 100-mean("usage_idle") FROM "cpu" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
Panel data source
default
+ Add query
Group by time interval
>5s
alias patterns
stacking & fill
group by time

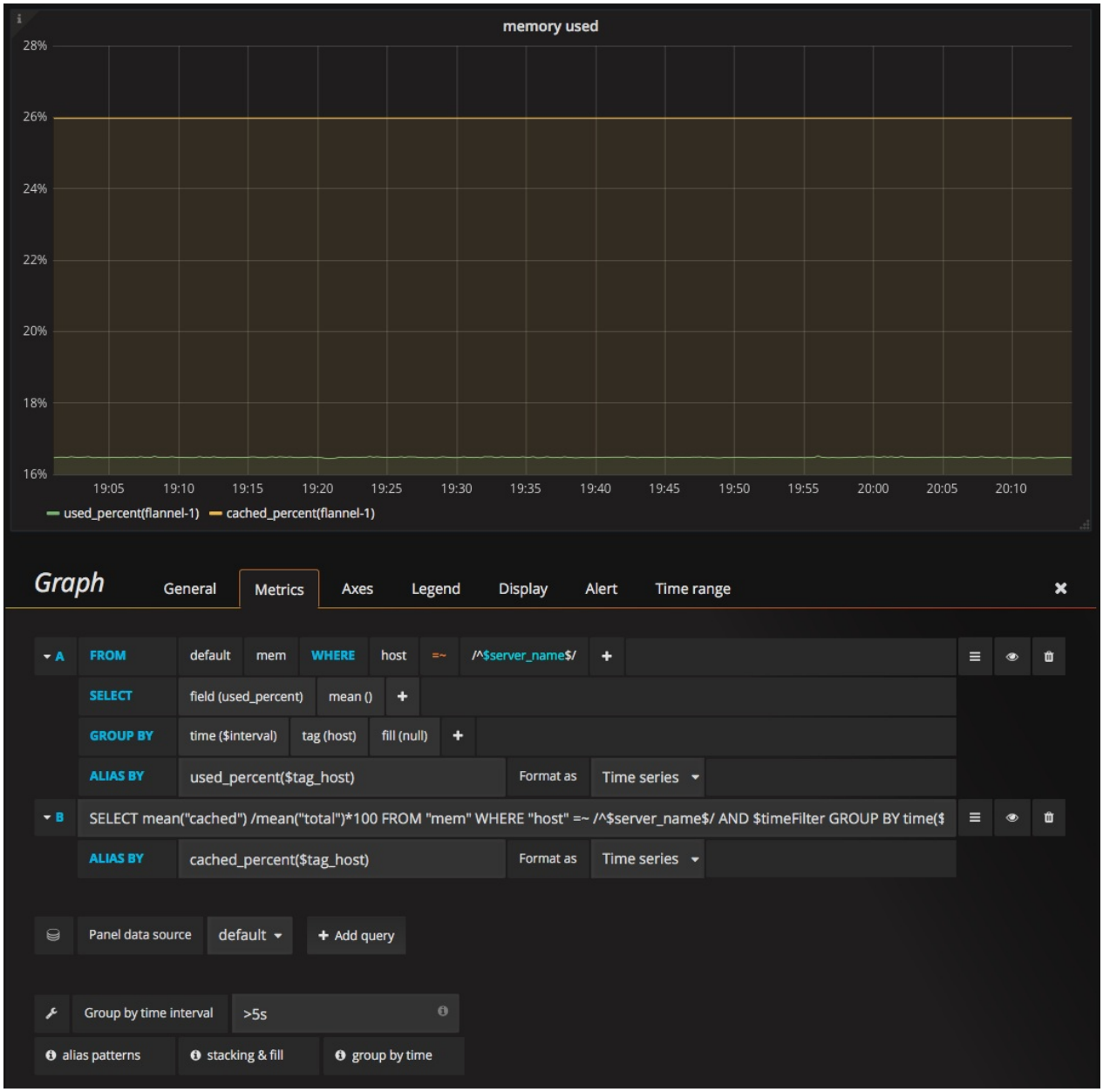
内存监控

监控项:

- 已用内存百分比
- cached占总内存的百分比

数据源: mem表中的used_percent、cached和total字段。

如下图:



General标签

- Title: 设为"memory used"
- Description: 设为"内存使用率"

Metrics标签

已有内存百分比的查询语句如下:

```
SELECT mean("used_percent") FROM "mem" WHERE "host" =~
/^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
```

cached占总内存的百分比如下:

```
SELECT mean("cached") /mean("total")*100 FROM "mem" WHERE "host" =~
/^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)
```

Axes标签

在Axes中可以设置坐标轴的单位:

LeffY - Uint - none - percent(0-100)

磁盘监控

- 使用量监控
- 读写速度监控
- IOPS监控

磁盘用量监控

目的: 监控根分区/和数据分区/data的使用空间百分比。

数据源: disk表中的used_percent字段。

添加panel时，选择Graph类型。

General标签

- Title: 设为"/ and /data used"
- Description: 设为"根分区和数据分区用量"

Metrics标签

根分区/用量的查询语句为:

```
SELECT mean("used_percent") FROM "disk" WHERE "host" =~
/^$server_name$/ AND "path" = '/' AND $timeFilter GROUP BY
time($interval), "host" fill(null)
```

ALIAS BY设置为**root_used($tag_host)**

数据分区/data用量的查询语句为:

```
SELECT mean("used_percent") FROM "disk" WHERE "host" =~
/^$server_name$/ AND "path" = '/data' AND $timeFilter GROUP BY
time($interval), "host" fill(null)
```

