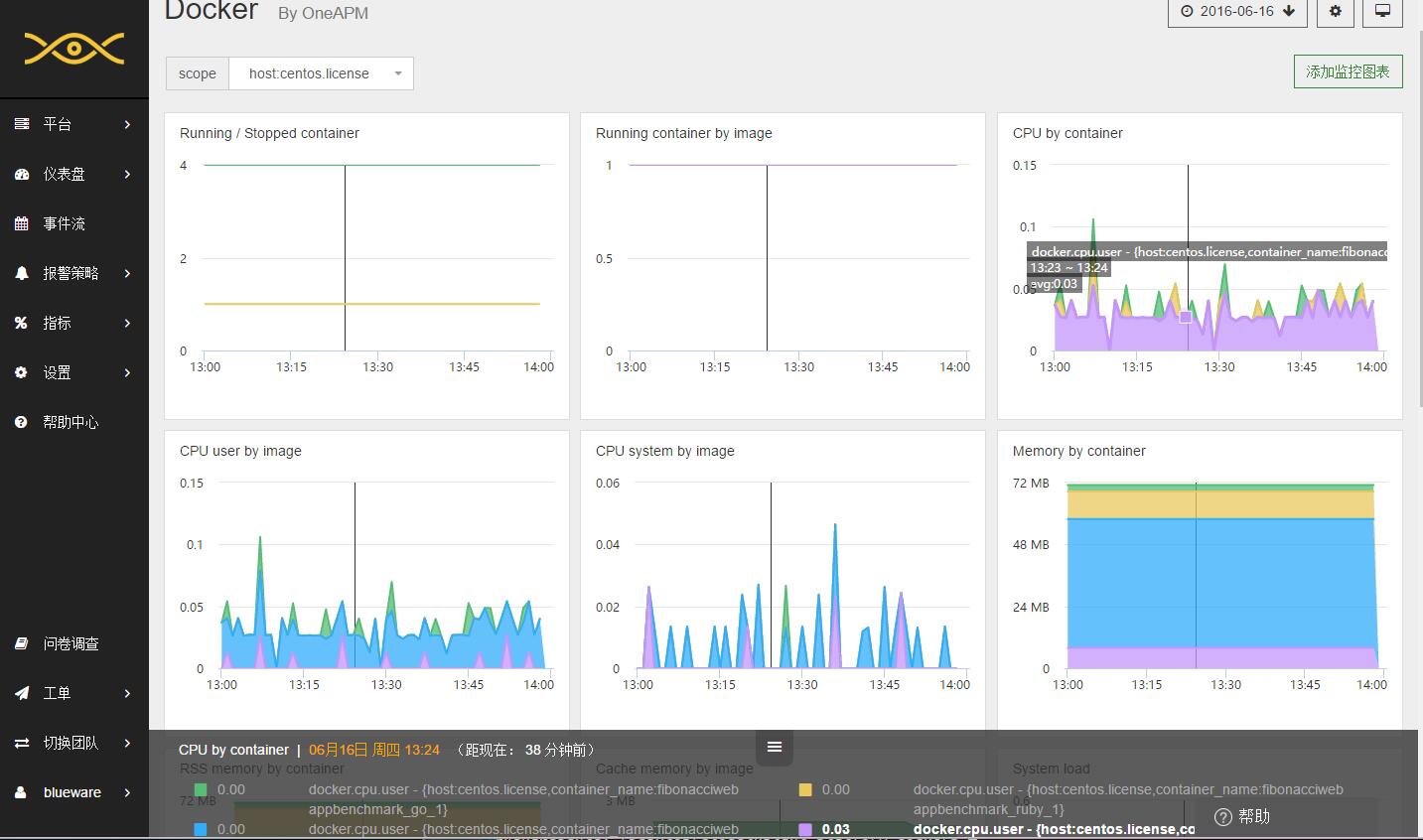
# 如何监控Docker|Docker监控|Docker性能指标|CloudInsight监控Docker

http://docs-ci.oneapm.com/services\_example/docker.html?utm\_source=Community&utm\_medium=Article&utm\_term=%E6%96%8C%E5%93%A5%E7%9A%84%20Docker%20%E8%BF%9B%E9%98%B6%E6%8C%87%E5%8D%97%E2%80%94%E7%9B%91%E6%8E%A7%E6%96%B9%E6%A1%88%E7%9A%84%E5%AE%9E%E7%8E%B0&utm\_content=wk502-508&utm\_campaign=CiArti&from=jscwpxei

