# Home · jmxtrans/jmxtrans Wiki

https://github.com/jmxtrans/jmxtrans/wiki

**Home**

Rob edited this page on 21 Sep 2016 · [12 revisions](https://github.com/jmxtrans/jmxtrans/wiki/Home/_history)

 Pages 28

Content

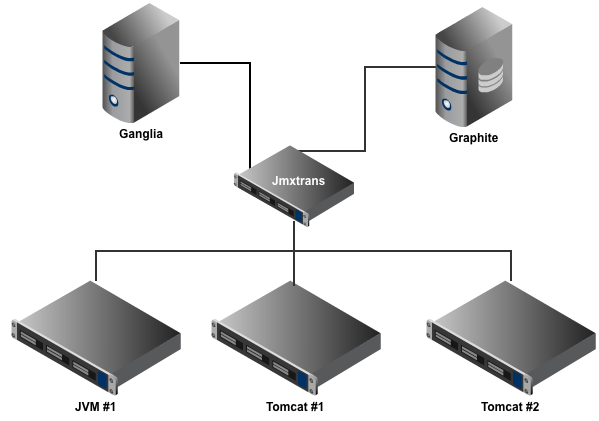
* [Installation](https://github.com/jmxtrans/jmxtrans/wiki/Installation)
* [ChangeLog](https://github.com/jmxtrans/jmxtrans/wiki/ChangeLog)
* [Queries](https://github.com/jmxtrans/jmxtrans/wiki/Queries)
* ... [MoreExamples](https://github.com/jmxtrans/jmxtrans/wiki/MoreExamples)
* ... [YAMLConfig](https://github.com/jmxtrans/jmxtrans/wiki/YAMLConfig)
* [OutputWriters](https://github.com/jmxtrans/jmxtrans/wiki/OutputWriters)
* ... [CloudWatchWriter](https://github.com/jmxtrans/jmxtrans/wiki/CloudWatchWriter)
* ... [GangliaWriter](https://github.com/jmxtrans/jmxtrans/wiki/GangliaWriter)
* ... [GraphiteWriter](https://github.com/jmxtrans/jmxtrans/wiki/GraphiteWriter)
* ... [KeyOutWriter](https://github.com/jmxtrans/jmxtrans/wiki/KeyOutWriter)
* ... [OpenTSDBWriter](https://github.com/jmxtrans/jmxtrans/wiki/OpenTSDBWriter)
* ... [RRDToolWriter](https://github.com/jmxtrans/jmxtrans/wiki/RRDToolWriter)
* ... [RRDWriter](https://github.com/jmxtrans/jmxtrans/wiki/RRDWriter)
* ... [StatsDWriter](https://github.com/jmxtrans/jmxtrans/wiki/StatsDwriter)
* ... [StatsDTelegrafWriter](https://github.com/jmxtrans/jmxtrans/wiki/StatsDTelegrafWriter)
* ... [StdoutWriter](https://github.com/jmxtrans/jmxtrans/wiki/StdoutWriter)
* ... [TCollectorUDPWriter](https://github.com/jmxtrans/jmxtrans/wiki/TCollectorUDPWriter)
* ... [LibratoWriter](https://github.com/jmxtrans/jmxtrans/wiki/LibratoWriter)
* ... [Log4JWriter](https://github.com/jmxtrans/jmxtrans/wiki/Log4JWriter)
* ... [SensuWriter](https://github.com/jmxtrans/jmxtrans/wiki/SensuWriter)
* ... [InfluxDBWriter](https://github.com/jmxtrans/jmxtrans/wiki/InfluxDBWriter)
* [BestPractices](https://github.com/jmxtrans/jmxtrans/wiki/BestPractices)
* [Thanks](https://github.com/jmxtrans/jmxtrans/wiki/Thanks)

Clone this wiki locally

[**Clone in Desktop**](github-windows://openRepo/https:/github.com/jmxtrans/jmxtrans.wiki)

Introduction to jmxtrans

Introduction



jmxtrans is a tool which allows you to connect to any number of Java Virtual Machines (JVMs) and query them for their attributes without writing a single line of Java code. The attributes are exported from the JVM via [Java Management Extensions (JMX)](http://www.oracle.com/technetwork/java/javase/tech/javamanagement-140525.html). Most Java applications have made their statistics available via this protocol and it is possible to add this to any codebase without a lot of effort. If you use the [SpringFramework](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/jmx.html) for your code, it can be as easy as just adding a couple of annotations to a Java class file.

The query language is based on the easy to write [JSON](http://json.org/) format. This allows non-programmers access to JMX without having to know how to write Java. That makes this tool perfect for the busy Ops person.

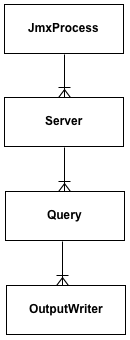
The results of the queries are processed by Java classes called OutputWriters. These are a bit more involved to write because generally it means integrating Java code with a third party tool such as[Graphite](http://graphite.wikidot.com/) or [Ganglia](http://ganglia.sourceforge.net/). Out of the box, jmxtrans supports several output writers and we are encouraging others to suggest new ideas by submitting requests to the [issue tracker](https://github.com/jmxtrans/jmxtrans/issues).

Engine Mode

There are two primary modes for using jmxtrans. The first is to use the JmxTransformer engine included with the distribution. This engine will read a directory of .json files, process them and then create 'jobs' to execute on a [cron-like schedule](http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html). Each job maps to a server that you would like to query jmx attributes on. Therefore, you can setup a complex query schedule for each server by setting the cronExpression field on the Server object (by default it is every minute).

For a given json file, there can be an unlimited number of servers defined within it. The servers are the JVMs that you want to gather stats from and are defined by a hostname, port, username and password.

Within each server, there can be an unlimited number of JMX queries executed against it. Each Query executed against a server can output its results using any number of OutputWriters. jmxtrans includes several different OutputWriters that you can use.



The [Queries] expect an object ("java.lang.type=Memory"), zero or more attributes ["HeapMemoryUsage", "NonHeapMemoryUsage"] and one or more OutputWriters to send the Results of the query to. If you don't specify any attributes, then it will get all of them. You can also specify a star within an object name to query dynamically generated object names.

Performance

The JmxTransformer engine is fully multithreaded. You specify the maximum number of threads that you want to start up for each part of the application. By default, up to 10 servers are queried at the same time. It is also possible to have multiple threads for each query against a server. Thus, you can specify that you want 10 threads to handle your 50 servers. Each one of your servers may have defined 10 queries. You can therefore, set the numQueryThreads to 2 to execute two queries against a server at the same time.

There are two sides to JmxTransformer. The input is the connection to the JMX server running in a JVM. The output is to OutputWriters. As necessary, both sides make use of connection object pools to maintain socket connections to both the input and output.

On a side note, I've heard from a few people now who are using it to monitor clusters of hundreds of machines. If you have even larger clusters, please let me know!

API Mode

The second mode for jmxtrans is to act as an API to build your own application for pulling data from JMX and writing it out. The Engine was written on top of this API. The Engine is how I'd use this project, but maybe you have other ideas so that is fully supported by allowing you to write your own engine.

jmxtrans uses the amazing [Jackson library](https://github.com/FasterXML/jackson) to parse json data into a Java object model. This model is primarily represented by the [JmxProcess, Server, Query, Result](https://github.com/jmxtrans/jmxtrans/tree/master/jmxtrans-core/src/main/java/com/googlecode/jmxtrans/model) objects. This means that if you know a bit of java, it is possible to fully customize your own usage of jmxtrans to however you see fit.

The core of the api is implemented as mostly static methods in the [JmxUtils class](https://github.com/jmxtrans/jmxtrans/blob/master/jmxtrans-core/src/main/java/com/googlecode/jmxtrans/jmx/JmxUtils.java). You pass in a Server object with a bunch of Queries and get back a list of Results. How you process those results is up to you.

This also means that you can use Java'ish languages like Jython, Scala and Groovy to script jmxtrans to do whatever you want.

Take a look at the included [example classes](https://github.com/jmxtrans/jmxtrans/tree/master/jmxtrans-examples/src/main/java/com/googlecode/jmxtrans/example). They show how you can either read a json file from disk into the object model or create the object model by hand and execute it. There is also examples of using wildcards, which jmxtrans fully supports with JDK 6.

# jmxtrans/jmxtrans: jmxtrans

https://github.com/jmxtrans/jmxtrans



This is the source code repository for the jmxtrans project.

This is effectively the missing connector between speaking to a JVM via JMX on one end and whatever logging / monitoring / graphing package that you can dream up on the other end.

jmxtrans is very powerful tool which uses easily generated JSON (or [YAML](https://github.com/jmxtrans/jmxtrans/blob/master/jmxtrans/tools/yaml2jmxtrans.py)) based configuration files and then outputs the data in whatever format you desire. It does this with a very efficient engine design that will scale to communicating with thousands of machines from a single jmxtrans instance.

The core engine is very solid and there are writers for [Graphite](http://graphite.wikidot.com/), [StatsD](https://github.com/etsy/statsd), [Ganglia](http://ganglia.sourceforge.net/), [cacti/rrdtool](http://www.cacti.net/), [OpenTSDB](http://opentsdb.net/), text files, and stdout. Feel free to suggest more on the discussion group or issue tracker.

* [Download a recent stable build](http://central.maven.org/maven2/org/jmxtrans/jmxtrans/) (or a [SNAPSHOT one](https://oss.sonatype.org/content/repositories/snapshots/org/jmxtrans/jmxtrans/))
* See the [Wiki](https://github.com/jmxtrans/jmxtrans/wiki) for full documentation.
* Join the [Google Group](http://groups.google.com/group/jmxtrans) if you have anything to discuss or [follow the commits](http://groups.google.com/group/jmxtrans-commits). Please don't email Jon directly because he just doesn't have enough time to answer every question individually.
* People are [talking - this is me! (skip to 21:45)](http://www.justin.tv/kctv88/b/290736874) and [talking](http://www.slideshare.net/cyrille.leclerc/paris-devops-monitoring-and-feature-toggle-pattern-with-jmx) and [talking (skip to 34:40)](http://www.justin.tv/kctv88/b/288229232) and [(french)](http://www.slideshare.net/henri.gomez/devops-retour-dexprience-marsjug-du-29-juin-2011%20taking) about it.
* If you are seeing duplication of output data, look for 'typeNames' in the documentation.
* If you like this project, please tell your friends, blog & tweet. I'd really love your help getting more publicity.

Coda Hale did [an excellent talk](http://pivotallabs.com/talks/139-metrics-metrics-everywhere) for [Pivotal Labs](http://pivotallabs.com/) on *why* metrics matter. Great justification for using a tool like jmxtrans.

# 使用grafana4的alert功能 - xixicat - SegmentFault

https://segmentfault.com/a/1190000008226841

序

grafana 4版本以上支持了alert功能，这使得利用grafana作为监控面板更为完整，因为只有alert功能才能称得上监控。

**万物docker**

根据[graphite\_docker](https://github.com/SamSaffron/graphite_docker)这个dockerfile来改造下，原本是grafana3的，现改为4版本：

**修改grafana**

在35行，改为获取grafana 4版本的包

# grafana

**run** cd ~ &&\

wget https://grafanarel.s3.amazonaws.com/builds/grafana\_4.1.1-1484211277\_amd64.deb &&\

dpkg -i grafana\_4.1.1-1484211277\_amd64.deb && rm grafana\_4.1.1-1484211277\_amd64.deb

**增加email配置(可选)**

如果需要开启email alert的话，则需要在dockerfile把email的配置提前配置进去，具体在grafana/config.ini文件里头，新增email配置即可。

**notification配置**

**email**

email是比较传统的告警渠道，不过在使用docker版的grafana，需要提前配置下stmp的配置，否则会报错(/var/log/grafana/grafana.log)：

t=2017-01-29T07:34:35+0000 lvl=eror msg="Failed to send alert notification email" logger=alerting.notifier.email error="Grafana mailing/smtp options not configured, contact your Grafana admin"

配置模板如下:

#################################### SMTP / Emailing ##########################

[smtp]

enabled = true

host = smtp.126.com:25

user = xxxxxx

password = xxxxx

;cert\_file =

;key\_file =

skip\_verify = true

from\_address = xxxxxx@126.com

[emails]

;welcome\_email\_on\_sign\_up = false

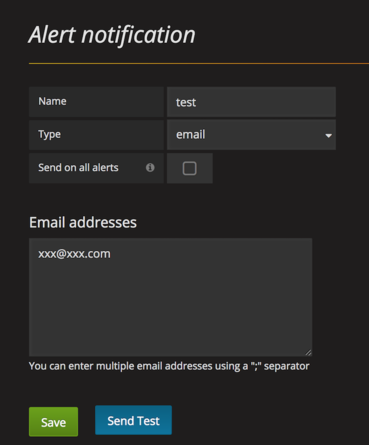
顺带开启下alert配置

#################################### Alerting ######################################

[alerting]

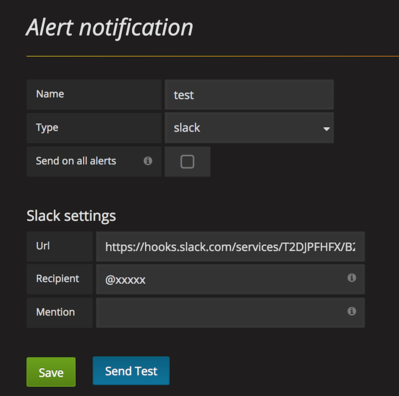
# Makes it possible to turn off alert rule execution.

execute\_alerts = true

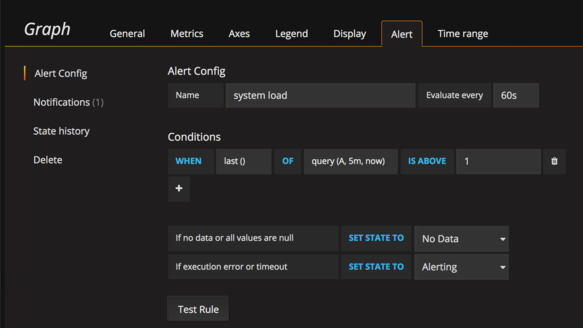
测试一下：  


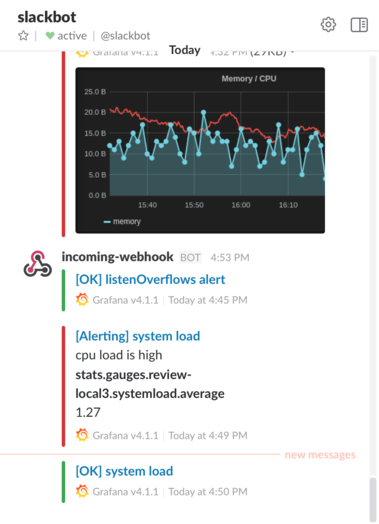
发送不成功的时候，记得去/var/log/grafana/grafana.log看下有没有错误日记，好进行排查。

**slack**

使用slack的话，那就更简单了，不需要在启动之前准备好配置项，只需要在界面上直接配置incoming webhook就可以了，非常适合docker版的grafana：  


**alert**

在每个graph的tab里头有个alert标签，里头可以配置：  


记得添加下notifications就是，然后就大功告成了。  


**doc**

* [grafana\_statsd\_graphite\_docker](https://github.com/patterncat/grafana_statsd_graphite_docker)
* [grafana-configration](http://docs.grafana.org/installation/configuration/#smtp)
* [grafana-alerting-rules](http://docs.grafana.org/alerting/rules/)

# JMXTRANS | communicate with thousands of machines via a single jmx instance

http://www.jmxtrans.org/

# JAVA 监控内容收集之 Jmxtrans Agent - 诡迹 - 51CTO技术博客

http://unixman.blog.51cto.com/10163040/1734975

自从运维TOMCAT 服务，一直令我很困恼的事情是如何对JVM 的健康状况去监控？如何获取这些健康状况的数据？

到底如何去监控JVM 的健康状况?

我想这个最起码要对JVM有个基础的了解，知道它的内存情况、GC情况 等一些细节信息。 有了大体轮廓的了解，对JVM健康状况的监控将不会再茫然。

如何获取JVM 的健康状态数据呢？

网络上的一些方法，基本上都需要开启一个AGENT ，通过这个AGENT 去获取数据。 但这种方式，有时候我想获取到的数据，往往不知道要怎么去表达获取。

比如使用了 cmdline-jmxclient 这个jar 工具

java -jar cmdline-jmxclient-0.10.3.jar - 127.0.0.1:9999 java.lang:type=Memory NonHeapMemoryUsage

可以轻松的去获取到一些 JVM 的简单信息，但是我想获取Mbeans 中的更详细的信息，如 Tomcat 相关的一些信息，真是不知道该如何去书写、表达。

尝试了一些表达形式，根本就获取不到信息(不知道这个东西是否支持这些信息的获取)。

但当你遇到了jmxtrans 和 jmxtrans-agent 后，这种感觉突然有了改观。在这里简单的去介绍一下jmxtrans-agent 的使用，有关详细的说明，还需要大家去github上自己去啃readme。

首先列举出github的地址吧(有可能你要翻墙才能看的到)

Jmxtrans:  https://github.com/jmxtrans/jmxtrans

Jmxtrans-agent:  https://github.com/jmxtrans/jmxtrans-agent

jmxtrans-agent 是什么?

是一个替代jmxtrans 集中式JVM 收集信息的工具，它不需要任何依赖即可使用。

如何去使用 jmxtrans-agent ?