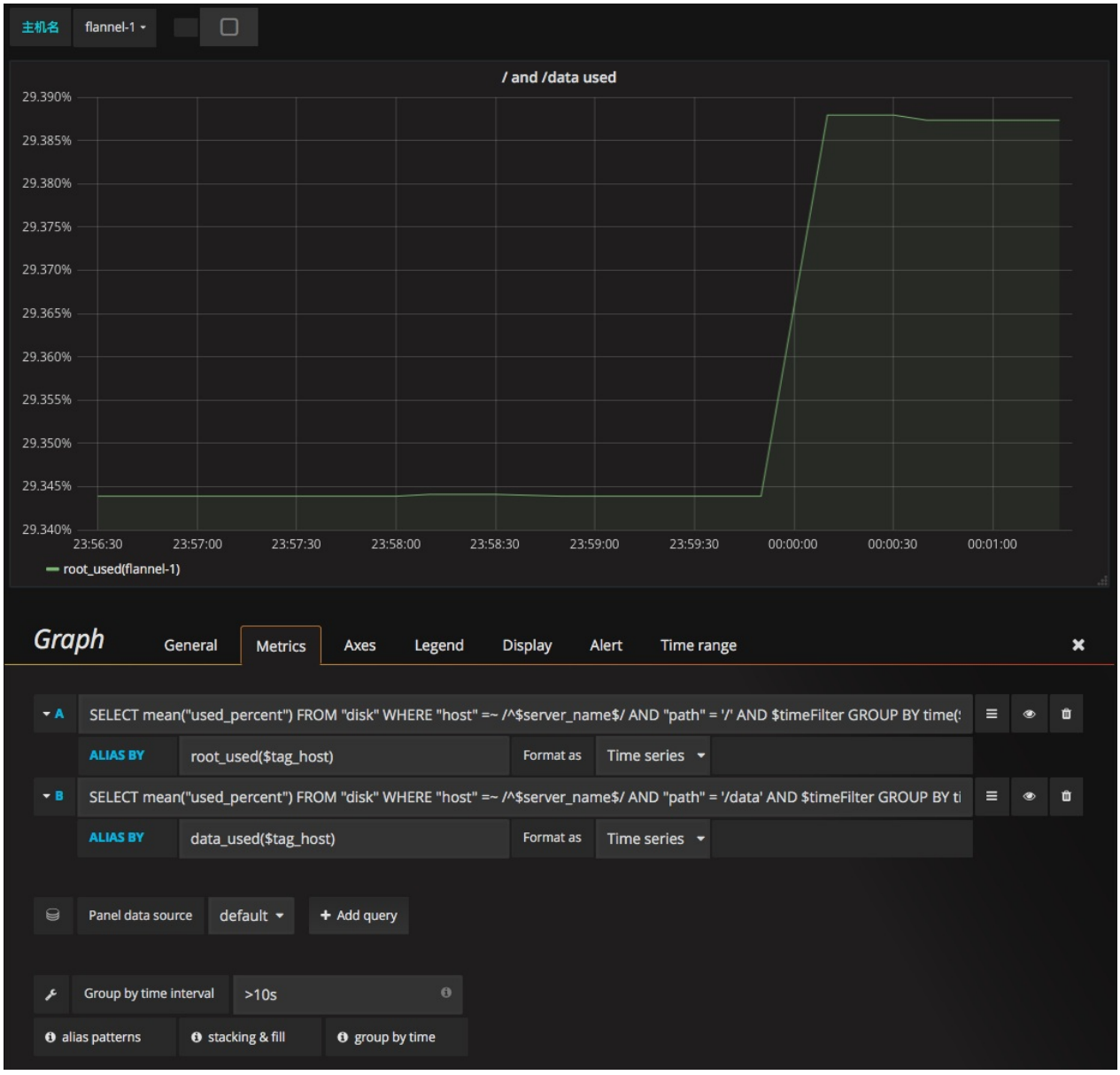
ALIAS BY设置为**data_used($tag_host)**

Axes标签

在Axes中可以设置坐标轴的单位为:

LeffY - Uint - none - percent(0-100)

如下图:



磁盘读写IOPS监控

IOPS的计算方法参考/proc/diskstats文件.

目的: 监控磁盘的IOPS。

数据源: diskio表中的reads和writes字段。

添加panel时，选择Graph类型。

General标签

- Title: 设为"disk iops"
- Description: 设为"磁盘读和写的IOPS"

Metrics标签

read iops的查询语句为:

> SELECT derivative("reads", 1s) FROM "diskio" WHERE "host" =~ /^$server_name$/ AND "name" =~ /^[sv]d[\D]{1,}$/ AND $timeFilter GROUP BY "host", "name" fill(null)

ALIAS BY设置为**read_iops(tag_host-tag_name)**

write iops的查询语句为:

> SELECT derivative("writes", 1s) FROM "diskio" WHERE "host" =~ /^$server_name$/ AND "name" =~ /^[sv]d[\D]{1,}$/ AND $timeFilter GROUP BY "host", "name" fill(null)

ALIAS BY设置为**write_iops(tag_host-tag_name)**

Axes标签

在Axes中可以设置坐标轴的单位为:

LeffY - Uint - none - none

如下图:

磁盘读写速度监控

IOPS的计算方法参考/proc/diskstats文件.