Docker

Cloudinsight 支持监控 Docker 容器情况，不仅监控容器使用 cpu、network、io、disk 等做通用指标，还因为 Doker 通过 cgroup 来做资源限制，Agent 通过在宿主机本身查看对应容器的 cgroup stats 来监控容器的信息，状态等。

通过配置文件安装和直接一键安装等区别是：直接一键安装会生成一个 Cloudinsight 容器，所有的配置文件会在一个容器内，其它监控到的数据是相同的。



**性能指标**

Cloudinsight 采集 Docker 以下性能指标：

| **指标** | **单位** | **具体含义** |
| --- | --- | --- |
| docker.container.size\_rootfs | bytes | 容器中所有文件的总大小 |
| docker.container.size\_rw | bytes | 容器中被正在运行的进程创建或更改的所有文件的总大小 |
| docker.containers.running |  | 这个主机上正在运行的容器数量 |
| docker.containers.stopped |  | 这个主机上已停止的容器数量 |
| docker.cpu.system | fractions | CPU 中代表此容器的进程执行系统调用（executing system calls）的时间占比 |
| docker.cpu.user | fractions | CPU 直接控制此容器的进程的时间分数（fraction） |
| docker.image.size | bytes | 磁盘上镜像的所有图层（layers）的大小 |
| docker.image.virtual\_size | bytes | 磁盘上镜像的所有图层（layers）的大小 |
| docker.images.available |  | 顶级（top-level）镜像的数量 |
| docker.images.intermediate |  | 中间镜像的数量，它们是构成其他镜像的中间图层 |
| docker.io.read\_bytes | bytes/second | 容器上的进程每秒从磁盘读取的字节数 |
| docker.io.write\_bytes | bytes/second | 容器上的进程每秒向磁盘写入的字节数 |
| docker.mem.cache | bytes | 正用于缓存磁盘数据的存储器的量 (e.g. 可以精确地与块设备上的块相关联的存储器内容) |
| docker.mem.in\_use | fractions | 已使用内存占可用内存的比例 |
| docker.mem.limit | bytes | 容器的内存限制（如果设置） |
| docker.mem.rss | samples/second | 除缓存以外的内存使用量，用于堆栈，堆等 |
| docker.mem.swap | bytes | 容器当前已使用的交换（swap）量 |
| docker.mem.sw\_in\_use | fractions | 已使用的 swap + memory 与可使用的 swap + memory 的比例 |
| docker.mem.sw\_limit | bytes | 容器的 swap + memory 限制 |
| docker.net.bytes\_rcvd | bytes/second | 每秒从网络接收的字节数 |
| docker.net.bytes\_sent | bytes/second | 每秒向网络发送的字节数 |

具体参数含义参考 [Docker Runtime metrics](https://docs.docker.com/engine/admin/runmetrics/)

**配置 Docker 监控**

**在 Docker 组中引入 Agent**

首先确保 Docker 已正常运行在服务器上，并引入 Agent 至 Docker 组中。

usermod -a -G docker cloudinsight-agent

**编辑配置文件**

编辑配置文件 [conf.d/docker\_daemon.yaml](http://docs-ci.oneapm.com/services/)，使 Cloudinsight Agent 可以与 Docker 通信。

**init\_config**:

**instances**:

- **url**: **"unix://var/run/docker.sock"**

**重启 Agent**

重启 Cloudinsight Agent，使配置生效。

您也可以通过查看 Agent Info 信息，来验证配置是否成功。当出现以下信息，则代表安装成功。

Checks

======

[...]

docker\_daemon

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

- instance #0 [OK]