以在  Tomcat 上的使用为例子，在 setenv.sh 中加入如下：

export JAVA\_OPTS="$JAVA\_OPTS -javaagent:/path/to/jmxtrans-agent-1.1.0.jar=jmxtrans-agent.xml"

注：

* javaagent 的路径可以相对于工作目录的路径
* jmxtrans-anget.xml 配置文件的路径可以是一个http(s)的 url 或 为一个相对于classpath 的路径 或为一个具体的路径

**如何去书写jmxtrans-agent 的配置文件 ?**

使用attribute 和 attributes 去收集单个或者多个属性值，多个属性值以逗号分割。

若不指定这个属性，则收集这个objeName 下的所有的属性。

使用 resultAlias 去制定metric 的名称，在resultAlias 中使用关键字 #attribute#" 时， 将指定 metric 的名字为attribute 或 attributes 中使用的值

以下是摘取的官方Example:

Example - collect the ThreadCount attribute from the Threading MBean:

<query objectName="java.lang:type=Threading" attribute="ThreadCount"

   resultAlias="jvm.thread.count"/>

Example - collect ThreadCount and TotalStartedThreadCount from the Threading MBean:

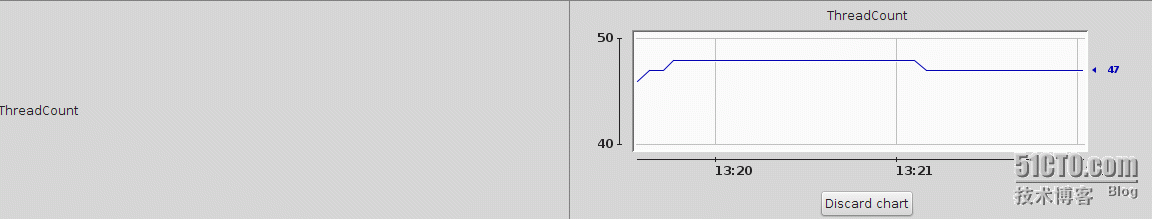
<query objectName="java.lang:type=Threading" attributes="ThreadCount,TotalStartedThreadCount"

  resultAlias="jvm.threads.#attribute#"/>

Example - collect all attributes from the Threading MBean:

<query objectName="java.lang:type=Threading" resultAlias="jvm.threads.#attribute#"/>

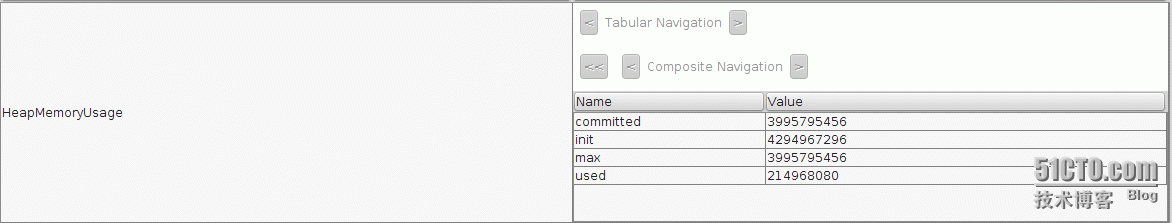
收集属性的单值(收集JVM threading 的 ThreadCount 数据)

[](http://s2.51cto.com/wyfs02/M00/79/A5/wKiom1aXNeaTgCcjAABoTJSVMXI565.png)

<query objectName="java.lang:type=Threading" attribute="ThreadCount"

   resultAlias="jvm.thread"/>

收集属性中的多值(收集 JVM Memory 的 HeapMemoryUsage 中的used 数据)

[](http://s2.51cto.com/wyfs02/M00/79/A4/wKioL1aXN07AOiHBAAChS6HPM40421.png)

使用关键字key 去指定到底收集属性中的那个值。

 <query objectName="java.lang:type=Memory" attribute="HeapMemoryUsage" key="used" resultAlias="jvm.heapMemoryUsage.used"/>

同样可以忽略key去收集所有的值。并在resultAlias 中使用 #key# 去动态的描述metric的名称

<query objectName="java.lang:type=Memory" attribute="HeapMemoryUsage" resultAlias="jvm.heapMemoryUsage.#key#"/>

官方提供的一个简单的jmxtrans-agent.xml

<jmxtrans-agent>

    <queries>        <!-- OS -->

        <query objectName="java.lang:type=OperatingSystem" attribute="SystemLoadAverage" resultAlias="os.systemLoadAverage"/>        <!-- JVM -->

        <query objectName="java.lang:type=Memory" attribute="HeapMemoryUsage" key="used"

               resultAlias="jvm.heapMemoryUsage.used"/>

        <query objectName="java.lang:type=Memory" attribute="HeapMemoryUsage" key="committed"

               resultAlias="jvm.heapMemoryUsage.committed"/>

        <query objectName="java.lang:type=Memory" attribute="NonHeapMemoryUsage" key="used"

               resultAlias="jvm.nonHeapMemoryUsage.used"/>

        <query objectName="java.lang:type=Memory" attribute="NonHeapMemoryUsage" key="committed"

               resultAlias="jvm.nonHeapMemoryUsage.committed"/>

        <query objectName="java.lang:type=ClassLoading" attribute="LoadedClassCount" resultAlias="jvm.loadedClasses"/>

        <query objectName="java.lang:type=Threading" attribute="ThreadCount" resultAlias="jvm.thread"/>        <!-- TOMCAT -->

        <query objectName="Catalina:type=GlobalRequestProcessor,name=\*" attribute="requestCount"

               resultAlias="tomcat.requestCount"/>

        <query objectName="Catalina:type=GlobalRequestProcessor,name=\*" attribute="errorCount"

               resultAlias="tomcat.errorCount"/>

        <query objectName="Catalina:type=GlobalRequestProcessor,name=\*" attribute="processingTime"

               resultAlias="tomcat.processingTime"/>

        <query objectName="Catalina:type=GlobalRequestProcessor,name=\*" attribute="bytesSent"

               resultAlias="tomcat.bytesSent"/>

        <query objectName="Catalina:type=GlobalRequestProcessor,name=\*" attribute="bytesReceived"

               resultAlias="tomcat.bytesReceived"/>        <!-- APPLICATION -->

        <query objectName="Catalina:type=Manager,context=/,host=localhost" attribute="activeSessions"

               resultAlias="application.activeSessions"/>

    </queries>

    <outputWriter class="org.jmxtrans.agent.GraphitePlainTextTcpOutputWriter">

        <host>localhost</host>

        <port>2003</port>

        <namePrefix>app\_123456.servers.i876543.</namePrefix>

    </outputWriter>

    <outputWriter class="org.jmxtrans.agent.ConsoleOutputWriter"/>

    <collectIntervalInSeconds>20</collectIntervalInSeconds>

</jmxtrans-agent>

以上描述了了所有常用的数据采集方式。

既然我们采集数据有了目标，知道了该采集什么样的数据，那么如何获取结果呢?

jmxtrans-agent 支持一下几种结果的存储形式：

* GraphitePlainTextTcpOutputWriter
* FileOverwriterOutputWriter
* SummarizingFileOverwriterOutputWriter
* ConsoleOutputWriter
* SummarizingConsoleOutputWriter
* RollingFileOutputWriter
* StatsDOutputWriter

假如说就是为了测试，我们可以尝试使用 ConsoleOutputWriter、FileOverwriterOutputWriter、RollingFileOutputWriter  这三种格式。

ConsoleOutputWriter ： 顾名思义，应该是将结果输出到了终端上吧(这个我没有去测试)。

FileOverwriterOutputWriter、RollingFileOutputWriter ： 这两种格式将结果存储到了文件中，而RollingFileOutputWriter 这个又更高级一些，支持文件的切割的一些功能。

StatsDOutputWriter、GraphitePlainTextTcpOutputWriter ： 这些应该结合 StatsD  和  Graphite 这样的开源画图工具去展示的吧。

暂时写这么多，也是初步的去了解。期望共同学习进步。多谢！

本文出自 “[诡迹](http://unixman.blog.51cto.com/)” 博客，请务必保留此出处<http://unixman.blog.51cto.com/10163040/1734975>

# jmxtrans监控kafka - 阳光丶微笑 - 博客园

http://www.cnblogs.com/ygwx/p/5411990.html

我们知道jmx可以将程序内部的信息暴露出来，但是要想监控这些信息的话，就还需要自己写java程序调用jmx接口去获取数据，并按照某种格式发送到其他地方（如监控程序zabbix,ganglia）。这时jmxtrans就派上用场了，jmxtrans的作用是自动去jvm中获取所有jmx格式数据，并按照某种格式（json文件配置格式）输出到其他应用程序（常用的有ganglia）

安装

[复制代码](javascript:void(0);)

主页：https://github.com/jmxtrans/jmxtrans（这里面也有一个下载地址，貌似版本更高）

下载地址：https://github.com/jmxtrans/jmxtrans/downloads

sudo yum install jmxtrans-20121016.145842.6a28c97fbb-0.noarch.rpm

jmxtrans安装目录：/usr/share/jmxtrans

jmxtrans配置文件 ：/etc/sysconfig/jmxtrans

json文件默认目录：/var/lib/jmxtrans/

日志路径：/var/log/jmxtrans/jmxtrans.log

[复制代码](javascript:void(0);)

启动

sudo /usr/share/jmxtrans/jmxtrans.sh start

sudo /usr/share/jmxtrans/jmxtrans.sh start /var/lib/jmxtrans/kafka.json # 也可以指定json文件路径

停止

sudo /usr/share/jmxtrans/jmxtrans.sh stop

监控jvm

jvm需要加入以下参数

-Dcom.sun.management.jmxremote

-Dcom.sun.management.jmxremote.port=9999

-Dcom.sun.management.jmxremote.authenticate=false

-Dcom.sun.management.jmxremote.ssl=false

监控kafka示例

配置kafka参数

第一步: 在kafka集群的所有机器上安装JMXTrans第二步: 编辑 “kafka-run-class.sh”

　　　　KAFKA\_JMX\_OPTS="-Dcom.sun.management.jmxremote=true -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false "第三步: 编辑 “kafka-server-start.sh”

　　　　export JMX\_PORT=${JMX\_PORT:-9999}

监控kafka信息发送到ganglia,/var/lib/jmxtrans/kafka.json

[复制代码](javascript:void(0);)

{

"servers": [

{

"port": "9999", # jmx 端口

"host": "127.0.0.1", # jmx 的主机

"queries": [

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",

"settings": {

"groupName": "jvmheapmemory",

"port": 8649,

"host": "127.0.0.1"

}

}

],

"obj": "java.lang:type=Memory",

"resultAlias": "heap",

"attr": [

"HeapMemoryUsage",

"NonHeapMemoryUsage"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",

"settings": {

"groupName": "kafka topic stats",

"port": 8649, # ganglia的gmond端口

"host": "127.0.0.1", # ganglia 主机

"typeNames":[

"name"

]

}

}

],

"obj": "kafka.server:type=BrokerTopicMetrics,name=\*",

"resultAlias": "Kafka",

"attr": [

"Count",

"OneMinuteRate"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",

"settings": {

"groupName": "kafka server request",

"port": 8649,

"host": "127.0.0.1"

}

}

],

"obj":"kafka.server:type=OffsetManager,name=NumGroups",

"resultAlias": "OffsetManager NumGroups",

"attr": [

"Value"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",

"settings": {

"groupName": "jvmGC",

"port": 8649,

"host": "127.0.0.1",

"typeNames": [

"name"

]

}

}

],

"obj": "java.lang:type=GarbageCollector,name=\*",

"resultAlias": "GC",

"attr": [

"CollectionCount",

"CollectionTime"

]

}

],

"numQueryThreads": 2

}

]

}

[复制代码](javascript:void(0);)

# JMXtrans + InfluxDB + Grafana 实现 Kafka 性能指标监控-搜狐

http://mt.sohu.com/20161106/n472443549.shtml

架构

　　一般系统监控通常分为3部分：

1. 数据采集
2. 分析与转换
3. 展现(可视化)

数据采集

　　对于前端应用，一般需要埋点，对用户的行为进行记录。 如果不埋点，则需要通过 Pagespeed、PhantomJS 这样的工具去模拟用户行为进行测试。后端的系统通常有自己的性能指标。我们可以通过命令／脚本的方式进行采集。

　　JMX（Java Management Extensions，即 Java 管理扩展）是一个为应用程序、设备、系统等管理功能的框架，通常可以用来监控和管理 Java 应用系统。

　　Kafka 做为一款Java应用，已经定义了丰富的性能指标，(可以参考 Kafka监控指标)，通过 JMX 可以轻松对其进行监控。

　　测试

　　首先需要在 Kafka 上打开 JMX

　　1.修改 ${kafka\_home}/bin/kafka-server-start.sh , 增加一个 JMX\_PORT 的配置，指定一个端口用于接受外部连接，注意如部署、运行在非 root 用户下，必须指定 1024以上端口

　　if[ "x$KAFKA\_HEAP\_OPTS"= "x"]; then export KAFKA\_HEAP\_OPTS= "-Xmx1G -Xms1G"export JMX\_PORT= "9999"fi

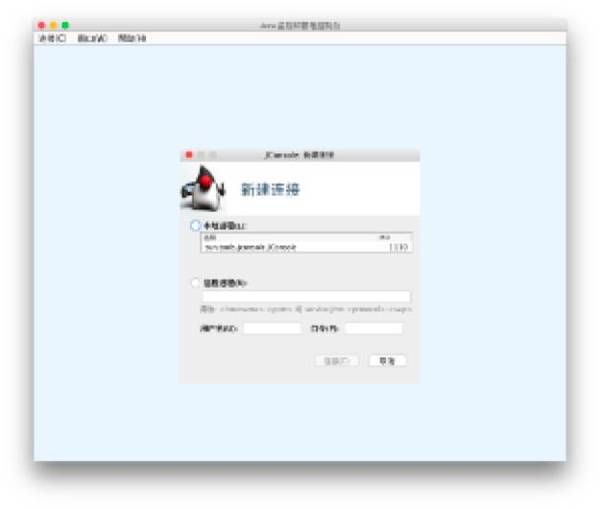
　　2.重启kafka

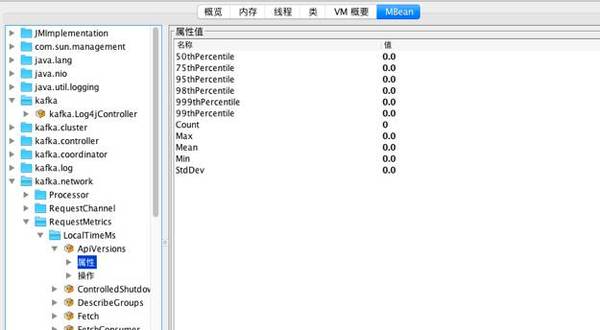
　　bin/kafka-server-stop.sh bin/kafka-server-start.sh config/server.properties &

　　3.重启后观察可以发现JMX已经启动了

　　$ps -ef | grep kafka www 5106510Nov03 ? 00: 08: 20/opt/vdian/java/bin/java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis= 20-XX:InitiatingHeapOccupancyPercent= 35-XX:+DisableExplicitGC -Djava.awt.headless= true-Xloggc: /home/www/kafka\_2 .11-0.10.0.1/bin/../logs/kafkaServer-gc.log -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:+PrintGCTimeStamps -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate= false-Dcom.sun.management.jmxremote.ssl=false-Dcom.sun.management.jmxremote.port= 9999-Dkafka.logs.dir= /home/www/kafka\_2 .11-0.10.0.1/bin/../logs -Dlog4j.configuration=file:bin/../config/log4j.properties -cp :/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/aopalliance-repackaged- 2.4.0-b34.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/argparse4j- 0.5.0.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/connect-api- 0.10.0.1.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/connect-file- 0.10.0.1.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/connect-json- 0.10.0.1.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/connect-runtime- 0.10.0.1.jar: /home/www/kafka\_2.11- 0.10.0.1/bin/../libs/guava- 18.0.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/hk2-api- 2.4.0-b34.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/hk2-locator- 2.4.0-b34.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/hk2-utils- 2.4.0-b34.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/jackson-annotations- 2.6.0.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/jackson-core- 2.6.3.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jackson-databind- 2.6.3.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jackson-jaxrs-base- 2.6.3.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jackson-jaxrs-json-provider- 2.6.3.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jackson- module-jaxb-annotations- 2.6.3.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/javassist- 3.18.2-GA.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/javax.annotation-api- 1.2.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/javax.inject- 1.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/javax.inject-2.4.0-b34.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/javax.servlet-api- 3.1.0.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/javax.ws.rs-api- 2.0.1.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/jersey-client- 2.22.2.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jersey-common- 2.22.2.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jersey-container-servlet-2.22.2.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jersey-container-servlet-core- 2.22.2.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jersey-guava- 2.22.2.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/jersey-media-jaxb- 2.22.2.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/jersey-server- 2.22.2.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-continuation- 9.2.15.v20160210.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-http-9.2.15.v20160210.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-io- 9.2.15.v20160210.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-security- 9.2.15.v20160210.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-server- 9.2.15.v20160210.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-servlet- 9.2.15.v20160210.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-servlets- 9.2.15.v20160210.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/jetty-util- 9.2.15.v20160210.jar: /home/www/kafka\_2.11- 0.10.0.1/bin/../libs/jopt-simple- 4.9.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/kafka-clients- 0.10.0.1.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/kafka-log4j-appender- 0.10.0.1.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/kafka-streams- 0.10.0.1.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/kafka-streams-examples- 0.10.0.1.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/kafka-tools- 0.10.0.1.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/kafka\_2.11- 0.10.0.1-sources.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/kafka\_2 .11- 0.10.0.1-test-sources.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/kafka\_2 .11- 0.10.0.1.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/log4j- 1.2.17.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/lz4- 1.3.0.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/metrics-core- 2.2.0.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/osgi-resource-locator- 1.0.1.jar: /home/www/kafka\_2.11- 0.10.0.1/bin/../libs/reflections- 0.9.10.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/rocksdbjni- 4.8.0.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/scala-library-2.11.8.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/scala-parser-combinators\_2 .11- 1.0.4.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/slf4j-api- 1.7.21.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/slf4j-log4j12- 1.7.21.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/snappy-java- 1.1.2.6.jar: /home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/validation-api- 1.1.0.Final.jar:/home/www/kafka\_2 .11- 0.10.0.1/bin/../libs/zkclient- 0.8.jar: /home/www/kafka\_2 .11-0.10.0.1/bin/../libs/zookeeper- 3.4.6.jar kafka.Kafka config/server.properties $ netstat -an | grep9999tcp 00::: 9999:::\* LISTEN

　　4.在安装了java的主机上运行jconsole，就会弹出一个控制台，在可以看到MBean中的性能指标





　　指标采集

　　传统的数据采集方案，一般是先开发数据采集的脚本，然后借助 nagios、zabbix 这样的监控软件来调度执行，并将采集到的数据进行上报。对于 java 应用，给大家介绍一个新朋友 jmxtrans。

　　读取 json 或 yaml 格式的配置文件，通过 jmx 采集 java 性能指标。支持输出到 Graphite、InfluxDB、RRDTool 等。

　　安装部署

　　1.首先下载 jmxtrans 的 RPM 包，地址

　　2.安装 jdk1.8

　　yum install -y java- 1.8.0-openjdk.x86\_64 java- 1.8.0-openjdk-devel.x86\_64

　　3.设置 JAVA\_HOME 和 PATH 环境变量

　　export JAVA\_HOME= /usr/lib/jvm/java- 1.8.0-openjdk- 1.8.0.65- 0.b17.el6\_7.x86\_64 export PATH=$JAVA\_HOME/bin: /usr/share/jmxtrans/bin:$PATH

　　4.编写配置文件

　　下文是一段测试用的配置，采集的数据会输出到日志中显示。

　　{

　　"servers": [ {

　　"port": "jmx端口",

　　"host": "IP地址",

　　"queries": [ {

　　"outputWriters": [ {

　　"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"} ],

　　"obj": "java.lang:type=Memory",

　　"attr": [ "HeapMemoryUsage", "NonHeapMemoryUsage"] }, {

　　"outputWriters": [ {

　　"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"} ],

　　"obj": "java.lang:name=CMS Old Gen,type=MemoryPool",

　　"attr": [ "Usage"] }, {

　　"outputWriters": [ {

　　"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"} ],

　　"obj": "java.lang:name=ConcurrentMarkSweep,type=GarbageCollector",

　　"attr": [ "LastGcInfo"] } ],

　　"numQueryThreads": 2} ] }

　　5.配置文件编辑完成后，将其放在/var/lib/jmxtrans/目录中

　　6.启动 jmxtrans

　　jmxtrans start

　　jmxtrans 会以后台 deamon 的形式运行，每隔1分钟采集一次数据

　　部署方式

　　由于 jmx 是通过网络连接的，因此 JMXtrans 的部署方案有 2 种

1. 集中式，在一台服务器上部署一个 JMXtrans，分别连接至所有的 Kafka 实例，并将数据吐到 InfluxDB。为了减少网络传输，可以考虑部署到 InfluxDB 的服务器上
2. 分布式，每个 Kafka 实例部署一个 JMXtrans

　　这里我们采用了方案2

　　P.S 如果 JMX 能够支持 UNIX socket 方式连接方案就更完美了。socket 连接较 TCP 连接开销更小，效率更高，非常适合同一台服务器上2个进程之间的通信

　　分析与转换

　　由于 Kafka 的性能数据非常全面，大部分指标已经做了分析了。



　　类似上述，指标的直方图，次数，最大最小、标准方差都已经计算好了，因此我们不再对数据再做加工。

　　这里只取了每项指标的 Mean 项(中位数) 做为指标值写入 InfluxDB 当中。

　　展现

　　Grafana 是一款非常强大的纯前端的画图软件，可以说画什么图，动动鼠标就可以配出来。

　　设置变量模版

　　支持变量模版，例如下图，我们将展现的指标所属的实例定义为变量



　　之后在图表的左上角就可以通过下拉框的方式选择要查看的实例的图表，支持多选

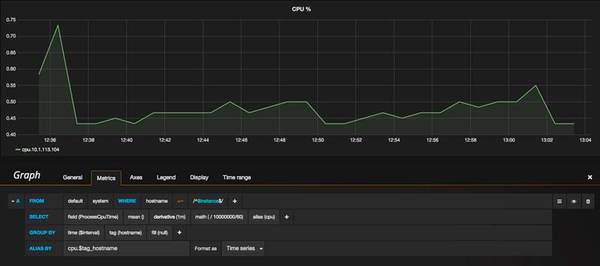
　　支持前端对数据进行加工

　　例如 CPU 使用率数据，我们采集到的数据实际是一个计数器，记录了采集时的 CPU 时间。

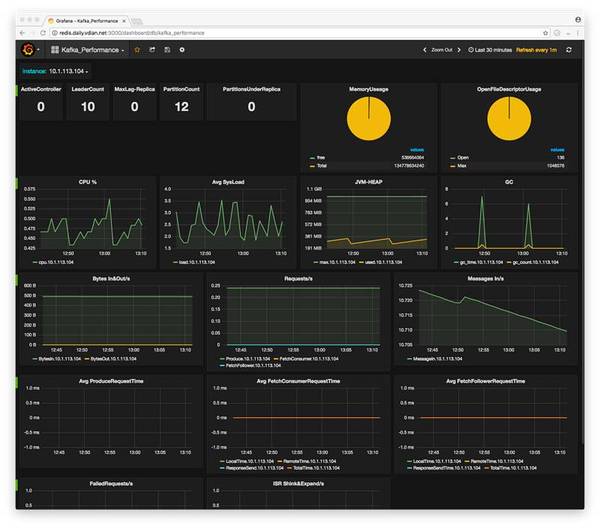
　　使用率我是这样定义的， 采集 2 次

　　CPU使用率 = (结束数据-开始数据)/采集间隔时间

　　这里通过配置的方式实现了数据加工与展现。



　　最后图表页面是这样一个效果，是不是非常炫酷呢



　　尾声

　　Tomcat、JBoss 这类的JAVA应用都支持 JMX，下面的还用我说么？

# JMXTrans、graphite、icinga集成方式 - u012333307的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/u012333307/article/details/42609595