目的: 监控磁盘的读写速度。

数据源: diskio表中的read_bytes和write_bytes字段。

添加panel时，选择Graph类型。

General标签

- Title: 设为"disk rate"

- Description: 设为"磁盘读和写的速度"

Metrics标签

read rate的查询语句为:

```
SELECT derivative("read_bytes", 1s) FROM "diskio" WHERE "host" =~ /^$server_name$/ AND "name" =~ /^[sv]d[\D]{1,}$/ AND $timeFilter GROUP BY "host", "name"
```

ALIAS BY设置为**read_rate(tag_host-tag_name)**

write rate的查询语句为:

```
SELECT derivative("write_bytes", 1s) FROM "diskio" WHERE "host" =~ /^$server_name$/ AND "name" =~ /^[sv]d[\D]{1,}$/ AND $timeFilter GROUP BY "host", "name"
```

ALIAS BY设置为**write_rate(tag_host-tag_name)**

Axes标签

在Axes中可以设置坐标轴的单位为:

LeffY - Uint - date rate - bytes/sec

如下图:


disk rate
250 kBps
200 kBps
150 kBps
100 kBps
50 kBps
0 Bps
00:27:30
00:28:00
00:28:30
00:29:00
00:29:30
00:30:00
00:30:30
00:31:00
00:31:30
00:32:00
read_rate(flannel-1-vda)
write_rate(flannel-1-vda)
Graph
General
Metrics
Axes
Legend
Display
Alert
Time range
A
SELECT derivative("read_bytes", 1s) FROM "diskio" WHERE "host" =~ /^$server_name$/ AND "name" =~ /^[sv]d[\D]{1,}$/ AND $tim
ALIAS BY
read_rate(tag_host-tag_name)
Format as
Time series
B
SELECT derivative("write_bytes", 1s) FROM "diskio" WHERE "host" =~ /^$server_name$/ AND "name" =~ /^[sv]d[\D]{1,}$/ AND $tim
ALIAS BY
write_rate(tag_host-tag_name)
Format as
Time series
Panel data source
default
+ Add query
Group by time interval
>5s
alias patterns
stacking & fill
group by time

网络监控

网速监控

网速数据来自net表中的bytes_recv和bytes_sent字段。

Metrics标签

input流量监控的查询语句:

```
SELECT derivative("bytes_recv", 1s) FROM "net" WHERE "host" =~
/^$server_name$/ AND $timeFilter GROUP BY "host"
```

output流量监控的查询语句:

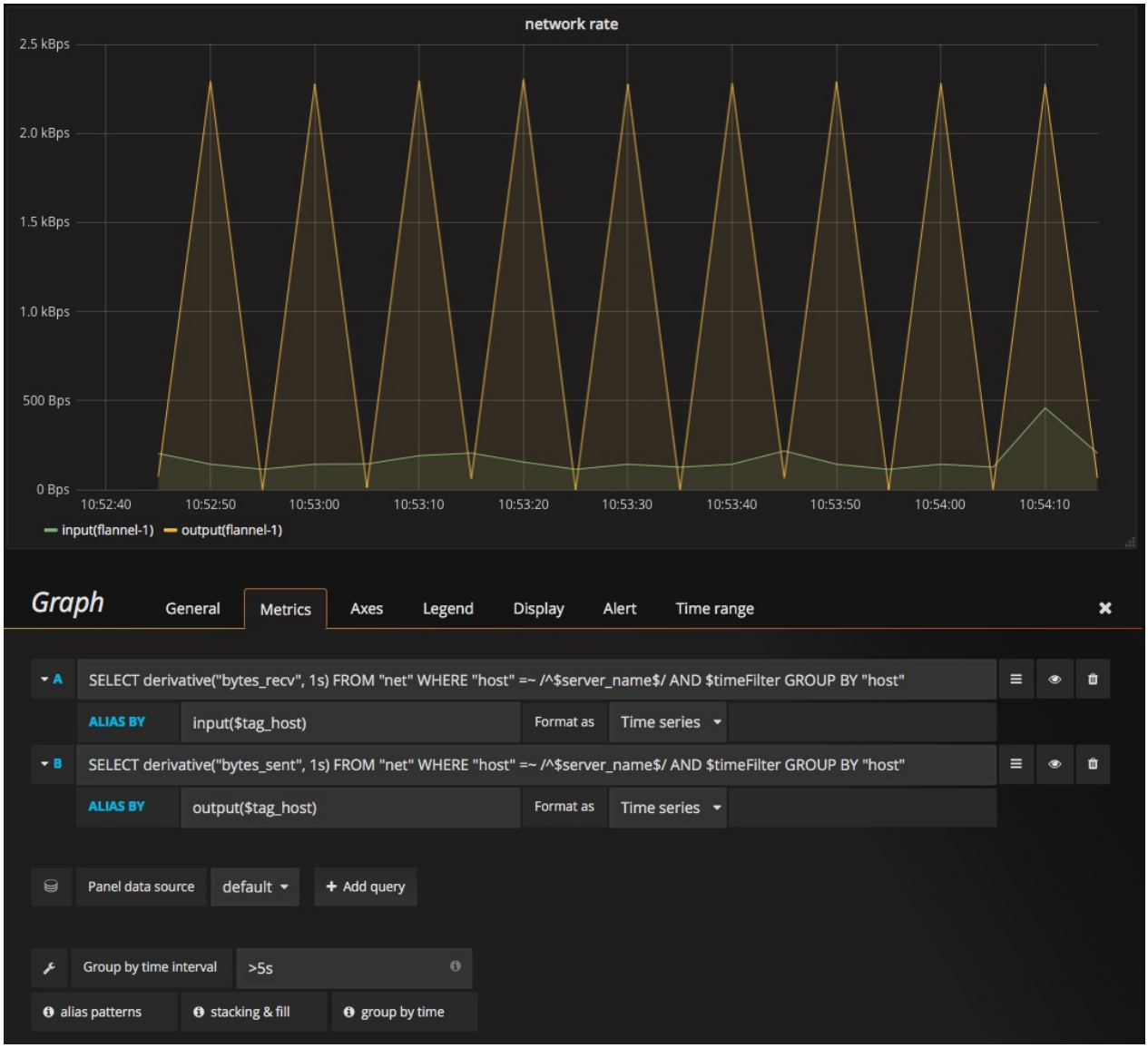
```
SELECT derivative("bytes_sent", 1s) FROM "net" WHERE "host" =~
/^$server_name$/ AND $timeFilter GROUP BY "host"
```