- Collected 8 metrics & 0 events

有关 Agent Info 信息的查看，请访问帮助中心，查看 [Cloudinsight Agent 常用操作](http://docs-ci.oneapm.com/quick-start/instrument.html)。

**默认标签**

Cloudinsight 采集 Docker 以下默认主机标签：

| **标签种类** | **标签含义** |
| --- | --- |
| container\_name | 容器的名称（例如 “boring\_euclid"） |
| image\_name | 镜像（image）的名称（例如 "nginx"） |
| image\_tag | 镜像（image）的标签（例如 "latest"） |
| docker\_image | 镜像（image）名的全称（例如 "nginx:latest"） |

**常见问题**

* [Unable to connect to socket. cloudinsight-agent user must be part of the ‘docker’ group](http://docs-ci.oneapm.com/qa/docker_daemon.html)。
* 若要在同一个服务器上监控多个相同的平台服务，参考[如何监控多个平台服务](http://docs-ci.oneapm.com/qa/more.html)。
* 有任何关于产品的使用疑惑，参考[常见问题](http://docs-ci.oneapm.com/qa/)。

# 2@斌哥的 Docker 进阶指南—监控方案的实现 - OneAPM 博客

http://blog.oneapm.com/oneapm-course/674.html

**过去的一年中，关于**[**Docker**](http://www.oneapm.com/ci/docker.html)**的话题从未断过，而如今，从尝试 Docker 到最终决定使用 Docker 的转化率依然在逐步升高，关于 Docker 的讨论更是有增无减。另一方面，大家的注意力也渐渐从 “Docker 是什么”转移到“实践 Docker”与“监控 Docker”上。**

本文转自刘斌博文 [「如何选择 Docker 监控方案 」](http://liubin.org/blog/2016/04/24/how-to-choose-a-docker-monitor-solution/)，文中刘斌从技术的角度深入解释了 [Docker 监控](http://www.oneapm.com/ci/docker.html)的数据采集原理，介绍了现有开源的监控方案，以及能够对 Docker 进行监控功能的主流 [SaaS](http://blog.oneapm.com/tags-SaaS.html) 服务工具。

上一篇文章中介绍了 Docker 监控目的及技术基础，本篇文章将介绍，Docker 监控方案的实现。

Docker 监控方案的实现

* 自己动手 + 开源软件
* SaaS

评价标准

* 功能
  + 满足
  + 信息详细程度
  + 查询的灵活程度
  + 报警 + API
* 灵活性
  + 定制
* 成本
  + 学习、开发
  + 维护
* 运维
  + 部署复杂程度
* 高可用

需要考虑的基本要素如上所示，不多述。

自己动手

* 灵活性强
* 成本高

这里的成本包括开发成本，开发成本可能包括招人和培训，开发时间和填坑时间。开发完了还需要维护成本，而且随着Docker的升级，可能还需要对metric的采集实现进行升级，以及各种bugfix。

自己动手打造监控方案

* 采集
* 存储
* 展示
* 报警（动作）

[StatsD](http://blog.oneapm.com/tags-StatsD.html) 是 Flickr 公司首先提出来的，后来由 Esty 公司发扬光大的一个轻量级的指标采集模块。

简单来讲，StatsD 就是一个简单的网络守护进程，基于 [Node.js](http://www.oneapm.com/ai/nodejs.html) 平台（Esty实现，其实也有其他语言版本），通过 UDP 或者 TCP 方式侦听各种统计信息，包括计数器和定时器，可以用来采集操作系统、不同数据库、中间件的数据指标，进行缓存、聚合，并发送到Graphite 等存储和可视化系统中。

StatsD 具有以下优点：

* 简单

首先安装部署简单，且StatsD 协议是基于文本的，可以直接写入和读取，方便实现各种客户端和SDK。

[Cloud Insight](http://www.oneapm.com/ci/feature.html)的探针也是采用这些方式，我们有些SDK也是基于StatsD的，目前有[Ruby](http://blog.oneapm.com/tags-Ruby.html)、[Python](http://blog.oneapm.com/tags-Python.html)和[Java](http://blog.oneapm.com/tags-Java.html)的，在  [GitHub](https://github.com/cloudinsight)  上可以看到。