1 采用jmxtrans统一封装 jmx数据上报到告警平台icinga和报表平台graphite

  1.1 jmxtrans安装

      wget https://github.com/downloads/jmxtrans/jmxtrans/jmxtrans-20121016.145842.6a28c97fbb-0.noarch.rpm

      rpm -i jmxtrans-20121016.145842.6a28c97fbb-0.noarch.rpm

      替换jmxtrans-all.jar包

      /usr/share/jmxtrans

  1.2 应用程序部署目录

      /usr/share/jmxtrans

  1.3 配置文件目录

      /etc/sysconfig/jmxtrans

  1.4 服务脚本

      /etc/init.d/jmxtrans

  1.5 采集配置文件

      /var/lib/jmxtrans

  1.6 启动命令

      采用service 方式启动

      service jmxtrans start  
  
1.7 修改规则文件的路径

      定义JSON\_DIR环境变量，可以修改规则文件的目录，否则默认在 /var/lib/jmxtrans目录下面。规则文件样例:  
      下述为和icinga对接规则文件样例:

{

    "servers": [

        {

            "port": "10008",

            "host": "xx.xx.xx.xx",

            "alias": "test",

            "numQueryThreads":5,

            "queries": [

                {

                    "obj": "metrics:name=mq-message-handle-timer",

                    "resultAlias": "mq\_message\_handle\_time",

                    "outputWriters": [

                        {

                            "@class": "com.googlecode.jmxtrans.model.output.NagiosWriter",

                            "outputFile": "/usr/local/icinga/var/rw/icinga.cmd",

                            "nagiosHost": "xx\_xx\_xx\_xx",

                            "prefix": "mq\_message\_handle\_time\_",

                            "suffix": "",

                            "filters": [

                                "Max",

                                "95thPercentile"

                            ],

                            "thresholds": [

                                "~:10000",

                                "~:50000"

                            ]

                        }

                    ]

                }

            ]

        }

    ]

}  
  
1.8 jmxtrans服务自启动

      chkconfig jmxtrans on

  1.9 业务进程增加rmi端口监听

      -Dcom.sun.management.jmxremote.port=10008 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false

  2.0 icinga对接

      1. 检查icinga的外部命令文件

         默认配置: /usr/local/icinga/var/rw/icinga.cmd

         (安装目录: /usr/local/icinga)

         检查该文件的权限

      2. 配置jmxtrans的icinga

      3. 增加icinga的对象文件

         样例:

   define host{

        use                  generic-linux-host,host-pnp4

        host\_name              115\_29\_250\_113

        alias                   linux-host

        icon\_image              redhat.gif

        statusmap\_image         redhat.gd2

        address                 115.29.250.113

}

define host{

        use                  generic-linux-host,host-pnp4

        host\_name              114\_215\_204\_97

        alias                   linux-host

        icon\_image              redhat.gif

        statusmap\_image         redhat.gd2

        address                 114.215.204.97

}

define command{

    command\_name check\_dummy

    command\_line $USER1$/check\_dummy $ARG1$ $ARG2$

}

define contactgroup{

        contactgroup\_name       test\_biz\_group

        alias                   test\_biz\_group

        members                 gtpi

}

define service{

    use                     generic-service   ; template to inherit from

    name                    passive-service   ; name of this template

    active\_checks\_enabled   0                 ; no active checks

    passive\_checks\_enabled  1                 ; allow passive checks

    check\_command           check\_dummy!0     ; use "check\_dummy", RC=0 (OK)

    check\_period            24x7              ; check active all the time

    check\_freshness         0                 ; don't check if check result is "stale"

    register                0                 ; this is a template, not a real service

}

define service{

name  jmx-error-count-notify-service          ; The name of this service template

use  passive-service          ; Inherit default values from the generic-service definition

notification\_options

u,c,w    ; Send notifications about warning, unknown, critical, and recovery events

  notification\_interval           60 ; Re-notify about service problems every hour

  notification\_period             24x7  ; Notifications can be sent out at any time

contact\_groups                  test\_biz\_group

;       Notifications get sent out to everyone in the 'admins' group

  register                        0       ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!

}

# jmxtrans梳理 - u012333307的专栏 - 博客频道 - CSDN.NET

http://blog.csdn.net/u012333307/article/details/42276557

1 jmxtrans入口类

    com.googlecode.jmxtrans.JmxTransformer

2 启动命令行参数指定了json规则文件的目录(-j)或者文件(-f)

3 jmxtrans可以监控json文件或者目录的变化

4 定时规则文件

   4.1 cronExpression 可以指定server元素的执行频率，格式为cron格式，如果没有指定该定时参数，则使用系统统一配置的默认参数如每隔60s执行一次

   4.2 如果查询所有的属性，则在query子元素中不定义attr属性

    4.3 typeNames 对查询结果的过滤, 如

        HeapMemoryUsage 有子属性committed,init,max,used,如果设定了typeNames为["committed","init"]则只展示committed,init子属性的值，其它子属性max,used不展示

5 jmxtrans和被监控机JVM之间采用长连接对象池

6 每一个server中可以指定query查询启用的线程数目，如果不指定则默认采用线程，依次查询

7 nagios对接方式

    7.1 写在jmxtrans本地文件中，由nagios进行收集

8 graphite对接方式

   8.1 和graphite建立长连接，并且使用了连接池

# 如何部署Icinga服务端 - iVictor - 博客园

http://www.cnblogs.com/ivictor/p/5124384.html

Icinga是Nagios的一个变种，配置，使用方式几乎一样，而且完全兼容Nagios的插件。所以下面的部署方案对Nagios同样使用。

它还推出了两个中文版本，icinga-cn原版和icinga-pnp4nagios-cn，前者和Nagios几乎一模一样，只不过界面是中文的，而后者则集成了php4绘图功能，能以图形化的方式呈现系统的监控信息，类似于Cacti。

Icinga服务端一般是指其内核，它提供的只是一个框架，并不能监控具体的资源，譬如CPU，内存，进程等。对这些的监控是通过Icinga插件来实现的。

对远程Linux主机的监控一般有两种方式：

1. check\_by\_ssh插件

譬如我要查看远程Linux主机的磁盘空间的使用情况，

# /usr/local/icinga/libexec/check\_by\_ssh -H 192.168.244.134 -C 'df -h'

root@192.168.244.134's password:

Filesystem Size Used Avail Use% Mounted on

/dev/sda2 7.7G 5.8G 1.6G 79% /

tmpfs 850M 0 850M 0% /dev/shm

/dev/sda1 194M 27M 158M 15% /boot

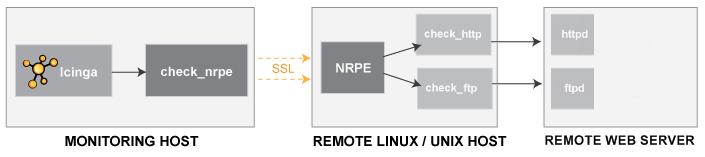
/dev/sda4 9.9G 7.2G 2.3G 77% /u01

该插件能够实现安全传输，使用SSH将会比NRPE插件更安全，而且，通过这种方式，远程被监控主机上不需要部署任何软件，但是这会导致远程主机和监控主机上的CPU负载过高。如果监控的主机比较多，这就会成为一个问题，因此，许多运维管理员选择NRPE插件，这样会使CPU负载降低。

2. NRPE插件

NRPE插件的原理是允许Icinga在远程主机上执行Nagios插件，这样就可监控远程主机上的本地资源，譬如CPU，内存，SWAP等不会暴露给外部机器的系统资源。

原理如下：



所以本方案实现的是Icinga内核+Nagios插件+NRPE插件。同时，本方案中使用了IDOUtils，这样，可将icinga的配置信息和监控数据等保存到数据库中。

**一、安装依赖包**

主要需安装以下几类包

Apache

GCC compiler

C/C++ development libraries

GD development libraries

libdbi/libdbi-drivers, database like MySQL or PostgreSQL

在**Fedora/RHEL/CentOS**系统中，具体如下：

# yum install httpd gcc glibc glibc-common gd gd-devel

# yum install libjpeg libjpeg-devel libpng libpng-devel

安装MySQL及其开发包

# yum install mysql mysql-server libdbi libdbi-devel libdbi-drivers libdbi-dbd-mysql

**二、创建账户**

# /usr/sbin/useradd -m icinga

# passwd icinga

如果是要从WEB界面发送命令给Icinga，还需要多配置一个组，并将web用户和icinga用户加入到该组中。

# /usr/sbin/groupadd icinga-cmd

# /usr/sbin/usermod -a -G icinga-cmd icinga

# /usr/sbin/usermod -a -G icinga-cmd apache

**三、下载Icinga及其插件包**

Icinga中文化项目的下载地址为：<http://sourceforge.net/projects/icinga-cn/files/> ，在这里，下载icinga-cn目录下的icinga-cn-1.12.2.tar.xz。

Icinga plugins的下载地址为：<http://sourceforge.net/projects/icinga-cn/files/icinga%20plugins/>，在这里，下载nagios-cn-plugins-2.0.3.tar.xz。

icinga nrpe的下载地址为：<http://sourceforge.net/projects/icinga-cn/files/icinga%20plugins/>，在这里，下载icinga-nrpe-2.14.tar.gz。

**四、安装Icinga内核**

即icinga-cn-1.12.2.tar.xz。

# cd /usr/src/

# tar xvf /root/icinga-cn-1.12.2.tar.xz

# cd icinga-cn-1.12.2/

编译

# ./configure --with-command-group=icinga-cmd --enable-idoutils

编译没有问题，则输出如下：

[复制代码](javascript:void(0);)

Web Interface Options:

------------------------

HTML URL: http://localhost/icinga/

CGI URL: http://localhost/icinga/cgi-bin/

Main URL: http://localhost/icinga/cgi-bin/status.cgi?allunhandledproblems

UI THEME: ui\_theme=ui-smoothness

Review the options above for accuracy. If they look okay,

type 'make all' to compile the main program and CGIs.

!!! Please take care about the upgrade documentation !!!

[复制代码](javascript:void(0);)

# make all

# make fullinstall

# make install-config

**五、创建MySQL数据及IDOUtils**

# mysql -u root -p

[复制代码](javascript:void(0);)

mysql> CREATE DATABASE icinga;

Query OK, **1** row affected (**0.00** sec)

mysql> GRANT SELECT, INSERT, UPDATE, DELETE, DROP, CREATE VIEW, INDEX, EXECUTE ON icinga.\* TO 'icinga'@'localhost' IDENTIFIED BY 'icinga';

Query OK, **0** rows affected (**0.00** sec)

mysql> quit

[复制代码](javascript:void(0);)

# cd /usr/src/icinga-cn-1.12.2/module/idoutils/db/mysql/

# mysql -u root -p icinga < mysql.sql

修改IDOUtils的配置文件

# vim /usr/local/icinga/etc/ido2db.cfg

db\_servertype=mysql

db\_port=**3306**

db\_user=icinga

db\_pass=icinga

其实，默认就是这样。

**六、配置经典的WEB界面**

# cd /usr/src/icinga-cn-1.12.2/

# make cgis

# make install-cgis

# make install-html

# make install-webconf

设置Icinga WEB界面的登录用户和密码

# htpasswd -c /usr/local/icinga/etc/htpasswd.users icingaadmin

如果要修改密码，可通过以下命令

# htpasswd /usr/local/icinga/etc/htpasswd.users icingaadmin

重启Apache服务，使上述设置生效

# service httpd restart

**七、编译和安装Icinga插件**

# cd /usr/src/

# tar xvf /root/nagios-cn-plugins-2.0.3.tar.xz

# cd nagios-cn-plugins-2.0.3/

# ./configure --prefix=/usr/local/icinga --with-cgiurl=/icinga/cgi-bin --with-nagios-user=icinga --with-nagios-group=icinga

# make

# make install

**八、编译和安装NRPE插件**

# cd /usr/src/

# tar xvf /root/icinga-nrpe-2.14.tar.gz

# cd icinga-nrpe-2.14/

# ./configure

# make

# make install

# make install-plugin

# make install-init

# make install-xinetd

# make install-daemon-config

其实make install-plugin，make install-init，make install-xinetd， make install-daemon-config也可以不执行，具体作用执行完make后有说明，建议都执行下。

**九、调整SELinux策略**

最简单的是直接关闭

临时关闭：# setenforce 0

永久关闭：# vim /etc/sysconfig/selinux

SELINUX=disabled

**十、开启IDOUtils和Icinga服务**

启动IDOUtils服务

# service ido2db start

验证Icinga的配置文件

# /usr/local/icinga/bin/icinga -v /usr/local/icinga/etc/icinga.cfg

输出如下：

[复制代码](javascript:void(0);)

Icinga 1.12.2

Copyright (c) 2009-2015 Icinga Development Team (http://www.icinga.org)

Copyright (c) 2009-2013 Nagios Core Development Team and Community Contributors

Copyright (c) 2009-2014 icinga-cn中文化组

Copyright (c) 1999-2009 Ethan Galstad

Last Modified: 02-14-2015

License: GPL

读取配置数据...

警报: 未知 'event\_profiling\_enabled' 配置设置. 将其从配置中移除!

Read main config file okay...

Processing object config directory '/usr/local/icinga/etc/conf.d'...

Processing object config file '/usr/local/icinga/etc/objects/commands.cfg'...

Processing object config file '/usr/local/icinga/etc/objects/contacts.cfg'...

Processing object config file '/usr/local/icinga/etc/objects/notifications.cfg'...

Processing object config file '/usr/local/icinga/etc/objects/timeperiods.cfg'...

Processing object config file '/usr/local/icinga/etc/objects/templates.cfg'...

Processing object config file '/usr/local/icinga/etc/objects/localhost.cfg'...

Processing object config file '/usr/local/icinga/etc/objects/linux.cfg'...

Processing object config directory '/usr/local/icinga/etc/modules'...

Processing object config file '/usr/local/icinga/etc/modules/idoutils.cfg'...

Read object config files okay...

Running pre-flight check on configuration data...

Checking services...

已检查17服务.

检查主机...

已检查2主机.

检查主机组...

已检查2主机组.

检查服务组...

已检查2服务组.

检查联系人...

已检查1联系人.

检查联系人组...

已检查1联系人组.

检查服务升级...

已检查0服务升级.

检查服务依赖关系...

已检查0服务依赖关系.

检查主机升级...

已检查0主机升级.

检查主机依赖关系...

已检查0主机依赖关系.

检查命令...

已检查36命令.

检查时间段...

已检查6时间段.

检查模块...

已检查1模块.

检查主机之间的回路...

检查回路主机和服务的依赖性...

检查全局事件处理...

检查强迫性处理命令...

检查杂项设置...

总计警报s: 0

总计错误: 0

Things look okay - No serious problems were detected during the pre-flight check

[复制代码](javascript:void(0);)

启动Icinga服务

# service icinga start

设置开机自启动

# chkconfig ido2db on

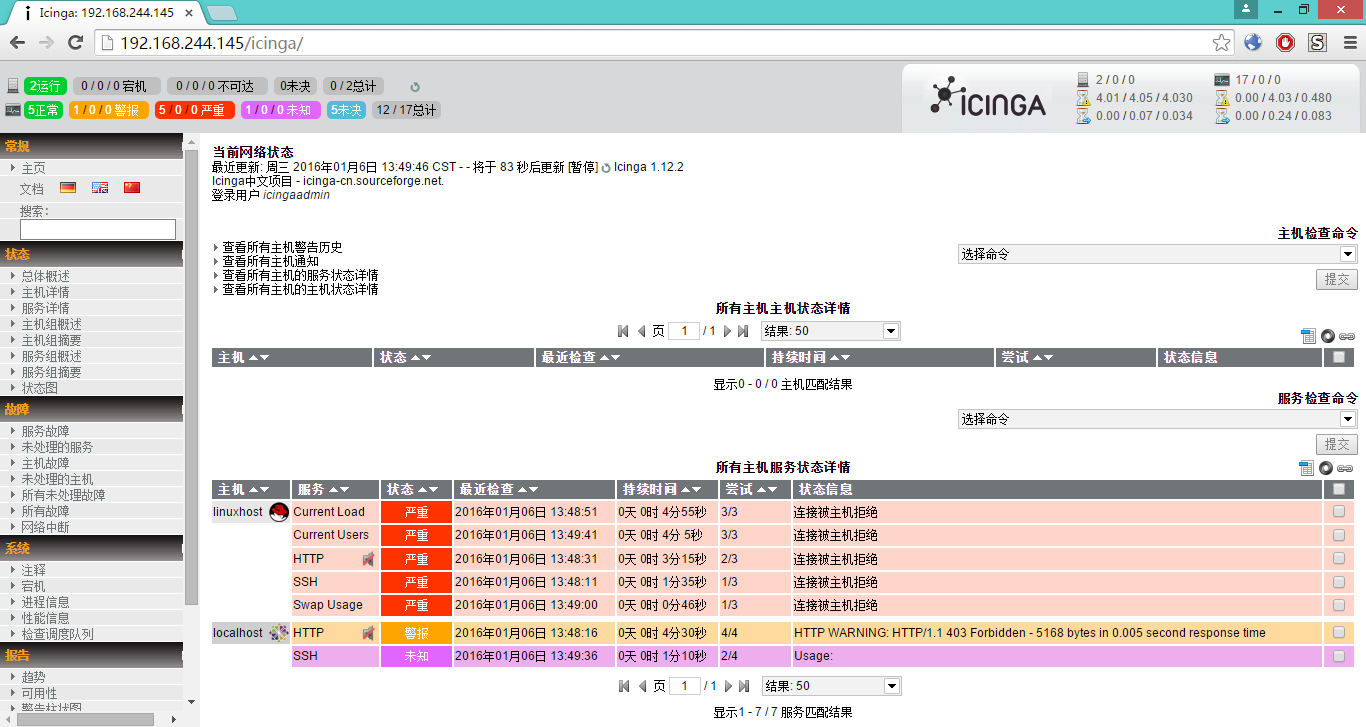
# chkconfig icinga on

**十一、登录WEB界面进行测试**

登录地址：http://192.168.244.145/icinga/

登录用户名为：icingaadmin

登录密码为第六步通过htpasswd命令设置的密码。



**总结：**

1. 在上述方案中，IDOUtils和NRPE并不是必需的，如果只需搭建一个简单的Icinga服务端，只需要Icinga内核和Nagios插件。具体可参考：

<http://docs.icinga.org/latest/en/quickstart-icinga.html>

2. 官方的部署文档在MySQL中创建icinga数据库时，没有指定字符集，而默认的字符集为latin1，这会导致中文的输出结果为乱码，所以，需显性执行数据库的默认字符集。

**参考：**

1. <http://docs.icinga.org/latest/en/quickstart-idoutils.html>

2. 《掌控-构建Linux系统Nagios监控服务器》

# ganglia3.6.1+jmxtrans+strom-0.9.4集成 - 滴水石穿 - 51CTO技术博客

http://chengyanbin.blog.51cto.com/3900113/1653572

1、安装ganglia参考之前的一篇博客（以下示例使用ganglia组播方式，单播方式大家自己尝试）

<http://chengyanbin.blog.51cto.com/3900113/1591373>

2、安装jmxtrans

<http://chengyanbin.blog.51cto.com/3900113/1654754>

3、安装storm

<http://chengyanbin.blog.51cto.com/3900113/1654757>

4、修改storm配置文件storm..yaml

|  |  |
| --- | --- |
| 1  2  3  4  5 | ###nimbus config   nimbus.childopts: "-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.port=12345 -Xmx1024m"    ###supervisor config   supervisor.childopts: "-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.port=12346 -Xmx256m" |