Axes标签

在Axes中可以设置坐标轴的单位为:

LeffY - Uint - date rate - bytes/sec

如下图:



pps监控

pps数据来自net表中的packets_recv和packets_sent字段。

Metrics标签

input pps监控的查询语句:

> SELECT derivative("packets_recv", 1s) FROM "net" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY "host"

output pps监控的查询语句:

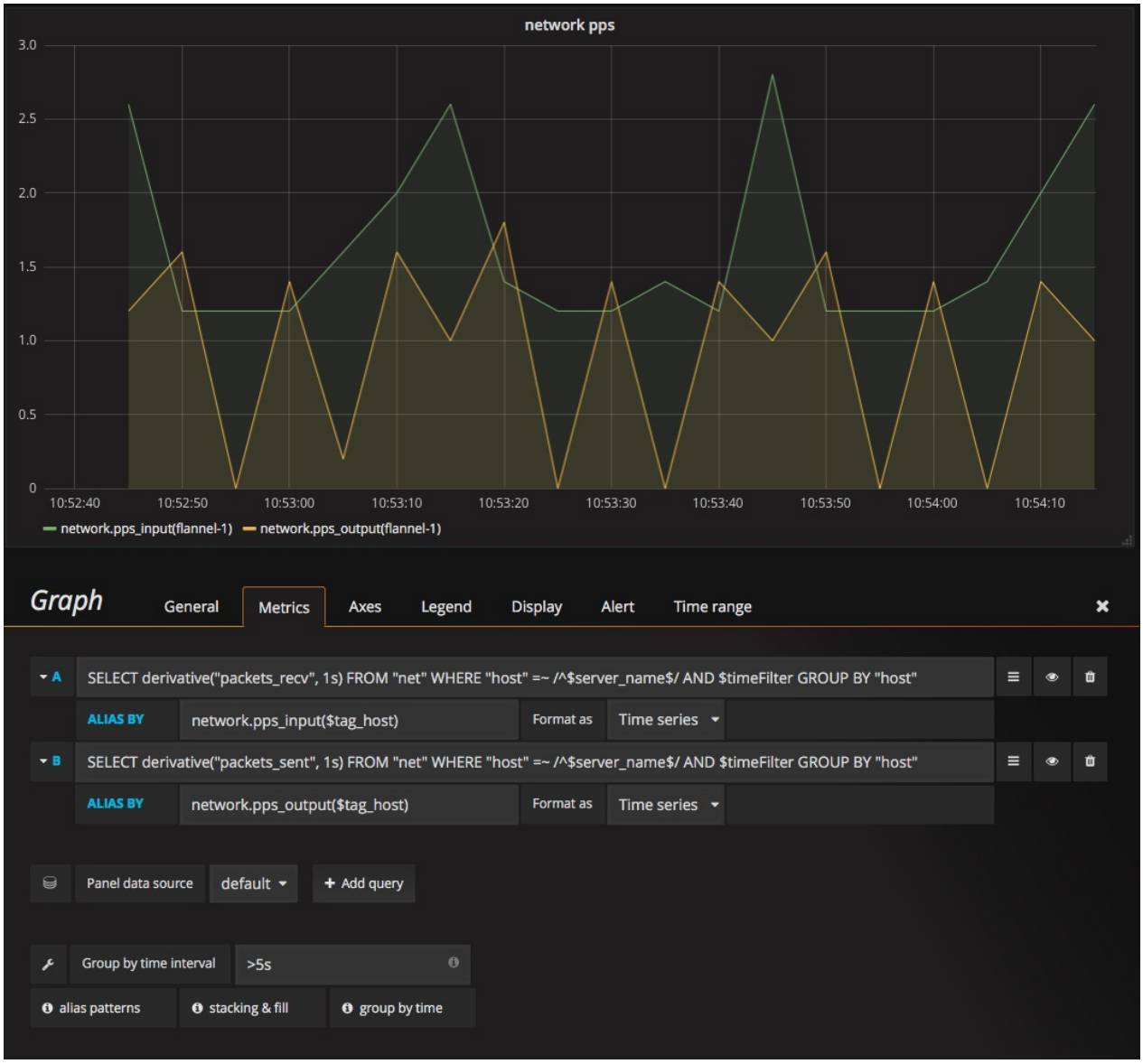
> SELECT derivative("packets_sent", 1s) FROM "net" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY "host"

Axes标签

在Axes中可以设置坐标轴的单位为:

LeffY - Uint - node - none

如下图:



tcp连接数监控

tcp连接数来自net表中的tcp_currestab字段。

Metrics标签

tcp连接数监控的查询语句:

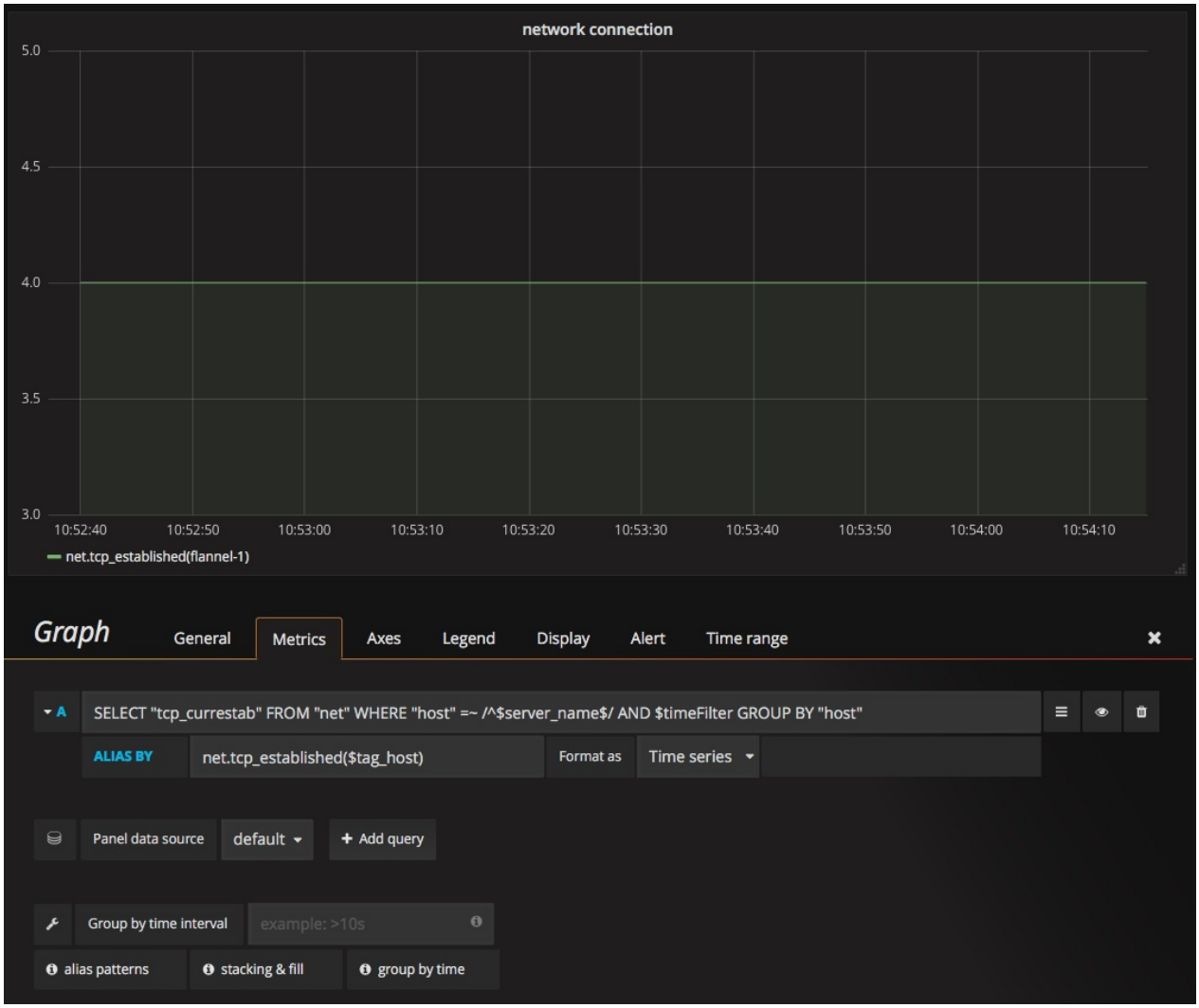
> SELECT "tcp_currestab" FROM "net" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY "host"

Axes标签

在Axes中可以设置坐标轴的单位为:

LeffY - Uint - node - none

如下图:



apache/httpd监控

apache需要开启server-status模块，才可以采集监控数据(参考apache启用server-status).

数据源

apache的监控数据存储在telegraf库的apache表中，如下:

```
> show field keys on telegraf from apache
name: apache
fieldKey             fieldType
--------             ---------
BusyWorkers          float
BytesPerReq          float
BytesPerSec          float
CPULoad              float
IdleWorkers          float
ReqPerSec            float
TotalAccesses        float
TotalkBytes          float
Uptime               float
scboard_closing      float
scboard_dnslookup    float
scboard_finishing    float
scboard_idle_cleanup float
scboard_keepalive    float
scboard_logging      float
scboard_open         float
scboard_reading      float
scboard_sending      float
scboard_starting     float
scboard_waiting      float
```

示例数据

```
# telegraf --input-filter apache -test
......
scboard_reading=0,TotalAccesses=6015163,CPULoad=1.80276,
scboard_sending=1,scboard_open=245,scboard_dnslookup=0,
BytesPerSec=1412750,BytesPerReq=5263.07,IdleWorkers=9,scboard_st
arting=0,
scboard_keepalive=1,scboard_finishing=0,scboard_idle_cleanup=0,
ReqPerSec=268.426,BusyWorkers=2,scboard_closing=0,
scboard_logging=0,TotalkBytes=30916230,Uptime=22409,
scboard_waiting=9 1495293442000000000
```

字段说明

- BusyWorkers: 表示正在工作的进程数
- IdleWorkers: 表示空闲的进程数(BusyWorkers+IdleWorkers和ps -f -u apache | wc -l命令的输出是一样的)
- BytesPerReq: 表示每个请求响应的大小，是从服务启动时间开始算的
- BytesPerSec: 表示每秒请求响应的总大小，是从服务启动时间开始算的
- TotalAccesses: 总的访问次数(用来计算每秒请求数)
- TotalkBytes: 总数据量(用来计算每秒数据量的大小)

监控配置

worker数量监控

数据源为**BusyWorkers**字段和**IdleWorkers**字段.