* 低耦合性

StatsD 守护进程采取 UDP 这种无状态的协议，收集指标和应用程序本身之间没有依赖，不会阻塞应用，不管StatsD的状态是运行中，还是没在运行，都不会影响应用程序，应用程序也不关心StatsD是否收到数据。

* 易集成

StatsD非常容易整合其他组件，可以自己编写采集业务逻辑，发送到StatsD守护进程即可。也就是说用户的工作很简单，只需要按定义好的规则采集数据发送到Stats，然后用Graphite存储、展示，通过使用Riemann进行报警。

Tcollector

* 来源于OpenTSDB

Tcollector 是一个采集指标数据并保存到OpenTSDB的框架，你可以使用该框架自己编写采集的业务逻辑。类似StatsD，运行在客户端，收集本地的metric信息，推送到OpenTSDB。

Collectd

* System statistics collection daemon
* 存储到RRD
* 插件机制（input/output）
* 简单报警功能

Collectd即是一个守护进程，也是一个框架，类似StatsD，它性能非常好，采用C语言编写。Collectd不直接支持从Docker中取数据，但是我们可以自己编写插件来采集性能指标数据。

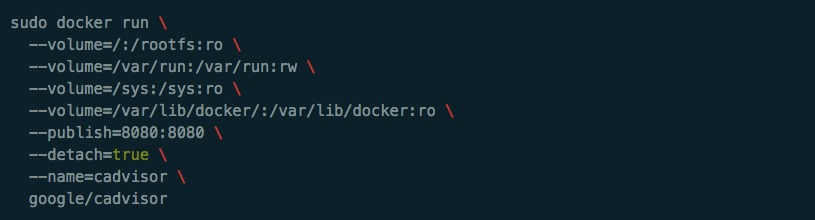
Collectd有强大的插件机制，已经实现了包括amqp、rrdtool、graphite、http、kafka、redis、mongodb、OpenTSDB以及CSV文件等在内的各种插件。

在4.3版本之后还支持简单的基于阈值检查的报警机制。



cAdvisor是一个用于收集、聚合处理和输出容器运行指标的守护进程。而且cAdvisor基本算是一个获取Docker性能数据的标配了吧。

1.



|  | true |  |
| --- | --- | --- |
|  |  |  |

一句命令就可以启动cAdvisor容器，访问8080端口即可看到性能指标数据。cAdvisor可以通过storage\_driver参数将数据存到influxdb，同时也可以将metric输出为Prometheus的格式，所以很多自定义Docker监控系统都会采取cAdvisor + Prometheus 的组合。

存储TSDB

* OpenTSDB
* Influxdb
* RRDTool
* Graphite

关于时序列数据库，可以看附录中相关的介绍文章。推荐使用OpenTSDB或者Influxdb，简单对比一下各自特点如下：

* OpenTSDB
* Java & HBase
* 易扩展（集群功能强大）
* 机器多，运维稍显麻烦
* Influxdb
* Golang
* 集群功能不太成熟
* 有类SQL的查询语句
* 单台即可工作