5、为jmxtrans增加两个json文件

分别获取nimbus和supervisor节点的jvm信息，以下文件仅列举了部分参数，如果有需要可以增加queries中的数据，以监控更多的参数，具体的MBean的参数，可以通过jconsole来查看

nimbus.json

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | {      "servers": [          {                  "host": "master",              "port": "12345",              "queries": [                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "nimbus",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:type=Memory",                      "resultAlias": "nimbus.heap",                      "attr": [                          "ObjectPendingFinalizationCount"                      ]                  },                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "nimbus",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:name=Copy,type=GarbageCollector",                      "resultAlias": "nimbus.gc",                      "attr": [                          "CollectionCount",                          "CollectionTime"                      ]                  },                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "nimbus",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:name=Code Cache,type=MemoryPool",                      "resultAlias": "nimbus.threads",                      "attr": [                          "CollectionUsageThreshold",                          "CollectionUsageThresholdCount",                          "UsageThreshold",                          "UsageThresholdCount"                      ]                  },                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "nimbus",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:type=Runtime",                      "resultAlias": "nimbus.runtime",                      "attr": [                          "StartTime",                          "Uptime"                      ]                  }              ],              "numQueryThreads": 2          }      ]  } |

supervisor.json

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | {      "servers": [          {                  "host": "node1",              "port": "12346",              "queries": [                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "supervisor",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:type=Memory",                      "resultAlias": "supervisor.heap",                      "attr": [                          "ObjectPendingFinalizationCount"                      ]                  },                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "supervisor",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:name=Copy,type=GarbageCollector",                      "resultAlias": "supervisor.gc",                      "attr": [                          "CollectionCount",                          "CollectionTime"                      ]                  },                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "supervisor",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:name=Code Cache,type=MemoryPool",                      "resultAlias": "supervisor.threads",                      "attr": [                          "CollectionUsageThreshold",                          "CollectionUsageThresholdCount",                          "UsageThreshold",                          "UsageThresholdCount"                      ]                  },                  {                      "outputWriters": [                          {                              "@class": "com.googlecode.jmxtrans.model.output.GangliaWriter",                              "settings": {                                  "groupName": "supervisor",                                  "host": "239.2.11.71",                                  "port": "8649"                              }                          }                      ],                      "obj": "java.lang:type=Runtime",                      "resultAlias": "supervisor.runtime",                      "attr": [                          "StartTime",                          "Uptime"                      ]                  }              ],              "numQueryThreads": 2          }      ]  } |

两个json文件一定要注意host是主机名，如果写ip的话，在ganglia里同一个节点会有两个显示，看着有点别扭，大家都统一使用hostname就好.ganglia的gmnod和gmetad不需要重启，用自动收集。

说明：

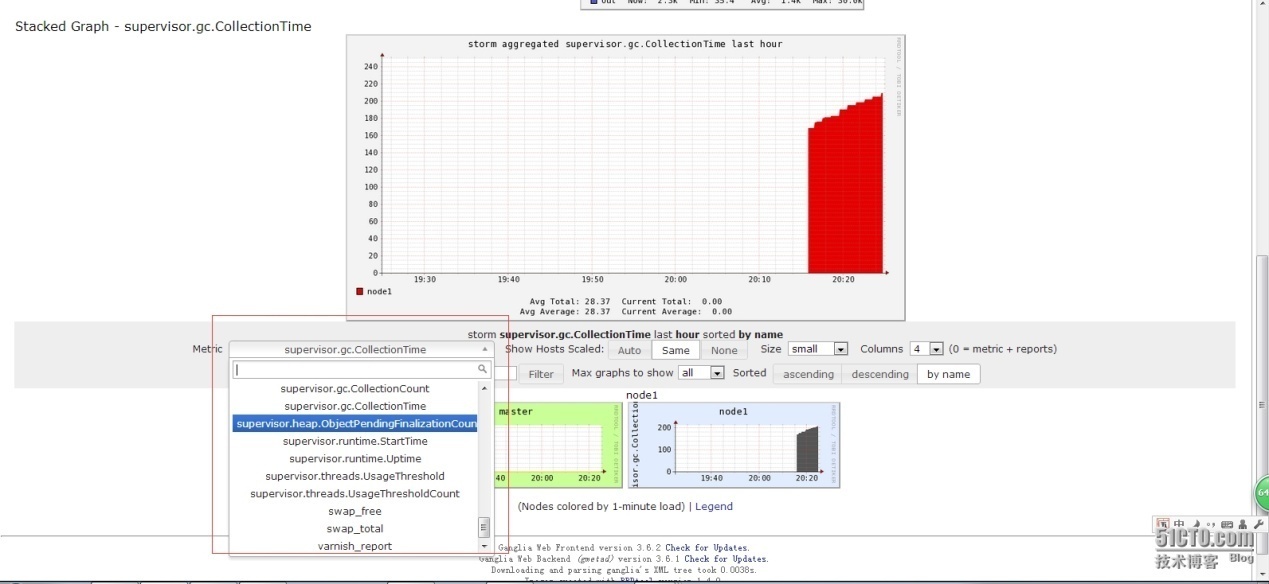
    修改json文件需要重新启动jmxtrans,查看jmxtrans日志/var/log/jmxtrans/jmxtrans.log.

    service jmxtrans restart

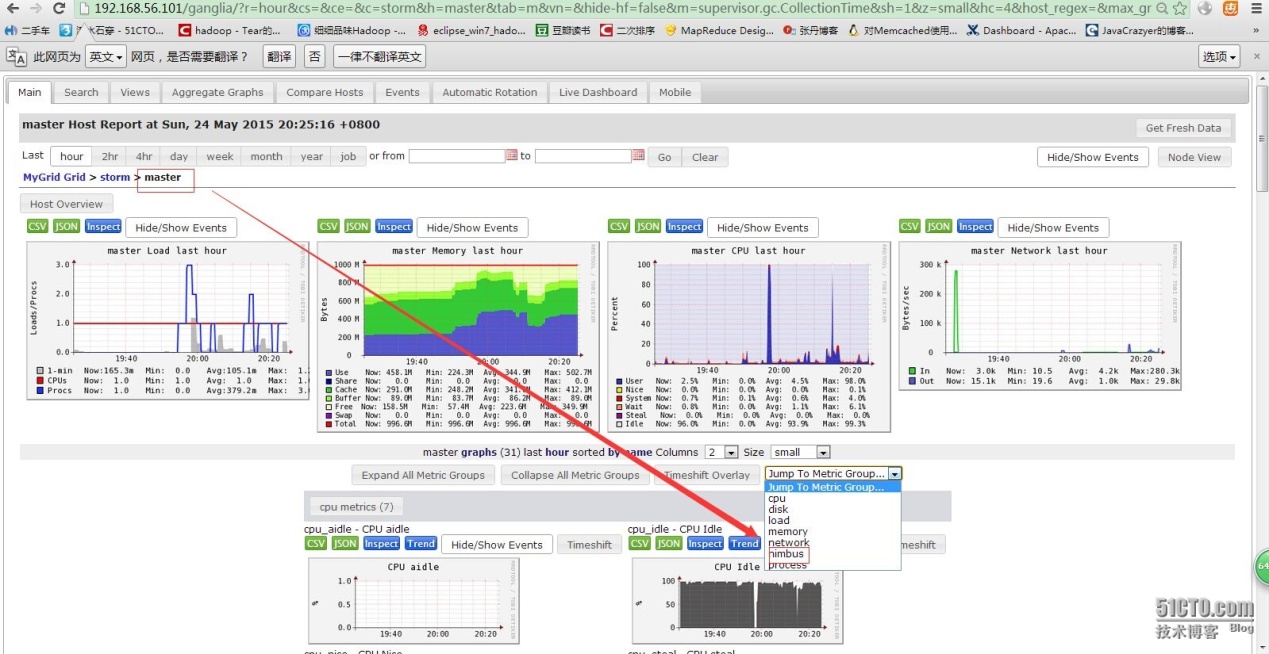
下面来张ganglia的主界面，监控两个节点

[](http://s3.51cto.com/wyfs02/M01/6D/59/wKiom1Vhw3nCBgzWAAlYkzA0ce8396.jpg)

上图显示不出来监控指标，就放下面来了，可以看到supervisor的监控数据已经收集到ganglia里了。

[](http://s3.51cto.com/wyfs02/M02/6D/59/wKiom1Vhw3mi12hpAAZfzJ9W5tU476.jpg)

下图搞了master节点的页面，可以看到nimbus group的监控数据已经显示，node1节点的supervisor group懒得上图了，大家可以明白了，不明白的去面壁去[j_0059.gif](http://img.baidu.com/hi/jx2/j_0059.gif)

[](http://s3.51cto.com/wyfs02/M02/6D/55/wKioL1VhxQaiKlK2AAiSbnnLzQY490.jpg)

本文出自 “[滴水石穿](http://chengyanbin.blog.51cto.com/)” 博客，请务必保留此出处<http://chengyanbin.blog.51cto.com/3900113/1653572>

# Hawkular - Monitoring JVM applications with jmxtrans

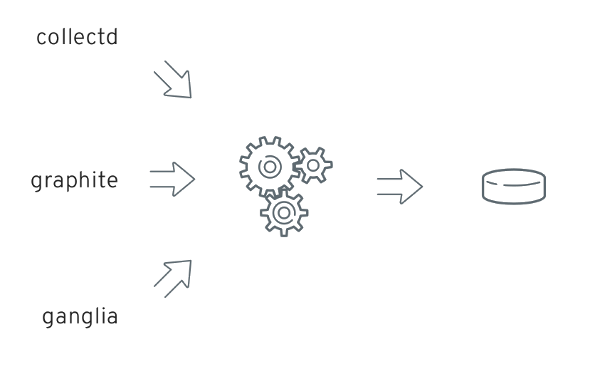
http://www.hawkular.org/blog/2016/04/19/jmxtrans-to-hawkular-metrics.html

In this post, you’ll learn how to monitor applications running on the Java Virtual Machine with jmxtrans and Hakwular Metrics.

[Hawkular Metrics](https://github.com/hawkular/hawkular-metrics) is an easy to install, scalable metric storage component.

[jmxtrans](http://www.jmxtrans.org/) is a popular monitoring tool for Java based applications. It connects to a JVM via JMX, collects metrics, and sends the data to the backend of your choice. Very often a Graphite backend is used. As an example, we will monitor a Tomcat 8 server.

Hawkular Metrics can store data sent over the Graphite text protocol, thanks to [ptrans](https://github.com/hawkular/hawkular-metrics/tree/master/clients/ptranslator). ptrans is a proxy server taking in metrics data in several common formats and emitting them as REST requests into Hawkular Metrics backend for storage. The figure below illustrates the process.

[](http://www.hawkular.org/img/blog/2016/ptrans.png)

**Figure 1. ptrans (Protocol Translator)**

jmxtrans will communicate with ptrans over the Graphite text protocol.

**THE SETUP IN DETAILS**

**Hawkular Metrics and PTrans**

[Hawkular Metrics standalone installation](http://www.hawkular.org/docs/components/metrics/installation.html) is documented on the Hawkular website. ptrans setup is documented on the same page.

**Tomcat 8**

JMX remote access must be enabled otherwise jmxtrans can’t connect. To do this, create (or modify) the setenv.sh script in the bin directory.

**setenv.sh**

#!/bin/bash

CATALINA\_OPTS="-Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.port=35135"

**jmxtrans**

Download the latest [jmxtrans distribution](https://github.com/jmxtrans/jmxtrans/releases). Expand it somewhere on your disk.

In the bin directory, create a jmxtrans.conf file. It will be sourced by the jmxtrans.sh script.

**jmxtrans.conf**

#!/bin/bash

JAR\_FILE="/path/to/jmx-trans/lib/jmxtrans-all.jar"

LOG\_DIR="/path/to/jmx-trans/bin"

LOG\_FILE="/path/to/jmx-trans/bin/jmxtrans.out"

SECONDS\_BETWEEN\_RUNS=5

|  |  |
| --- | --- |
|  | Where the script will find the jmxtrans fat JAR |
|  | The directory where log files should be written |
|  | The file where the process output will be redirected |
|  | Instructs jmxtrans to collect metrics and send data every 5 seconds |

Then create the main configuration file, config.json. The [jmxtrans wiki](https://github.com/jmxtrans/jmxtrans/wiki) has a detailed reference.

**config.json**

{

"servers": [

{

"numQueryThreads": "2",

"host": "localhost",

"port": "35135",

"queries": [

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"obj": "java.lang:type=OperatingSystem",

"attr": [

"SystemLoadAverage",

"AvailableProcessors",

"TotalPhysicalMemorySize",

"FreePhysicalMemorySize",

"TotalSwapSpaceSize",

"FreeSwapSpaceSize",

"OpenFileDescriptorCount",

"MaxFileDescriptorCount"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"resultAlias": "heap",

"obj": "java.lang:type=Memory",

"attr": [

"HeapMemoryUsage",

"NonHeapMemoryUsage"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"resultAlias": "cmsoldgen",

"obj": "java.lang:name=CMS Old Gen,type=MemoryPool",

"attr": [

"Usage"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"resultAlias": "gc",

"obj": "java.lang:type=GarbageCollector,name=\*",

"attr": [

"CollectionCount",

"CollectionTime"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter",

"settings": {}

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"resultAlias": "memorypool",

"obj": "java.lang:type=MemoryPool,name=\*",

"attr": [

"Usage"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"resultAlias": "threads",

"obj": "java.lang:type=Threading",

"attr": [

"DaemonThreadCount",

"PeakThreadCount",

"ThreadCount",

"TotalStartedThreadCount"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"resultAlias": "tomcat8-connectors",

"obj": "Catalina:type=ThreadPool,name=\*",

"attr": [

"currentThreadCount",

"currentThreadsBusy"

]

},

{

"outputWriters": [

{

"@class": "com.googlecode.jmxtrans.model.output.StdOutWriter"

},

{

"@class": "com.googlecode.jmxtrans.model.output.GraphiteWriter",

"settings": {

"host": "127.0.0.1",

"port": "2003"

}

}

],

"resultAlias": "tomcat8-requests",

"obj": "Catalina:type=GlobalRequestProcessor,name=\*",

"attr": [

"bytesReceived",

"bytesSent",

"errorCount",

"processingTime",

"requestCount"

]

}

]

}

]

}

|  |  |
| --- | --- |
|  | name of the host where Tomcat is running |
|  | jmx remoting port (must match the value set in setenv.sh in Tomcat installation) |
|  | queries item lists the MBeans which should be invoked |
|  | output writers item indicates where data collected should be sent |
|  | StdOutWriter simply prints collected data to the process output stream (useful for configuration debugging) |
|  | GraphiteWriter sends data to a remote server which understands the Graphite protocols; by default, the text protocol is used |
|  | graphite remote host (must match the value set in ptrans.conf) |
|  | graphite remote port (must match the value set in ptrans.conf) |
|  | name of the MBean to invoke |
|  | list of attributes to collect |

Eventually, start jmxtrans.

**Starting jmxtrans**

./jmxtrans.sh start config.json

That’s it!

# StreamSets Monitoring with Grafana, InfluxDB, and jmxtrans

https://streamsets.com/blog/streamsets-monitoring-grafana-influxdb-jmxtrans/

The ability to monitor your critical infrastructure is a must, and we designed the StreamSets Data Collector (SDC) with this in mind: metrics are exposed through both the REST API and JMX. While there are many approaches to monitoring these metrics, let’s walk through a specific end-to-end example using jmxtrans to collect metrics, InfluxDB to store them, and Grafana to visualize them.

We’ll use Docker to make things easy to set up in this demo environment, but you should follow each tool’s production-ready installation guides for a real deployment.

**StreamSets Data Collector**

The one prerequisite configuration for the Data Collector is that we enable JMX metrics and choose a port to listen on. We'll use port **1105**. This can be set by modifying the environment variable SDC\_JAVA\_OPTS and restarting the Data Collector. Let’s start an SDC instance with the following command.

docker run -d --name datacollector --expose 1105 -p 18630:18630 -e SDC\_JAVA\_OPTS="-Dcom.sun.management.jmxremote.port=1105 -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false" streamsets/datacollector:1.1.4

**InfluxDB Setup**

We've found a pretty good docker image for [InfluxDB](https://hub.docker.com/r/tutum/influxdb/) from the folks over at [Tutum](https://www.tutum.co/). In order to make sure that our data persists in the case we want to make changes to our InfluxDB container, we'll first create a separate data-only container.

docker create --name influx-data -v /data tutum/influxdb:0.9

docker create --name influx-data -v /data/jmx tutum/influxdb

Next, we'll start up our InfluxDB instance.

docker run -d \

--volumes-from influx-data \

-p 8083:8083 -p 8086:8086 --expose 2003 --expose 8084 \

-e PRE\_CREATE=grafana \

-e GRAPHITE\_DB="grafana" \

-e GRAPHITE\_BINDING=':2003' \

-e GRAPHITE\_PROTOCOL="tcp" \

--name influxdb \

tutum/influxdb:0.9

docker run -d \

--volumes-from influx-data \

-p 8083:8083 -p 8086:8086 --expose 2003 --expose 8084 \

-e PRE\_CREATE=grafana \

-e GRAPHITE\_DB="grafana" \

-e GRAPHITE\_BINDING=':2003' \

-e GRAPHITE\_PROTOCOL="tcp" \

--name influxdb \

tutum/influxdb

In the above command, we're launching the InfluxDB container with the user-facing ports statically mapped to the host to keep things easy since this is a single-node installation. The InfluxDB web interface runs on 8083, 8086 has the REST API that Grafana will use for queries, and 2003 is the port for the Graphite protocol plugin for InfluxDB. This is important because jmxtrans (and the yaml2json converter) currently supports Graphite, but not the native InfluxDB interface.

The next set of options simply pre-create a database called ‘grafana' and set up the Graphite protocol plugin.

**Grafana**

For Grafana, we'll use the [official docker image](https://hub.docker.com/r/grafana/grafana/) from the Grafana team. Starting this up is really simple; the only extra option we specify is a link to the InfluxDB container and an optional static port mapping.

docker run -d --link influxdb:influxdb -p 3000:3000 grafana/grafana:latest

docker run -d --link influxdb:influxdb -p 3000:3000 grafana/grafana

**jmxtrans**

For running the JMX collector, we’ll use a pre-built docker image similar to InfluxDB and Grafana.

Rather than writing JSON configuration files manually, let's use the YAML configuration option instead. The YAML configuration below is usable as-is, but for those curious, the full YAML documentation is available [here](https://github.com/jmxtrans/jmxtrans/wiki/YAMLConfig).

You'll need the **yaml2jmxtrans.py** script that is bundled with the jmxtrans distribution. You can download a zip or tarball from [jmxtrans.org](http://www.jmxtrans.org/) to obtain this script.

Download the example jmxtrans configuration file [here](https://gist.github.com/kunickiaj/a052c751a6efaa8571db). The ‘streamsets’ query will collect all of the StreamSets-specific mbeans exposed over JMX under the alias “streamsets”. The full example YAML configuration is shown below for context..

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | - name: streamsets    obj : "metrics:name=sdc.pipeline.\*"    resultAlias: "streamsets"    attr:      - "Count"      - "Max"      - "Mean"      - "Min"      - "StdDev"      - "Value"      - "50thPercentile"      - "75thPercentile"      - "95thPercentile"      - "98thPercentile"      - "99thPercentile"      - "999thPercentile"      - "FiveMinuteRate"      - "FifteenMinuteRate"      - "MeanRate"      - "OneMinuteRate" |

Once you've defined a YAML configuration it needs to be converted into the JSON file that jmxtrans expects using the yaml2jmxtrans.py script (Python 2.6+ required). If your configuration file is called ‘jmxtrans.yml' you can invoke it with:

|  |  |
| --- | --- |
| 1 | &lt;jmxtrans directory&gt;/tools/yaml2jmxtrans.py jmxtrans.yml |

You'll have a JSON file called StreamSetsBase.json output in the current directory. We’ll mount this to the jmxtrans container as a volume.