1.busyworker的查询语句如下:

> SELECT mean("BusyWorkers") FROM "apache" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)

2.idelworker的查询语句如下:

> SELECT mean("IdleWorkers") FROM "apache" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY time($interval), "host" fill(null)

每秒请求数监控

数据源为**TotalAccesses**字段.

每秒请求数的查询语句如下:

> SELECT derivative("TotalAccesses", 1s) FROM "apache" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY "host"

每秒总请求大小监控

数据源为**TotalkBytes**字段.

查询语句如下，注意单位是KB/s:

> SELECT derivative("TotalkBytes", 1s) FROM "apache" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY "host"

每次请求的数据大小监控

数据源为**TotalAccesses**字段和**TotalkBytes**字段.

查询语句如下，注意单位是KB:

> SELECT derivative("TotalkBytes", 1s)/derivative("TotalAccesses", 1s) FROM "apache" WHERE "host" =~ /^$server_name$/ AND $timeFilter GROUP BY "host"

监控配置完后，使用ab开1000个并发压测了一把，效果图如下:

http响应时间监控

数据源

apache的监控数据存储在telegraf库的http_response表中，如下:

```
> show field keys on telegraf from http_response
name: http_response
fieldKey           fieldType
--------           ---------
http_response_code integer
response_time      float
```

示例数据

```
# telegraf --input-filter http_response --test
* Plugin: inputs.http_response, Collection 1
> http_response,server=http://10.10.10.10,method=GET,host=xxx,
response_time=0.0011195200000000002,http_response_code=403i
1495353263000000000
```

字段说明

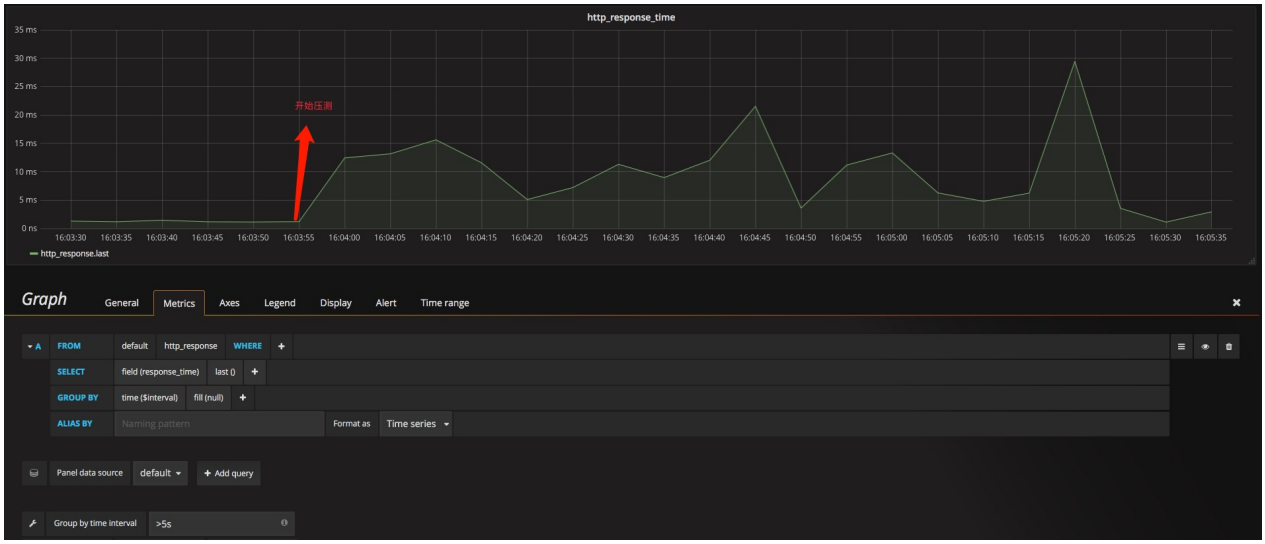
- http_response_code: 响应码
- response_time: 响应时间，单位为秒

监控配置

响应时间的查询语句如下:

> SELECT last("response_time") FROM "http_response" WHERE $timeFilter GROUP BY time($interval) fill(null)

使用ab -c 1000 -n 1000000 url压测httpd后，可以看到响应时间明显上升。如下图:


http_response_time
35 ms
30 ms
25 ms
20 ms
15 ms
10 ms
5 ms
0 ns
开始压测
16:03:30 16:03:35 16:03:40 16:03:45 16:03:50 16:03:55 16:04:00 16:04:05 16:04:10 16:04:15 16:04:20 16:04:25 16:04:30 16:04:35 16:04:40 16:04:45 16:04:50 16:04:55 16:05:00 16:05:05 16:05:10 16:05:15 16:05:20 16:05:25 16:05:30 16:05:35
http_response.last
Graph
General
Metrics
Axes
Legend
Display
Alert
Time range
A
FROM
default
http_response
WHERE
SELECT
field (response_time)
last ()
GROUP BY
time ($interval)
fill (null)
ALIAS BY
Naming pattern
Format as
Time series
Panel data source
default
Add query
Group by time interval
>5s