这两者都支持自由模式和多维度，非常适合用于采用tag机制的数据模式建模。

开源可视化工具

* Graphite
* Influxdb + Grafana
* Prometheus

光有数据是不够的，raw data没有任何意义，我们需要良好的可视化组件来展示数据和数据的内在意义，发挥数据的作用。

我们也可以将数据存储和展示交给其他开源软件。

如果你的数据采集和存储都是自己来完成的，只想使用一个外部的图形化界面的话，选Grafana应该没错，Grafana展现形式非常丰富，配置也很灵活。



以上，先到这里。

下一章，刘斌将为大家介绍 Docker 监控的开原方案，主流 SaaS 服务，及其特点。

超好用的监控软件[Cloud Insight](http://docs-ci.oneapm.com/services_example/docker.html?utm_source=Community&utm_medium=Article&utm_term=%E6%96%8C%E5%93%A5%E7%9A%84%20Docker%20%E8%BF%9B%E9%98%B6%E6%8C%87%E5%8D%97%E2%80%94%E7%9B%91%E6%8E%A7%E6%96%B9%E6%A1%88%E7%9A%84%E5%AE%9E%E7%8E%B0&utm_content=wk502-508&utm_campaign=CiArti&from=jscwpxei) 不仅能监控 Docker，还能对 [Nagios 进行更好的可视化](http://docs-ci.oneapm.com/services_example/nagios.html?utm_source=Community&utm_medium=Article&utm_term=%E6%96%8C%E5%93%A5%E7%9A%84%20Docker%20%E8%BF%9B%E9%98%B6%E6%8C%87%E5%8D%97%E2%80%94%E7%9B%91%E6%8E%A7%E6%96%B9%E6%A1%88%E7%9A%84%E5%AE%9E%E7%8E%B0&utm_content=wk502-508&utm_campaign=CiArti&from=jscwpxei)哦~

阅读更多技术文章，请访问[OneAPM 官方博客](http://blog.oneapm.com/?utm_source=Community&utm_medium=Article&utm_term=%E6%96%8C%E5%93%A5%E7%9A%84%20Docker%20%E8%BF%9B%E9%98%B6%E6%8C%87%E5%8D%97%E2%80%94%E7%9B%91%E6%8E%A7%E6%96%B9%E6%A1%88%E7%9A%84%E5%AE%9E%E7%8E%B0&utm_content=wk502-508&utm_campaign=CiArti&from=jscwpxei)。

# 3@如何选择Docker监控方案 - 自言自语

http://liubin.org/blog/2016/04/24/how-to-choose-a-docker-monitor-solution/

如何选择Docker监控方案

这是2016/4/23关于Docker性能监控的一次Meetup的内容节选，完整内容请参见文末[*链接*](https://github.com/liubin/presentations/tree/master/20160423-docker-monitoring)

大家好，非常高兴来上海跟大家分享一下我们在Docker监控方面的一点点经验。

今天我要跟大家享主题是《如何选择Docker监控方案》。

这里我将主要对Docker监控的原理和常用工具以及主流的解决方案进行简单介绍，如果大家正准备对Docker进行监控，希望这次分享能为大家带来一些帮助，如果你们已经进行了对Docker的监控，我也希望能的大家交流一下经验，跟大家互相学习一下。

同时，最后我也会安利一下SaaS这种商业模式，甚至是一种消费观念。可以说，我这次代表的不是某具体厂商，而是代表我们同样做类似服务的SaaS行业。

什么是监控

为什么监控，监控什么内容？

我们要对自己系统的运行状态了如指掌，有问题及时发现，而不让让用户先发现我们系统不能使用，打电话过来到客服，客服再反映到开发，这个过程很长，而且对工程师来说，是一件比较没面子的事情。

我们也不能一问三不知，比如领导问我们这个月的MySQL并发到了什么情况？slowsql处于什么水平，平均响应时间超过200ms的占比有百分之多少？

回答不出来这个问题很尴尬，尽管你工作很辛苦，但是却没有拿得出来的成果。不要以为没出问题就没事了，你要换位想想，站在领导的角度，领导什么都不干，你提案，他签字，出了问题领导责任也很大啊。

监控目的

* 减少宕机时间
* 扩展和性能管理
* 资源计划
* 识别异常事件
* 故障排除、分析

我们为什么需要监控我们的服务？其中有一些显而易见的原因，比如需要监控工具来提醒我服务出现了故障，比如通过监控服务的负载来决定扩容或缩容。如果机器普遍负载不高，则可以考虑是否缩减一下机器规模，如果数据库连接经常维持在一个高位水平，则可以考虑一下是否可以进行拆库处理，优化一下架构。

此外，监控还可以帮助进行内部统制，尤其是对安全比较敏感的行业，比如证券银行等。比如服务器受到攻击时，我们需要分析事件，找到根本原因，识别类似攻击，发现没有发现的被攻击的系统，甚至完成取证等工作。