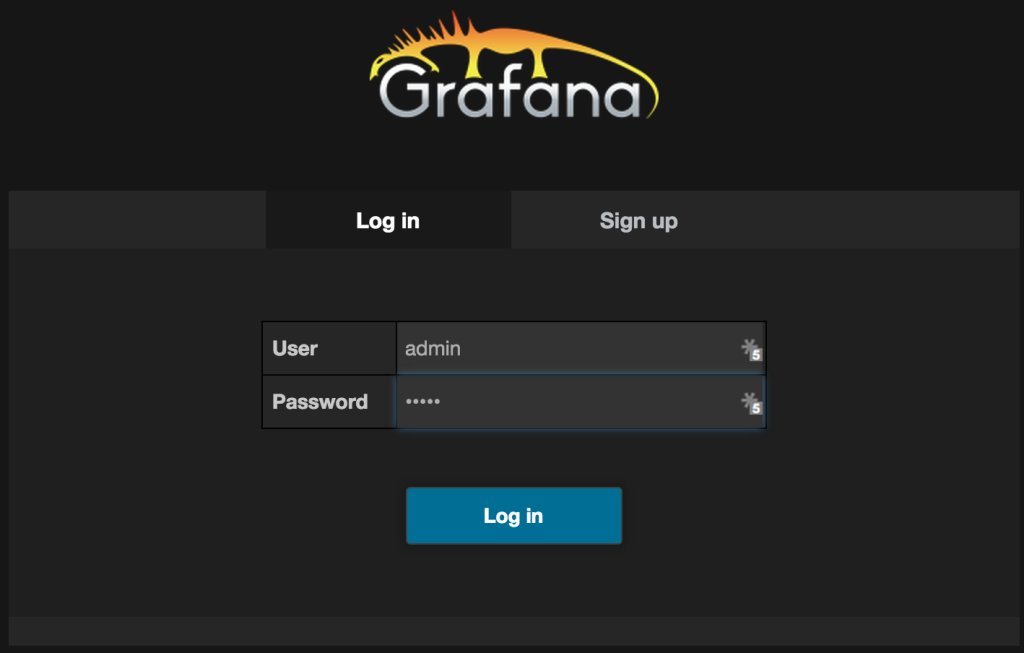
Start jmxtrans with

docker run -d --name jmxtrans --link datacollector --link influxdb -e JMXTRANS\_GRAPHITE\_HOST=influxdb -e JMXTRANS\_GRAPHITE\_PORT=2003 -e JMXTRANS\_JMX\_HOST=datacollector -e JMXTRANS\_JMXPORT=1105 -v $PWD/StreamSetsBase.json:/opt/jmxtrans/conf/StreamSetsBase.json kunickiaj/jmxtrans:latest

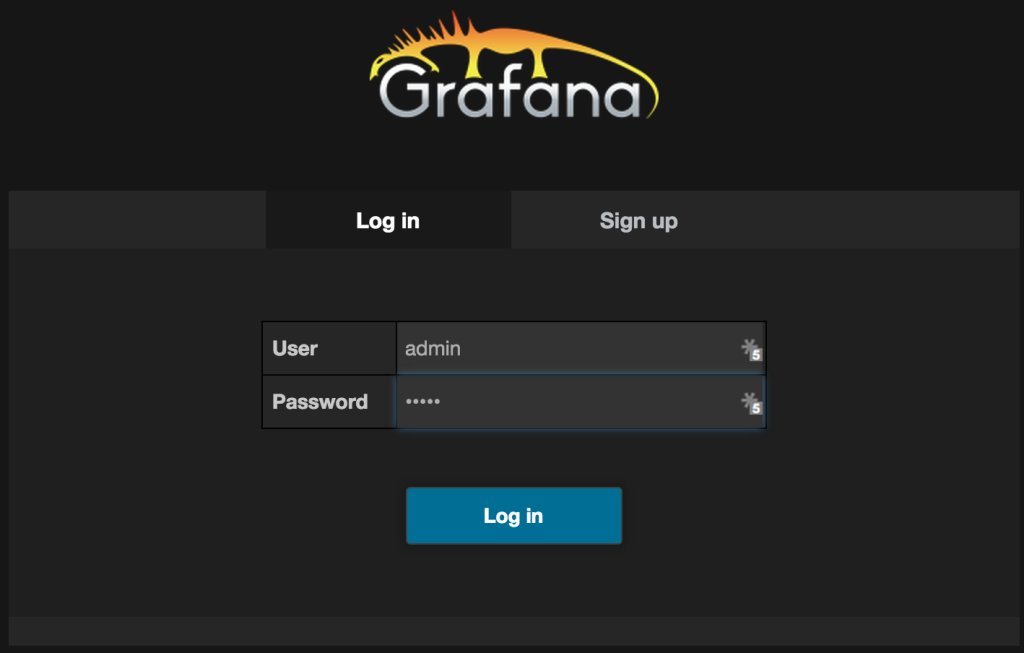
We should now have metrics flowing into InfluxDB and are ready to setup some dashboards!

**Dashboards**

You should now be able to browse over to http://<your grafana host>:3000 and login with the default username and password: admin / admin.

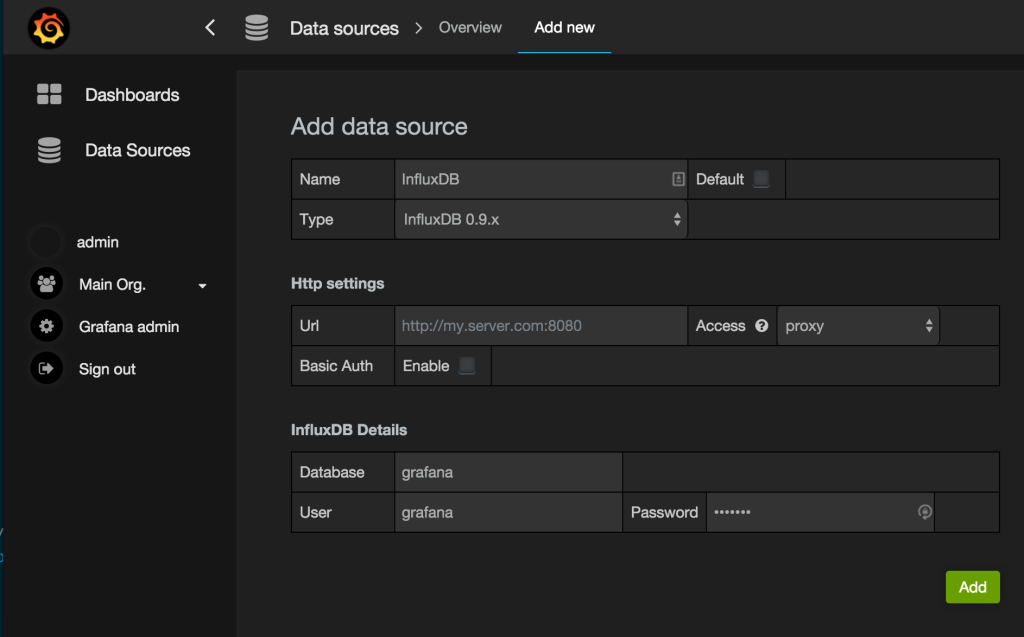
[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-15.05.57.png)

Login with the default: admin / admin xxx

[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-15.05.57.png)

Login with the default: admin / admin

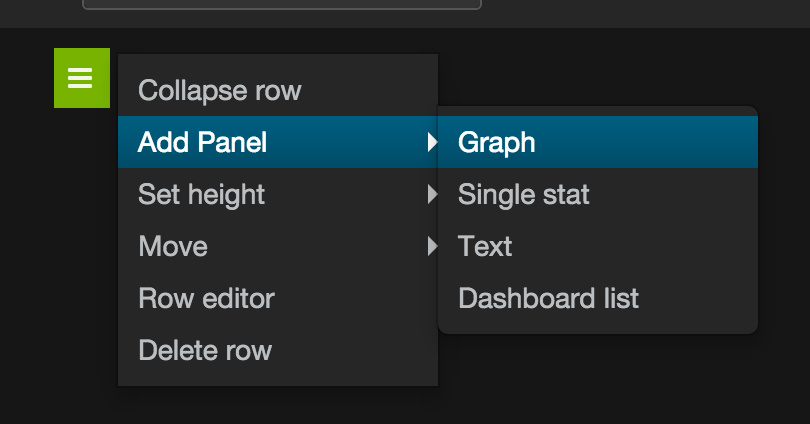
The first thing we need to do is tell Grafana about our InfluxDB data source.

[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-15.54.53.png)

In the URL use the docker host for InfluxDB and port 8086. For Database, Username, and Password use ‘grafana'

Next, we will create a new dashboard by clicking on ‘Home' in the Dashboard Selector and then New. This will create a new dashboard titled New Dashboard.

Now, click the green bar on the left hand side of the empty dashboard and choose Add Panel > Graph. This adds a new graph with some sample data. We'll want to click the title bar of the graph and choose Edit to specify our own metrics from InfluxDB.

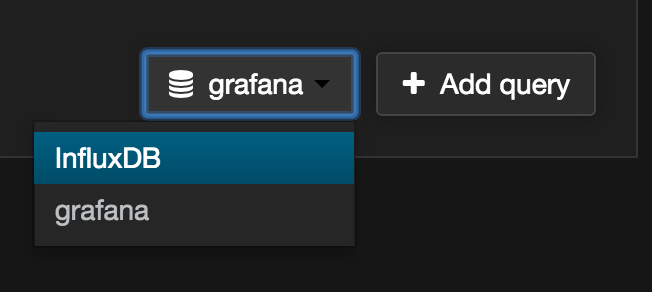
[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-16.00.14.png)

Adding a new graph to the dashboard.

[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-16.00.25.png)

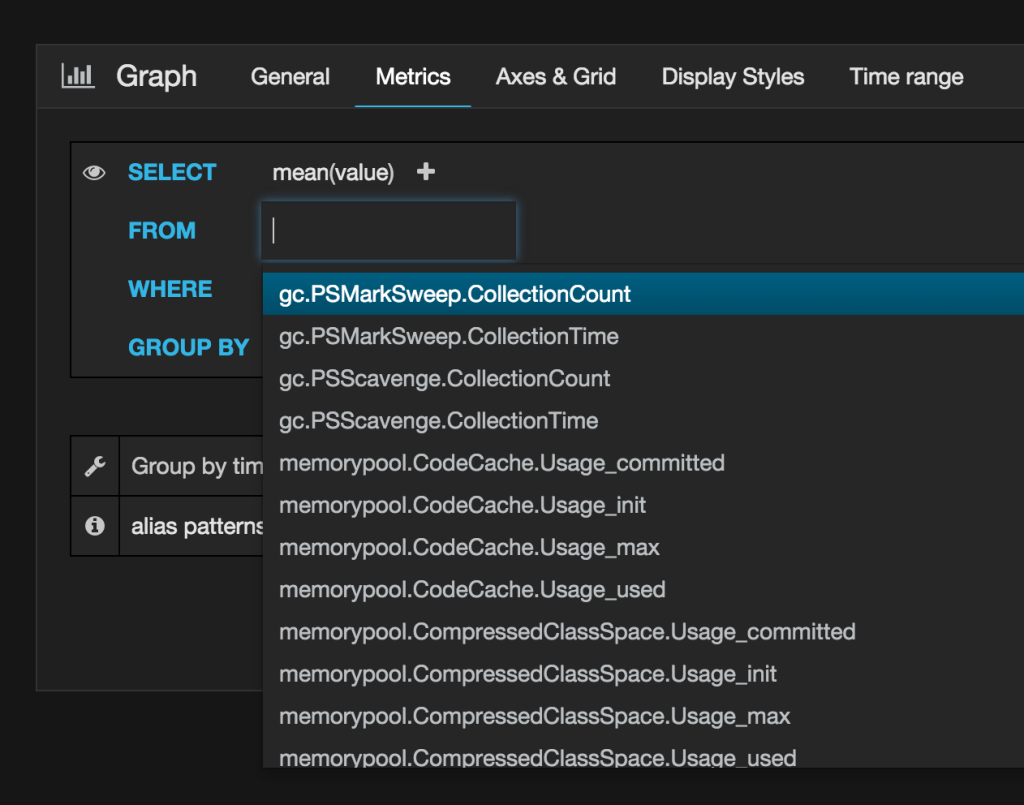
Editing a Graph

Now, change the data source from the default to ‘InfluxDB'

[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-16.00.37.png)

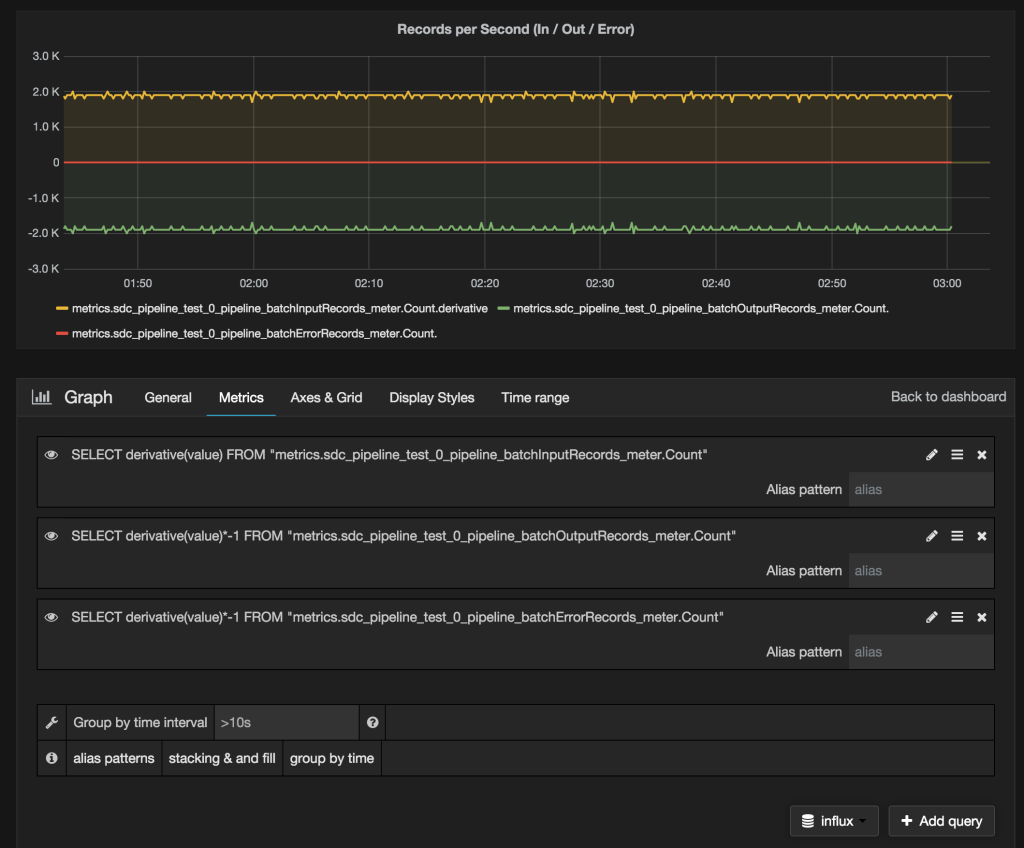
Choosing a Data Source

Configuring metrics. In the FROM box you can choose from an automatically populated list of metrics. Data Collector metrics will be prefixed with streamsets based on our jmxtrans configuration. Please note that until you've created a pipeline in StreamSets Data Collector and have started it, there won't yet be any data available for Grafana to populate the autocomplete list. Create and start a pipeline first in order to display some metrics.

[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-16.00.49.png)

Selecting Metrics

StreamSets provides pre-aggregated rates for many of the metrics, but you can also use the advanced query editor and functions like DERIVATIVE to create custom rate metrics that suit your needs.

[](https://19ttqs47cfw33zkecq3dz58m-wpengine.netdna-ssl.com/wp-content/uploads/2015/10/Screenshot-2015-10-05-16.34.35.png)

Sample of Records per second by Type

You can also import this [Complete JSON for Example Grafana Dashboard](https://gist.github.com/kunickiaj/53ebebf16ebba9876b92) to try out some charts already setup and ready to go.

Hope you enjoyed this quick start to monitoring using [Grafana](http://grafana.org/), [InfluxDB](https://influxdata.com/), [jmxtrans](http://jmxtrans.org/) and [StreamSets Data Collector](https://streamsets.com/).

Now that you’ve walked through step-by-step, you can download the YAML file below and use it with docker-compose or [Tutum](http://tutum.co/) to launch this setup automatically.

[jmxdemo.yml](https://gist.github.com/kunickiaj/42c9a24c704b5980182d)

Launch the entire stack with:

|  |  |
| --- | --- |
| 1 | docker-compose -f jmxdemo.yml up |

* http://www.rittmanmead.com/2015/02/obiee-monitoring-and-diagnostics-with-influxdb-and-grafana/
* https://github.com/tutumcloud/influxdb
* https://github.com/jmxtrans/jmxtrans/wiki/YAMLConfig

# 2@搭建例子

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans -P jmxtrans/jmxtrans

java -Djava.rmi.server.hostname=192.168.0.179 -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.port=10001 -jar openbridge-monitor.jar

**【注意】URL的端口是8086，而刚才配置的8083是UI的端口。**

    - 8083端口是InfluxDB的UI界面展示的端口

    - 8086端口是Grafana用来从数据库取数据的端口

    - 2003端口则是刚刚设置的，Jmeter往数据库发数据的端口

自动创建用户名和密码

"outputWriters": [{

"@class": "com.googlecode.jmxtrans.model.output.InfluxDbWriterFactory",

"url": "http://192.168.0.179:8086/",

"username": "admin",

"password": "admin",

"database": "jmxDB"

}

]

http://192.168.0.179:8083/

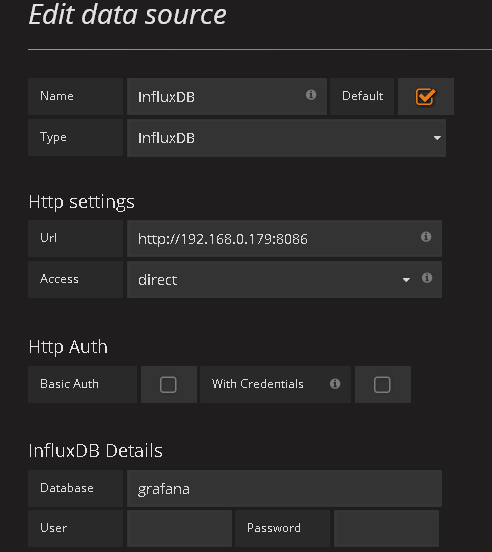
# 可以使用这个，这个是查询所有表，显示1条记录

select \* from /.\*/ limit 1

# 也可以使用这个，这个是显示所有表

show measurements

curl -XPOST 'http://localhost:8086/write?db=grafana' -d 'cpu,host=server01,region=uswest load=42 1434055562000000000'



# Collectd & InfluxDb & Grafana 之一: 常用系统统计 - Erlang/Elixir实践 - SegmentFault

https://segmentfault.com/a/1190000006868587



**Collectd**

安装

apt-**get** install collected

配置

# vi /etc/collectd/collectd.conf

**Hostname** "localhost"

**FQDNLookup** true

**Interval** 5

**Timeout** 4

**LoadPlugin** syslog

<Plugin syslog>

**LogLevel** info

</Plugin>

**LoadPlugin** battery

**LoadPlugin** cpu

**LoadPlugin** cpufreq

**LoadPlugin** df

**LoadPlugin** disk

**LoadPlugin** entropy

**LoadPlugin** interface

**LoadPlugin** irq

**LoadPlugin** load

**LoadPlugin** memory

**LoadPlugin** network

**LoadPlugin** processes

**LoadPlugin** rrdtool

**LoadPlugin** swap

**LoadPlugin** users

<Plugin df>

**FSType** rootfs

**FSType** sysfs

**FSType** proc

**FSType** devtmpfs

**FSType** devpts

**FSType** tmpfs

**FSType** fusectl

**FSType** cgroup

**IgnoreSelected** true

**ReportByDevice** true

**ReportInodes** true

**ValuesAbsolute** true

**ValuesPercentage** true

</Plugin>

<Plugin interface>

**Interface** "eno1"

**IgnoreSelected** false

</Plugin>

# 网络插件, 把Collectd搜集的数据通过接口eno1发往192.168.212.127:25826

<Plugin network>

<Server "192.168.212.127" "25826">

**Interface** "eno1"

</Server>

</Plugin>

<Plugin rrdtool>

**DataDir** "/var/lib/collectd/rrd"

</Plugin>

<Include "/etc/collectd/collectd.conf.d">

**Filter** "\*.conf"

</Include>

**InfluxDB**

安装

**wget** https://dl.influxdata.com/influxdb/releases/influxdb\_1.0.0\_amd64.deb

sudo dpkg -i influxdb\_1.0.0\_amd64.deb

启动

root@ubuntu:~# service influxdb status

influxdb process is **not** running [ FAILED ]

root@ubuntu:~# service influxdb start

Starting influxdb...

influxdb process was started [ OK ]

运行客户端influx创建数据库

➜ ~ influx

> CREATE DATABASE "collectdb"

编辑 /etc/influxdb/influxdb.conf, 找到 [[collectd]]部分, 修改如下

[[collectd]]

enabled = true

# 在 `192.168.212.127:25826` 上监听从 Collectd 发过来的数据.

bind-address = "192.168.212.127:25826"

database = "collectdb"

typesdb = "/usr/share/collectd/types.db"

batch-size = 5000

batch-pending = 10

batch-timeout = "10s"

read-buffer = 0

重启

**service** influxdb restart

**Grafana**

安装

# vi /etc/apt/source.list.d/grafana.list

**deb** https://packagecloud.io/grafana/stable/debian/ wheezy main

curl https://packagecloud.io/gpg.key | sudo apt-key add -

sudo apt-**get** update

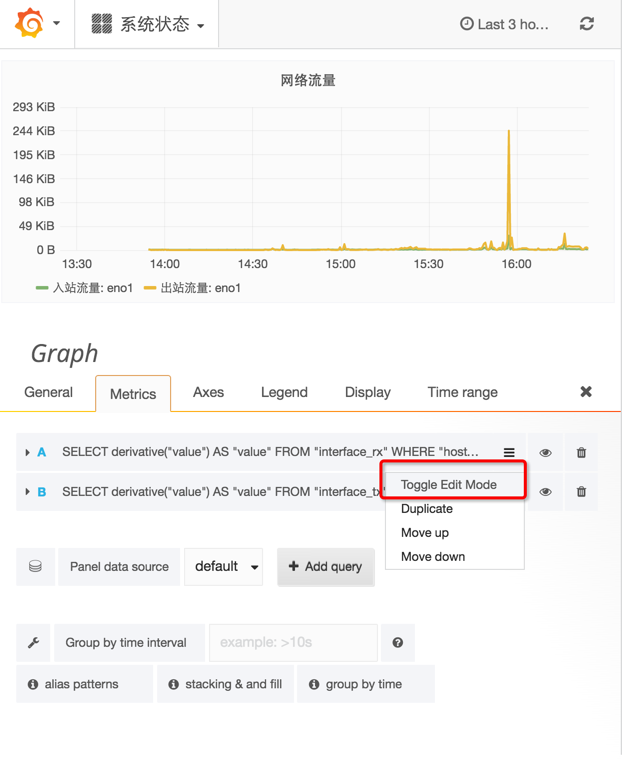
sudo apt-**get** install grafana

配置

[http://docs.grafana.org/insta...](http://docs.grafana.org/installation/debian/)

**网络流量统计**

切换编辑模式, 然后输入自定义SQL查询



输入查询语句

**SELECT** derivative("value") **AS** "value" **FROM** "interface\_rx" **WHERE** "host" = 'localhost' **AND** "type" = 'if\_octets' **AND**"instance" = 'eno1'

函数 derivative 意为导数, 微积分中的概念. value 为传输总量(字节), derivative("value") 为 value 在时间上的增量.