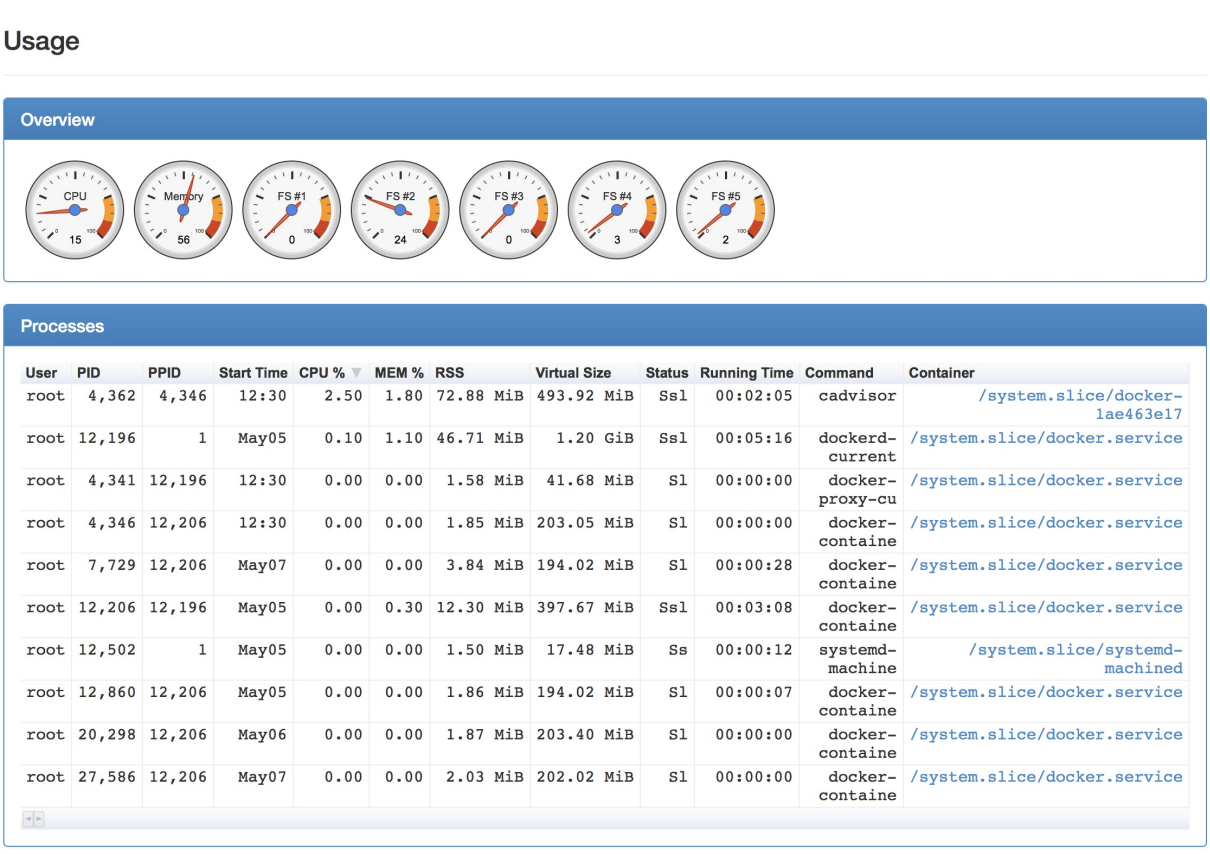
cAdvisor

cAdvisor是Google提供一套容器监控工具

cAdvisor的使用很简单,在宿主机上启动一个cadvisor的容器就可以了,如下:

```
# 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
```

然后在浏览器中打开 http://IP:8080 就可以看到容器的监控信息了。如下图:



cAdvisor可以监控容器的cpu、内存、网络和fs信息。

但是cAdvisor是通过查询docker api得到实时数据进行展示的,不能查看历史数据;并且只能监控一个宿主机。

所以cAdvisor单独使用意义不大,可以将cAdvisor的数据写到InfluxDB中,然后通过Grafana进行展示。具体配置参考下一篇文章(cAdvisor+InfluxDB+Grafana集成)。

cAdvisor+InfluxDB+Grafana集成

- InfluxDB的安装配置参考这里
- Grafana安装与配置参考这里
- InfluxDB和Grafana的集成参考这里

下面主要演示cAdvisor收集的数据怎么写入到InfluxDB中。

参考: https://www.brianchristner.io/how-to-setup-docker-monitoring/

启动容器时加上InfluxDB相关的参数:

```
# docker run --volume=/:/rootfs:ro --volume=/var/run:/var/run:rw
  \
    --volume=/sys:/sys:ro --volume=/var/lib/docker/:/var/lib/doc
ker:ro
    --publish=8080:8080 --detach=true --link influxsrv:influxsrv
 --name=cadvisor \
    google/cadvisor:latest
    -storage_driver_db=cadvisor
    -storage_driver_host=IP:8086
    -storage_driver_user=cadvisor
    -storage_driver_password=password
```