其中

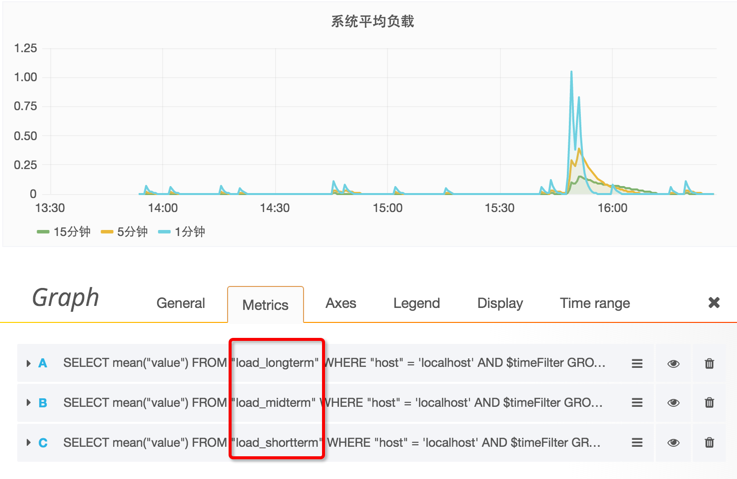
* host = localhost
* type = if\_octets
* instance = eno1

**系统负载**

**SELECT** mean("value") **FROM** "load\_longterm" **WHERE** "host" = 'localhost' **AND** $timeFilter **GROUP** **BY** **time**($interval) fill(null)

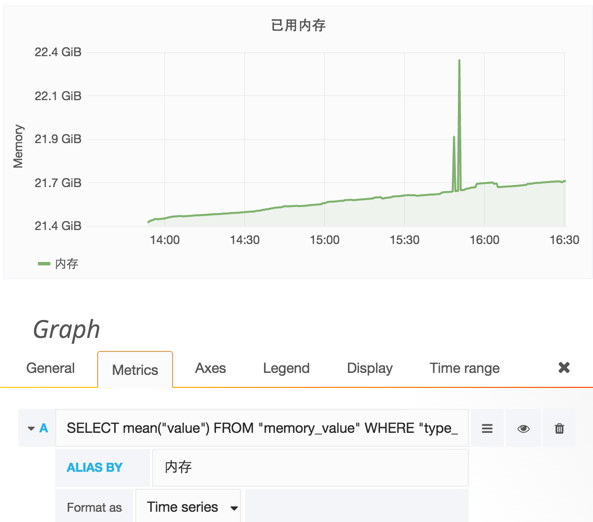
**SELECT** mean("value") **FROM** "load\_midterm" **WHERE** "host" = 'localhost' **AND** $timeFilter **GROUP** **BY** **time**($interval) fill(null)

**SELECT** mean("value") **FROM** "load\_shortterm" **WHERE** "host" = 'localhost' **AND** $timeFilter **GROUP** **BY** **time**($interval) fill(null)



**内存用量**

**SELECT** mean("value") **FROM** "memory\_value" **WHERE** "type\_instance" = 'used' **AND** $timeFilter **GROUP** **BY** **time**($interval) fill(null)



* [2016年09月10日发布](https://segmentfault.com/a/1190000006868587)

* [更多](javascript:void(0);)

# JmxTrans Docker image

https://hub.docker.com/r/jmxtrans/jmxtrans/

Short Description

Monitor JVM via JMX

Full Description

<img src="http://www.jmxtrans.org/assets/img/jmxtrans-logo.gif"/>

Connecting the outside world to the JVM.

This is a fully functional JmxTrans application instance, based on the last releaase.  
[<http://www.jmxtrans.org/>).

**Usage**

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans -P jmxtrans/jmxtrans

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans -P jmxtrans/jmxtrans start-without-jmx

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans -p 9999:9999 jmxtrans/jmxtrans

You have two commands available :

* start-with-jmx (default value)
* start-without-jmx without jmx if you think there is an extra runtime cost (we don't think so)

Example: docker run -d -v `pwd`/json-files:/var/lib/jmxtrans -P jmxtrans/jmxtrans

This will automatically create a 'json-files' volume on docker host, that will survive container stop/restart/deletion.

**Passing JMXTRANS launcher parameters**

The arguments as environment variable are :

* SECONDS\_BETWEEN\_RUNS
* HEAP\_SIZE
* PERM\_SIZE
* MAX\_PERM\_SIZE
* CONTINUE\_ON\_ERROR
* JMXTRANS\_OPTS
* JAVA\_OPTS

You might need to customize the JVM running JMXTRANS, typically to pass system properties or tweak heap memory settings.  
Use JAVA\_OPTS environment variable for this purpose :

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans

--env JAVA\_OPTS="-Dkey1=value1 -Dkey2=value2"

jmxtrans/jmxtrans

or change the timing :

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans

--env SECONDS\_BETWEEN\_RUNS=5

--env HEAP\_SIZE=1024

--env PROXY\_HOST=192.168.50.4

jmxtrans/jmxtrans

If you log into the container and exec ps -ef | grep java, you will see :

jmxtrans 9 1 25 13:09 ? 00:00:01 java -server

-Dlog4j.configuration=file:////usr/share/jmxtrans/conf/log4j.xml

-Xms1024m -Xmx1024m -XX:PermSize=384m -XX:MaxPermSize=384m -Dcom.sun.management.jmxremote

-Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.authenticate=false

-Dcom.sun.management.jmxremote.port=9999 -Dcom.sun.management.jmxremote.rmi.port=9999

-Djava.rmi.server.hostname=192.168.50.4 -jar /usr/share/jmxtrans/lib/jmxtrans-all.jar

-e -j /var/lib/jmxtrans -s 5 -c false

**Building**

Build with the usual :

docker build -t jmxtrans/jmxtrans .

Or with more arguments for a special release :

docker build -t jmxtrans/jmxtrans:256 --build-arg JMXTRANS\_VERSION=256 .

docker build -t jmxtrans/jmxtrans:259 --build-arg JMXTRANS\_VERSION=259 .

docker build -t jmxtrans/jmxtrans:260 --build-arg JMXTRANS\_VERSION=260 .

**Monitor JMXTrans with JMX (aka Inception)**

Make sure to publish JMX container’s port 9999 as the Docker host port 9999 when starting the Docker container.  
**PROXY\_HOST** is the IP where is present the JMX client (example jvisualvm).  
It is a mandatory parameter for jmx because you need a remote access to jvm instance.

Example with docker engine into a vagrantbox (static ip as 192.168.50.4)

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans -p 9999:9999 --env PROXY\_HOST=192.168.50.4 jmxtrans/jmxtrans

Example with native docker MacOSX engine.  
It is tricky despite the hidden virtualmachine (xhive) into Mac, we need to use localhost and not 172.16.123.1  
Docker for Mac is awesome but maybe hard to understand...

docker run -d -v `pwd`/json-files:/var/lib/jmxtrans -p 9999:9999 --env PROXY\_HOST=localhost jmxtrans/jmxtrans

**Questions ?**

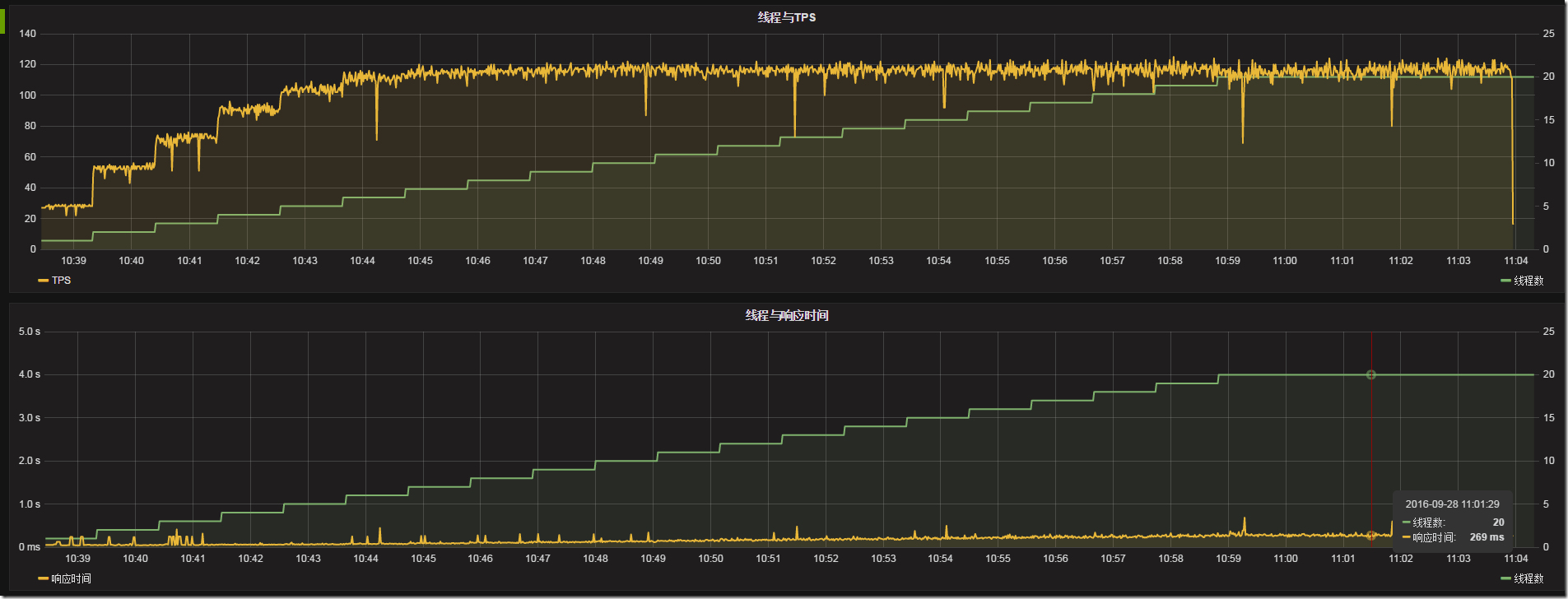
Jump on <https://groups.google.com/forum/#!forum/jmxtrans> and ask!

# Jmeter + Grafana + InfluxDB 性能测试监控 - ﹏猴子请来的救兵 - 博客园

http://www.cnblogs.com/yyhh/p/5990228.html

序章

        前几天在群里看到大神们在讨论Jmeter + InfluxDB + Grafana监控。说起来Jmeter原生的监控确实太丑了。当年不断安慰自己说“丑一点没关系，只要能用，好用，就行了！”。但是内心并不是这样，做为一名测试人员，都有一颗精益求精的心。看到有东西可以替代那原生的监控数据，果断亲自动手部署了一套。

[](http://images2015.cnblogs.com/blog/626593/201612/626593-20161201153359615-1142945723.png)

        是吧，很帅吧！数据是用InfluxDB来存储的，展示则是用Grafana来展示的

        InfluxDB是一个年轻的时序数据库，是用同样很年轻的语言“GO”开发出来的。小数据量的时候还性能还不错，但是数据量大一点，性能问题就体现出来了。不过只是收集一下我几个小时测试的数据，还是足够了。要是几个月的测试数据那还是挑别的数据库吧。

        Grafana是纯粹用js编写出来的，专门用来展示数据用的。

        基本上，就是Jmeter通过“Backend Listener”，将测试的数据上传到我的虚拟机上，通过InfluxDB来存储，Grafana来展示出来。我们访问web，稍微配置一下，就可以看到展示的数据了。

    我的InfluxDB和Grafana都是部署在一台Linux虚拟机下面的。

虚拟机 --- IP 192.168.245.131

1. 安装InfluxDB

新建InfluxDB下载源

cat << EOF | tee /etc/yum.repos.d/influxdb.repo

[influxdb]

name = InfluxDB Repository - RHEL \$releasever

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

enabled = 1

gpgcheck = 1

gpgkey = https://repos.influxdata.com/influxdb.key

EOF

使用yum下载InfluxDB

yum install -y influxdb

修改InfluxDB的配置，主要配置jmeter存储的数据库与端口号，还有需要将UI端口开放

[root@localhost ~]# vi /etc/influxdb/influxdb.conf

# 找到graphite并且修改它的库与端口

[[graphite]]

  enabled = true

  database = "jmeter"

  bind-address = ":2003"

  protocol = "tcp"

  consistency-level = "one"

# 找到admin，将前面的#号去掉，开放它的UI端口

[admin]  
  # Determines whether the admin service is enabled.  
  enabled = true

  # The default bind address used by the admin service.  
  bind-address = ":8083"

  # Whether the admin service should use HTTPS.  
  # https-enabled = false

  # The SSL certificate used when HTTPS is enabled.  
  # https-certificate = "/etc/ssl/influxdb.pem

启动InfluxDB

[root@localhost ~]# /etc/init.d/influxdb restart

Stopping influxdb...

influxdb process was stopped [ OK ]

Starting influxdb...

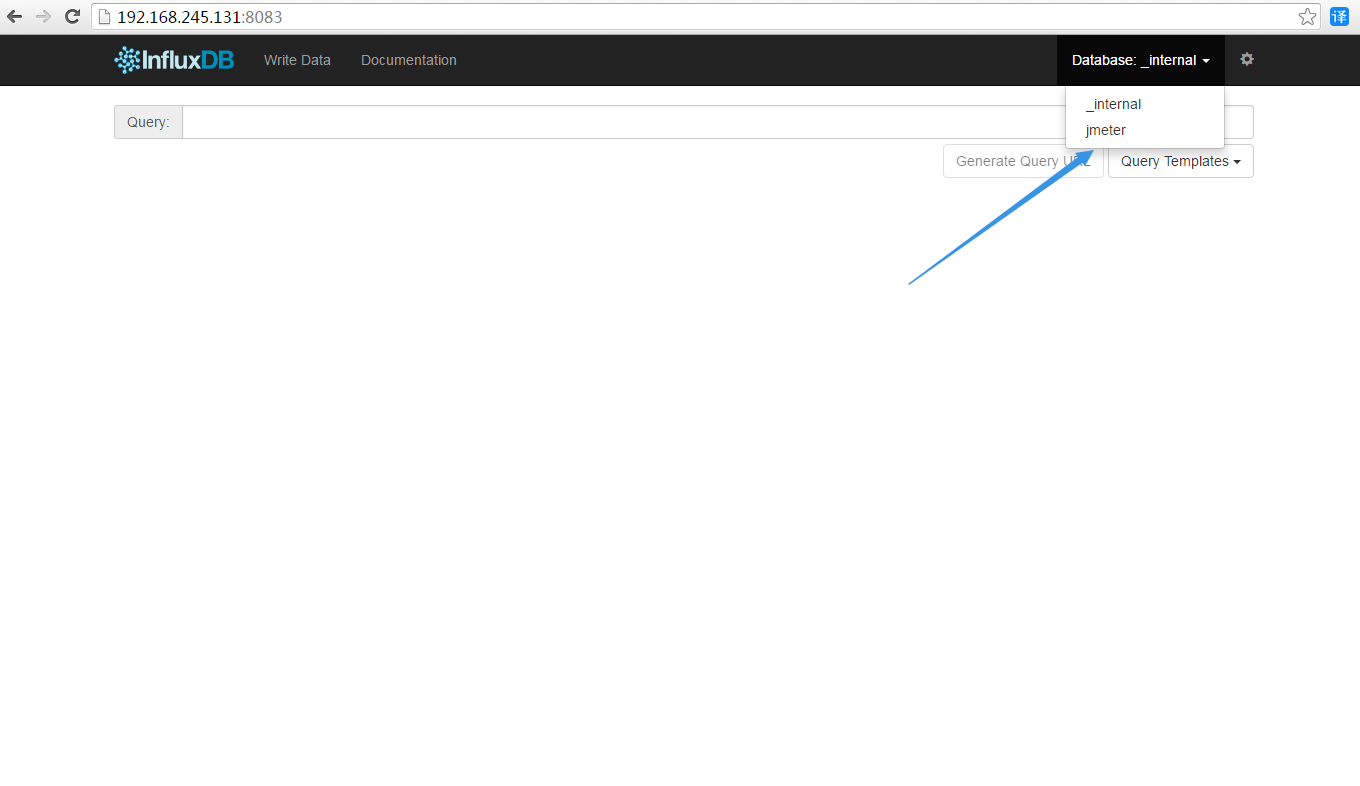
influxdb process was started [ OK ]

打开浏览器，访问虚拟机IP“http://192.168.245.131:8083”

如果启动成功应该会InfluxDB的web ui界面。查看有没有jmeter库，没有就新建一个。

在输入框中，输入如下，来新建库：

CREATE DATABASE "jmeter"

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192721873-597161591.png)

2. 安装Grafana

使用yum下载Grafana并且安装

yum install https://grafanarel.s3.amazonaws.com/builds/grafana-3.0.1-1.x86\_64.rpm