发现一个坑,cAdvisor连接InfluxDB时,虽然可以配置用户名和密码,但是没有TLS相关的配置(kafka倒是有)。 cadvisor的配置如下:

```
/ # cadvisor  -h
Usage of cadvisor:
  -allow_dynamic_housekeeping
        Whether to allow the housekeeping interval to be dynamic
 (default true)
  -alsologtostderr
        log to standard error as well as files
  -application_metrics_count_limit int
        Max number of application metrics to store (per containe
r) (default 100)
```

```
  -boot_id_file string
        Comma-separated list of files to check for boot-id. Use
the first one that exists. (default "/proc/sys/kernel/random/boo
t_id")
  -bq_account string
        Service account email
  -bq_credentials_file string
        Credential Key file (pem)
  -bq_id string
        Client ID
  -bq_project_id string
        Bigquery project ID
  -bq_secret string
        Client Secret (default "notasecret")
  -collector_cert string
        Collector's certificate, exposed to endpoints for certif
icate based authentication.
  -collector_key string
        Key for the collector's certificate
  -container_hints string
        location of the container hints file (default "/etc/cadv
isor/container_hints.json")
  -disable_metrics metrics
        comma-separated list of metrics to be disabled. Options
are 'disk', 'network', 'tcp'. Note: tcp is disabled by default d
ue to high CPU usage. (default tcp)
  -docker string
        docker endpoint (default "unix:///var/run/docker.sock")
  -docker_env_metadata_whitelist string
        a comma-separated list of environment variable keys that
 needs to be collected for docker containers
  -docker_only
        Only report docker containers in addition to root stats
  -docker_root string
        DEPRECATED: docker root is read from docker info (this i
s a fallback, default: /var/lib/docker) (default "/var/lib/docke
r")
  -enable_load_reader
        Whether to enable cpu load reader
  -event_storage_age_limit string
```

```
        Max length of time for which to store events (per type).
 Value is a comma separated list of key values, where the keys a
re event types (e.g.: creation, oom) or "default" and the value
is a duration. Default is applied to all non-specified event typ
es (default "default=24h")
  -event_storage_event_limit string
        Max number of events to store (per type). Value is a com
ma separated list of key values, where the keys are event types
(e.g.: creation, oom) or "default" and the value is an integer.
Default is applied to all non-specified event types (default "de
fault=100000")
  -global_housekeeping_interval duration
        Interval between global housekeepings (default 1m0s)
  -housekeeping_interval duration
        Interval between container housekeepings (default 1s)
  -http_auth_file string
        HTTP auth file for the web UI
  -http_auth_realm string
        HTTP auth realm for the web UI (default "localhost")
  -http_digest_file string
        HTTP digest file for the web UI
  -http_digest_realm string
        HTTP digest file for the web UI (default "localhost")
  -listen_ip string
        IP to listen on, defaults to all IPs
  -log_backtrace_at value
        when logging hits line file:N, emit a stack trace
  -log_cadvisor_usage
        Whether to log the usage of the cAdvisor container
  -log_dir string
        If non-empty, write log files in this directory
  -logtostderr
        log to standard error instead of files
  -machine_id_file string
        Comma-separated list of files to check for machine-id. U
se the first one that exists. (default "/etc/machine-id,/var/lib
/dbus/machine-id")
  -max_housekeeping_interval duration
        Largest interval to allow between container housekeeping
s (default 1m0s)
```

```
  -max_procs int
        max number of CPUs that can be used simultaneously. Less
 than 1 for default (number of cores).
  -port int
        port to listen (default 8080)
  -profiling
        Enable profiling via web interface host:port/debug/pprof
/
  -prometheus_endpoint string
        Endpoint to expose Prometheus metrics on (default "/metr
ics")
  -stderrthreshold value
        logs at or above this threshold go to stderr
  -storage_driver driver
        Storage driver to use. Data is always cached shortly in
memory, this controls where data is pushed besides the local cac
he. Empty means none. Options are: <empty>, bigquery, elasticsea
rch, influxdb, kafka, redis, statsd, stdout
  -storage_driver_buffer_duration duration
        Writes in the storage driver will be buffered for this d
uration, and committed to the non memory backends as a single tr
ansaction (default 1m0s)
  -storage_driver_db string
        database name (default "cadvisor")
  -storage_driver_es_enable_sniffer
        ElasticSearch uses a sniffing process to find all nodes
of your cluster by default, automatically
  -storage_driver_es_host string
        ElasticSearch host:port (default "http://localhost:9200"
)
  -storage_driver_es_index string
        ElasticSearch index name (default "cadvisor")
  -storage_driver_es_type string
        ElasticSearch type name (default "stats")
  -storage_driver_host string
        database host:port (default "localhost:8086")
  -storage_driver_kafka_broker_list string
        kafka broker(s) csv (default "localhost:9092")
  -storage_driver_kafka_ssl_ca string
        optional certificate authority file for TLS client authe
```

```
ntication
  -storage_driver_kafka_ssl_cert string
        optional certificate file for TLS client authentication
  -storage_driver_kafka_ssl_key string
        optional key file for TLS client authentication
  -storage_driver_kafka_ssl_verify
        verify ssl certificate chain (default true)
  -storage_driver_kafka_topic string
        kafka topic (default "stats")
  -storage_driver_password string
        database password (default "root")
  -storage_driver_secure
        use secure connection with database
  -storage_driver_table string
        table name (default "stats")
  -storage_driver_user string
        database username (default "root")
  -storage_duration duration
        How long to keep data stored (Default: 2min). (default 2
m0s)
  -v value
        log level for V logs
  -version
        print cAdvisor version and exit
  -vmodule value
        comma-separated list of pattern=N settings for file-filt
ered logging
```