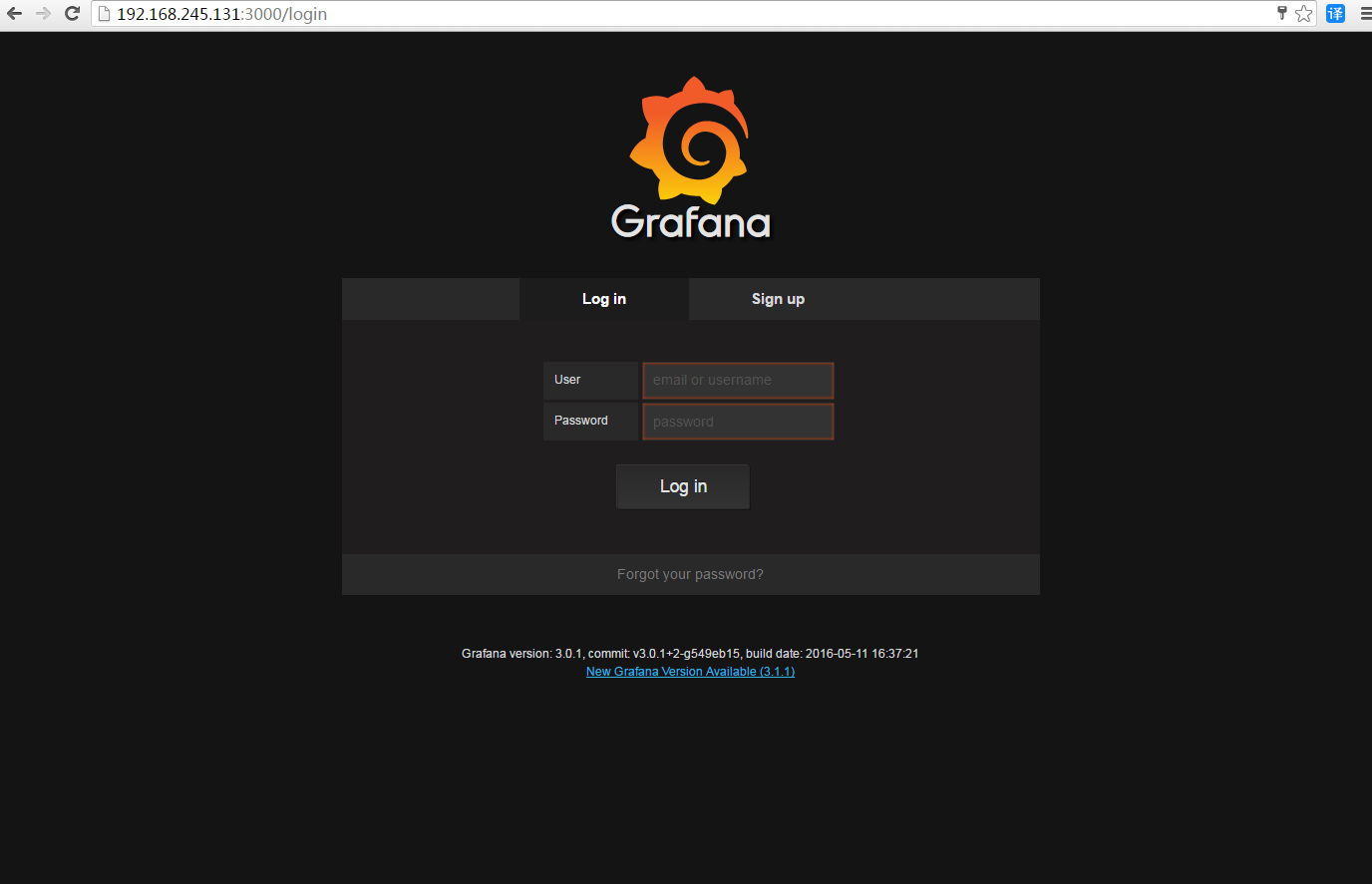
启动Grafana

[root@localhost ~]# /etc/init.d/grafana-server restart

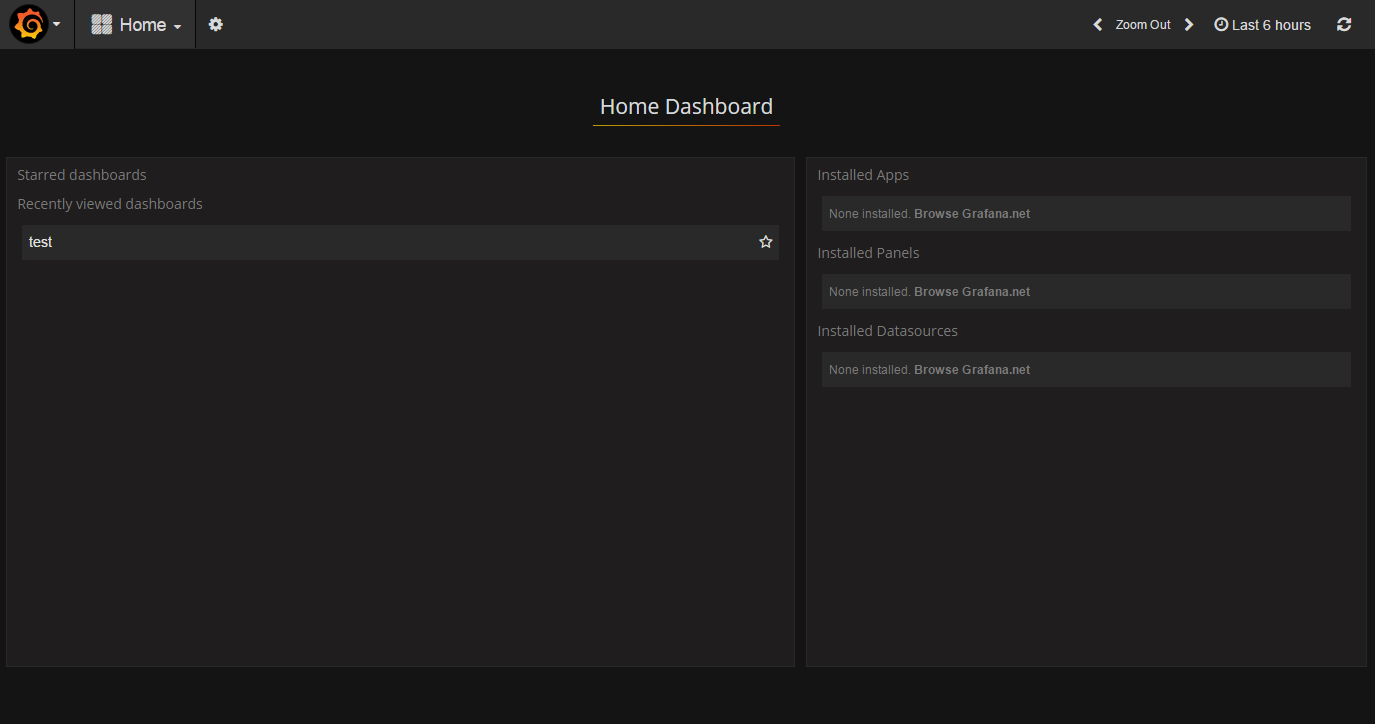
OKopping Grafana Server ...                                [  OK  ]

Starting Grafana Server: .... OK

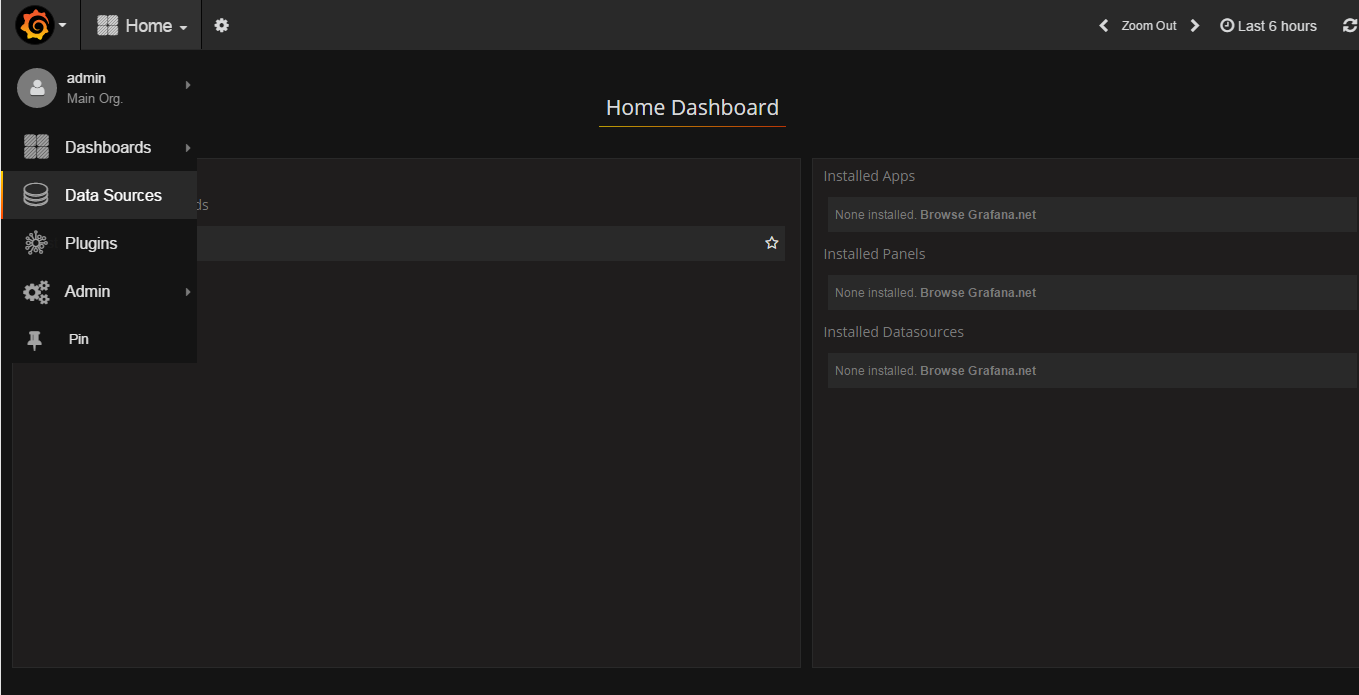
打开浏览器，访问虚拟机IP“http://192.168.245.131:3000”

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192722920-135936607.png)

输入用户名，密码登录系统。用户名与密码都是"admin"

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192723967-2135324018.png)

添加需要展示数据的数据库

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192724732-109529297.png)

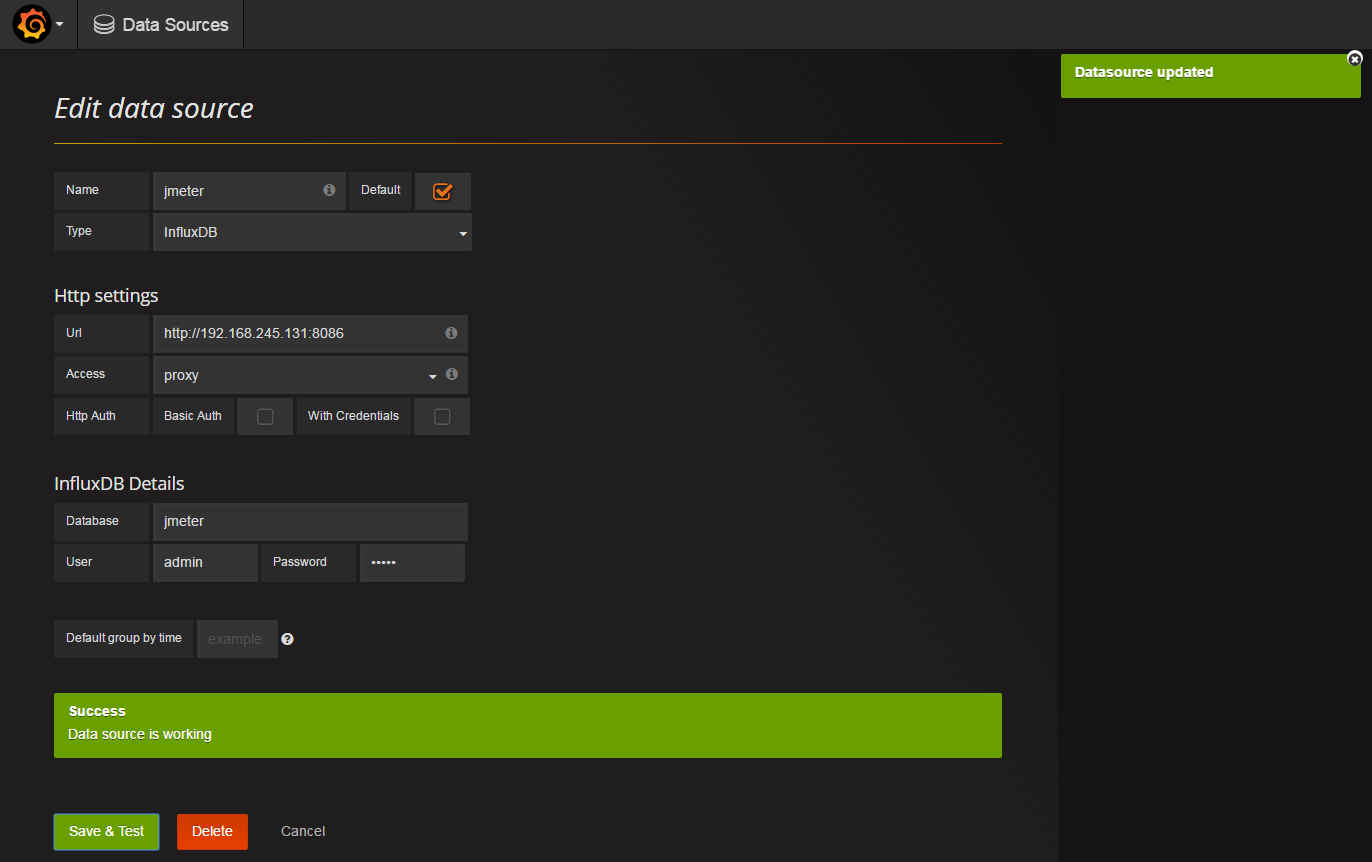
添加InfluxDB数据库配置。输入帐号密码“admin / admin”，点击Test & Save 提示“Success”说明成功了

**【注意】URL的端口是8086，而刚才配置的8083是UI的端口。**

    - 8083端口是InfluxDB的UI界面展示的端口

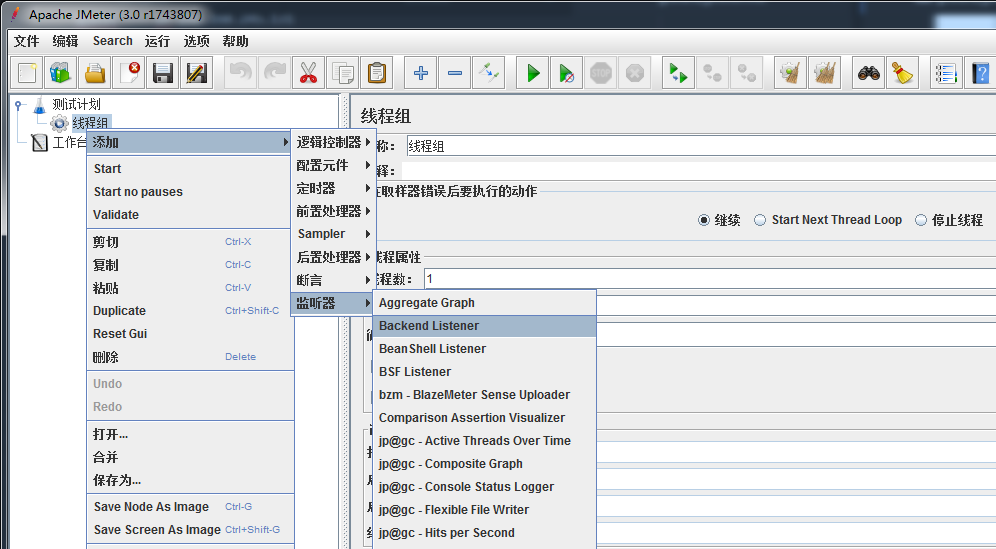
    - 8086端口是Grafana用来从数据库取数据的端口

    - 2003端口则是刚刚设置的，Jmeter往数据库发数据的端口

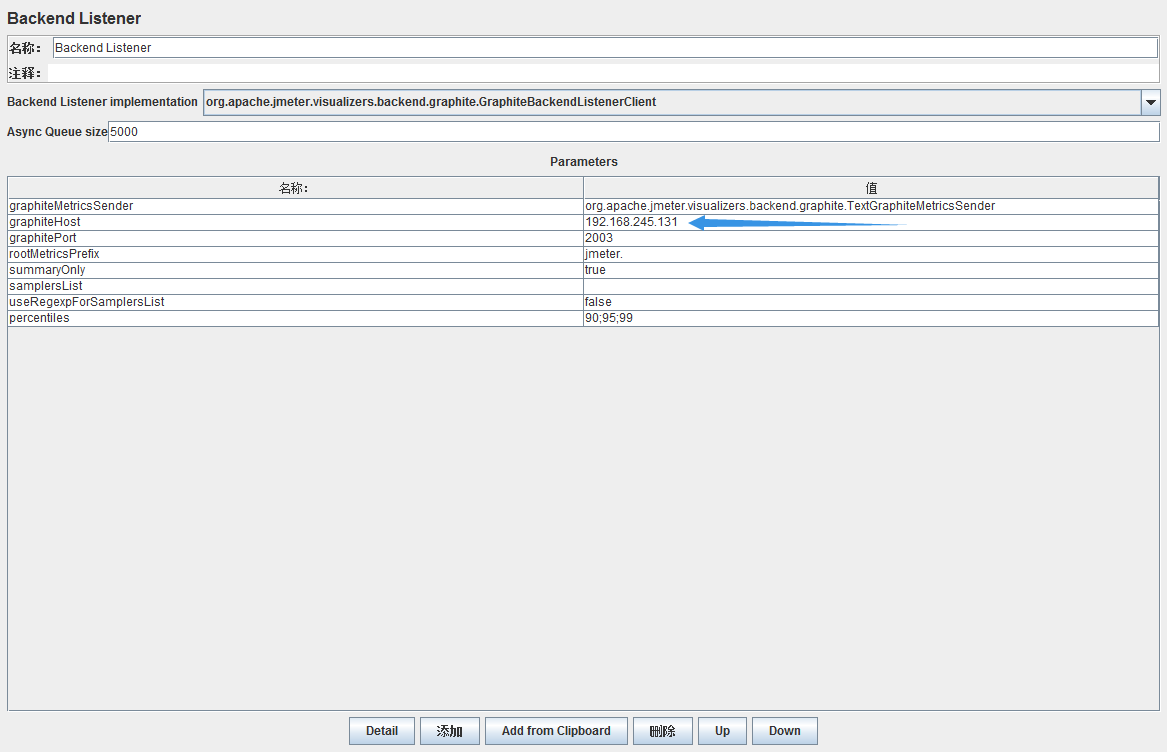
[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192726154-623885262.png)

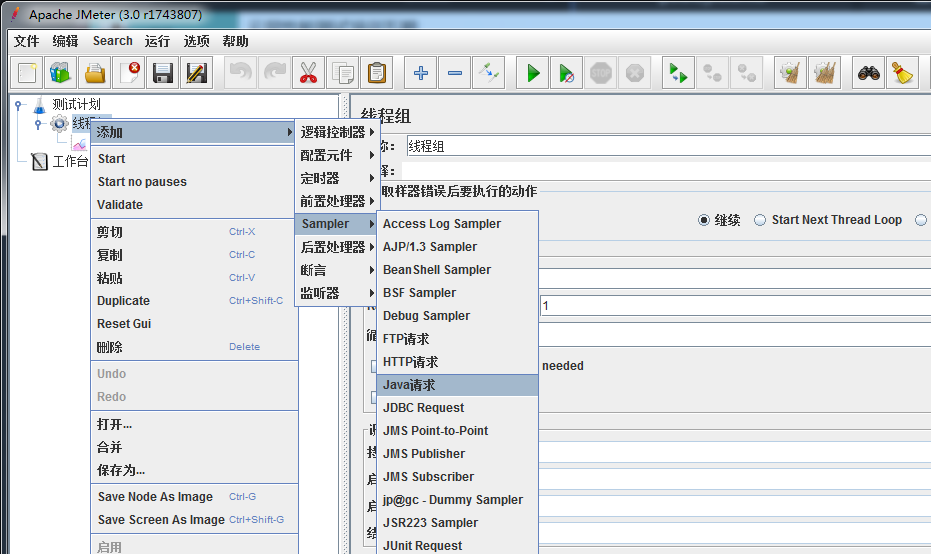
3. 配置Jmeter

1. jmeter中，添加“监听器 -> Backend Listener”

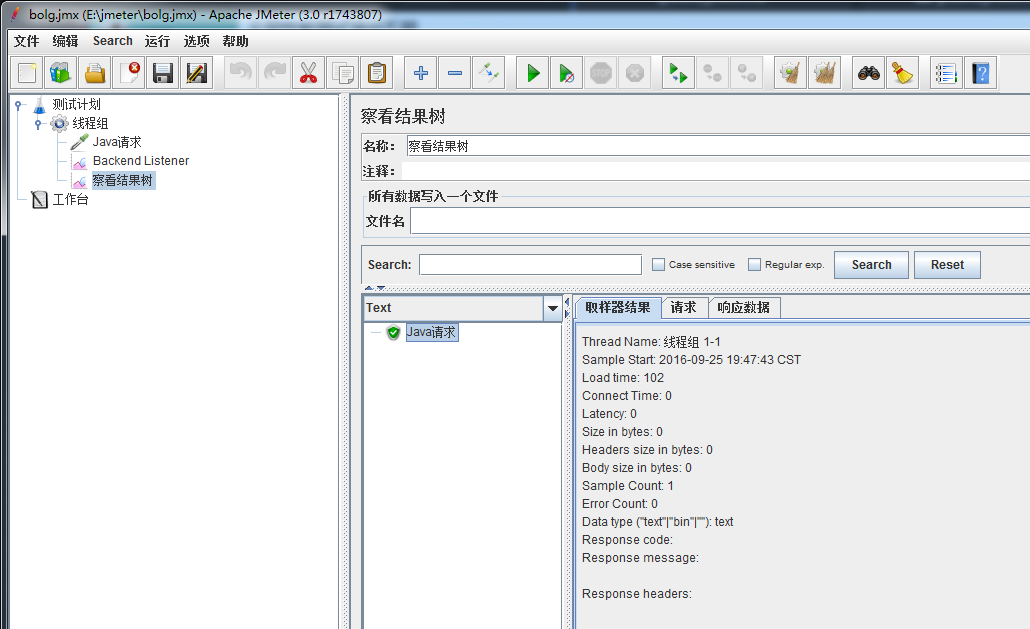
[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192727592-1123868598.png)

2. 配置“Backend Listener”,主要配置Host，如下图

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192728810-318534931.png)

3. 添加一个Java请求，方便测试。(因为想偷懒，Java请求我什么都不用写，直接运行就能成功) [](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192729638-213359995.png)

4. 添加“监听器 -> 查看结果树” 运行一下Jmeter，主要看Java请求是否发送出去了

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192730795-1819451320.png)

没有什么问题，这个时候访问InfluxDB“http://192.168.245.131:8083”， 在输入框中输入如下，点击回车：

http://192.168.0.179:8083/

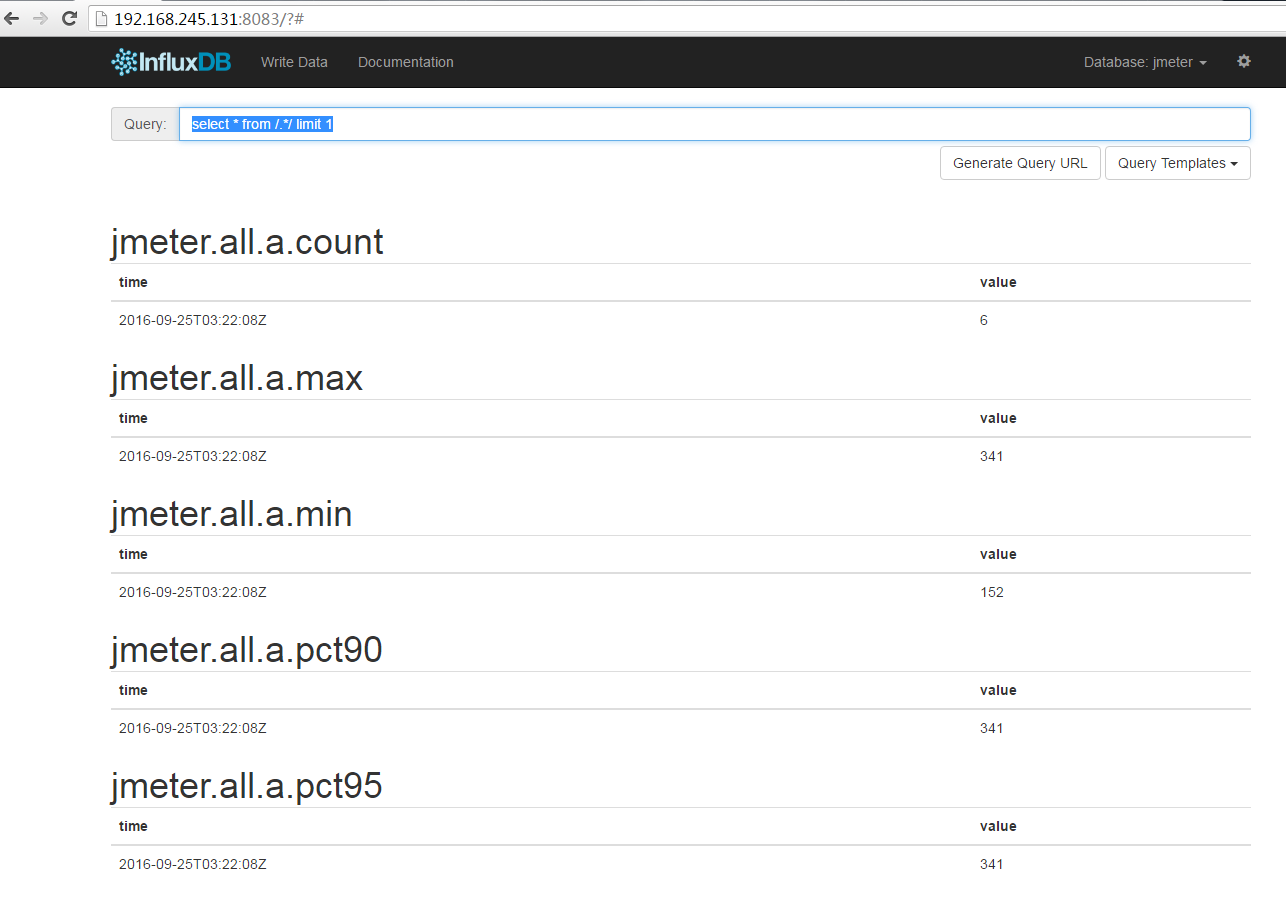
# 可以使用这个，这个是查询所有表，显示1条记录

select \* from /.\*/ limit 1

# 也可以使用这个，这个是显示所有表

show measurements

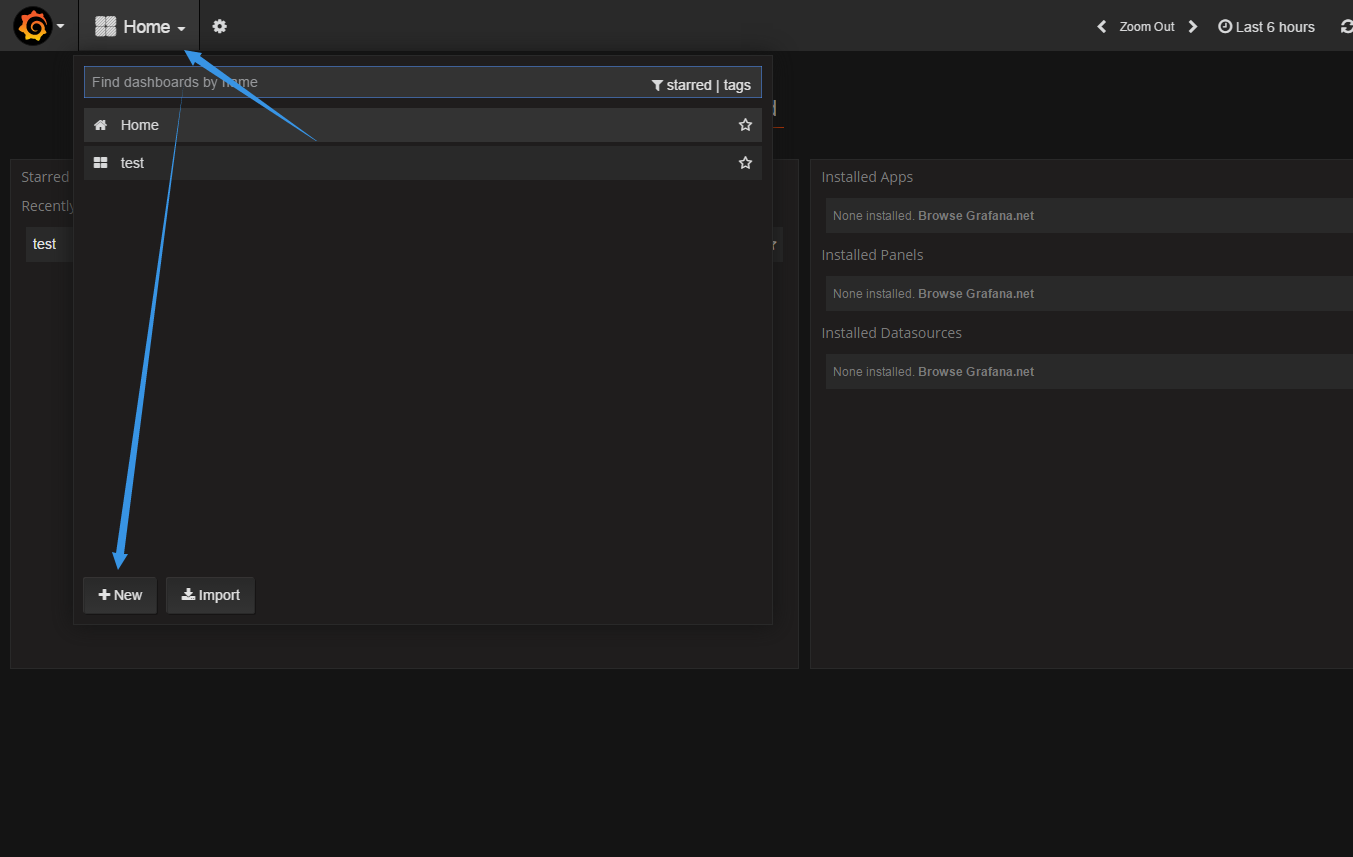
点击回车后，就应该有数据了，会出现下图：

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192732013-2117757874.png)

这个时候再回来配置Grafana，来展示这些数据

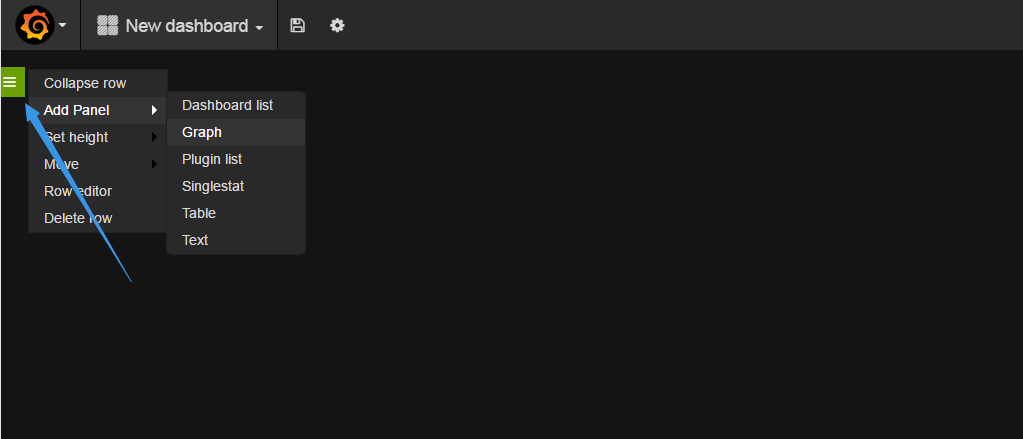
添加一个展示项目

点击“Home -> New”

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192732951-2049060818.png)

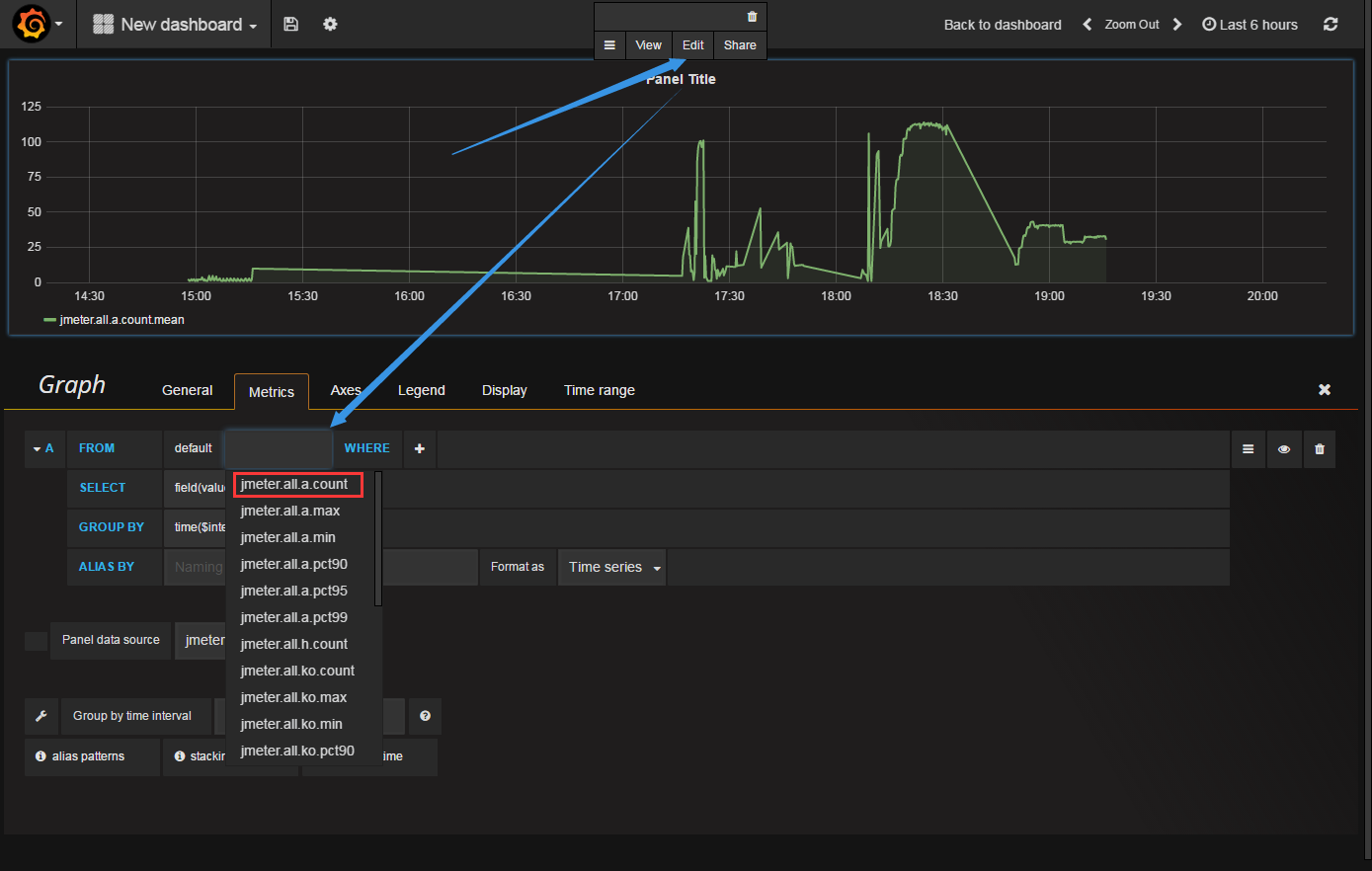
添加一个图表

点击旁边的绿点“Add Panel -> Graph”

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192733967-1025036903.png)

配置图表

配置好了，就能看到图了。如果看不到图，请用Jmeter多发几次Java请求。下图中选择监控的选项，可以在Jmeter的官网上查看到对应的解释。

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192735154-1439786703.png)

大致介绍几种我常用的监控。

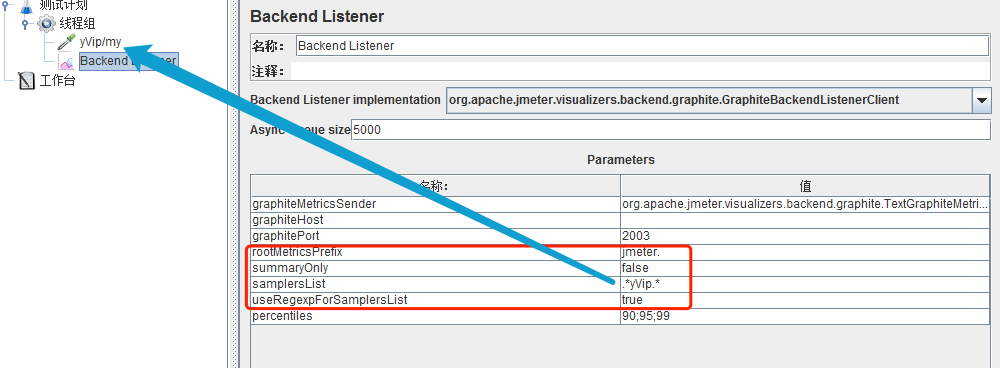
|  |  |
| --- | --- |
| 名称 | 描述 |
| jmeter.all.h.count | 所有请求的TPS |
| jmeter.<请求名称>.h.count | 对应<请求名称>的TPS |
| jmeter.all.ok.pct99 | 99%的请求响应时间 |
| jmeter.<请求名称>.ok.pct99 | 对应<请求名称>99%的请求响应时间 |
| jmeter.all.test.startedT | 线程数 |

【注意】如果要监控<请求名称>的话，Jmeter上的“Backend Listener”修改如下参数

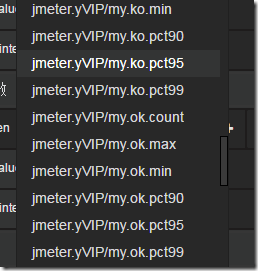
1. 将“summanyOnly”修改成False，

2. 将“userRegexpForSamplersList”修改成True，

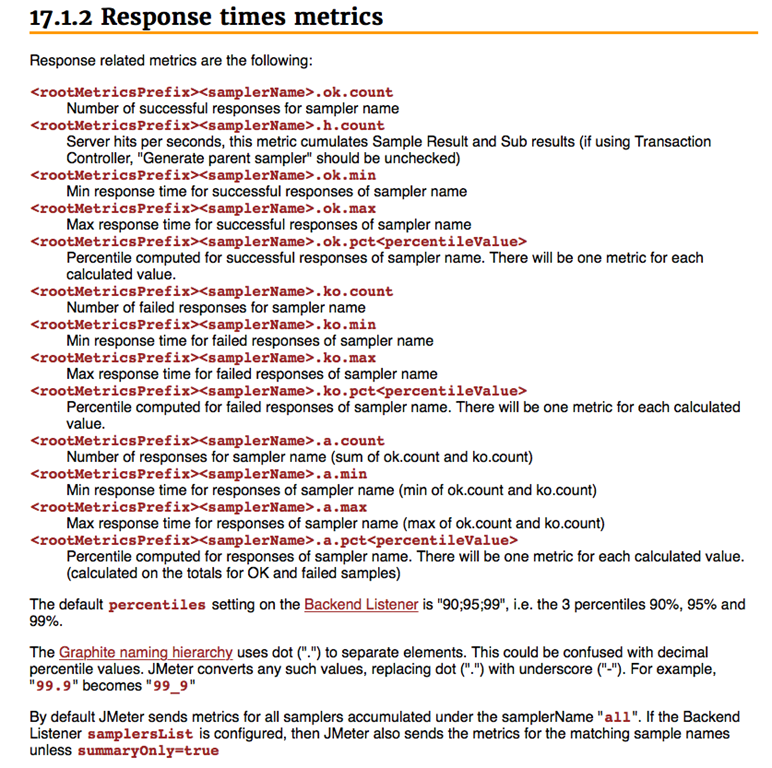
3. 并且要设置“samplersList”的值，“samplersList”是可以支持正则表达式的，“samplersList”的设置要与请求对应，否则找不到该请求。如图

[](http://images2015.cnblogs.com/blog/626593/201611/626593-20161120192737435-256567136.png)

设置好了，运行一下，在Grafana里面，就可以看到自己的请求了。

[](http://images2015.cnblogs.com/blog/626593/201610/626593-20161023162013107-1459342136.png)

如果想要了解这些监控都代表什么意思，可以访问Jmeter的官网地址去查看阅读“<http://jmeter.apache.org/usermanual/realtime-results.html>”



最后来一张帅气的图：

分类: [Grafana](http://www.cnblogs.com/yyhh/category/888723.html),[Jmeter](http://www.cnblogs.com/yyhh/category/888724.html)