Docker监控的挑战

* Docker特点
  + 像host但不是host
  + 量大
  + 生命周期短
* 监控盲点（断层）
* 微服务
* 集群
* 全方位
  + Host（VM） + Services + Containers + Apps

容器为我们的开发和运维带来了更多的方向和可能性，我们也需要一种现代的监控方案来应对这种变化。

随着不可变基础设施概念的普及，云原生应用的兴起，云计算组件已经越来越像搭建玩具的积木块。很多基础设施生命周期变短，不光容器，云主机、VM也是。

在云计算出现之前，一台机器可能使用3、5年甚至更长都不会重装，主机名也不会变，而现在，我们可能升级一个版本，就重建一个云主机或者重新启动一个容器。监控对象动态变化，而且非常频繁。即使全部实现自动化，也会在负载和复杂度方面带来不利影响。

集群的出现，应用的拓扑结构也变得复杂，不同的应用的指标和日志格式也不统一，再加上如何应对多租户，也给监控带来了新挑战。

传统的监控内包括主机、网络和应用，但是Docker出现了，容器这一层容易被忽略，成为三不管地区，监控的盲点。

有人说，容器不就是个普通的OS么？装个Zabbix的探针不就行了么？

Docker host和Docker 容器都要装Zabbix探针。。。其实问题很多。

除了容器内部看到的cpu内存情况不准之外，而且容器生命周期短，重启之后host名，ip地址都会变，所以最好在Docker host上安装Zabbix agent。

如果每个容器都像OS那样监控，则metric数量将会非常巨大，而且这些数据很可能几分钟之后就无效率了（容器已经停止）。容器生命周期短暂，一旦容器结束运行，之前收集的数据将不再有任何意义。

主要的解决方式就是对以App或者Service为单位进行监控（通过Tag等方式）。

Docker监控技术基础

* docker stats
* Remote API
* 伪文件系统

我们可以通过 docker stats 命令或者Remote API以及Linux的伪文件系统来获取容器的性能指标。

使用API的话需要注意一下，那就是不要给Docker daemon带来性能负担。如果你一台主机有200个容器，如果非常频繁的采集系统性能可能会大量占据CPU时间。

最好的方式应该就是使用伪文件系统。如果你只是想通过shell来采集性能数据，则 docker stats 可能是最简单的方式了。

docker stats命令

|  |  |
| --- | --- |
|  | $ docker stats redis1 redis2  CONTAINER CPU % MEM USAGE/LIMIT MEM % NET I/O  redis1 0.07% 796 KB/64 MB 1.21% 788 B/648 B  redis2 0.07% 2.746 MB/64 MB 4.29% 1.266 KB/648 B |

该命令默认以流式方式输出，如果想打印出最新的数据并立即退出，可以使用 no-stream=true 参数。

伪文件系统

* CPU、内存、磁盘
* 网络

文件位置大概在（跟系统有关，这是Systemd的例子）：

|  |  |
| --- | --- |
|  | /sys/fs/cgroup/{memory,cpuacct,blkio}/system.slice/${docker ps --no-trunc}.scope |

Docker各个版本对这三种方式的支持程度不同，取得metric的方式和详细程度也不同，其中网络metric是在1.6.1之后才能从伪文件系统得到。

Memory

内存的很多性能指标都来自于 memory.stat 文件

|  |  |
| --- | --- |
|  | ... ...  cache 11492564992  rss 1930993664  swap 0  pgfault 728281223  ... ...  total\_cache 11492564992  total\_rss 1930993664  total\_pgpgin 406632648  total\_pgpgout 403355412  total\_swap 0  total\_pgfault 728281223  ... ... |

前面的不带total的指标，表示的是该cgroup中的process所使用的、不包括子cgroup在内的内存量，而total开头的指标则包含了这些进程使用的包括子cgroup数据。这里我们看到的数据都是一样的，由于这里并没有子cgroup。

两个比较重要的指标：

* RSS： resident set size

进程的所有数据堆、栈和memory map等。rss可以进一步分类为active和inactive（active\_anon and inactive\_anon）。在内存不够需要swap一部分到磁盘的时候，会选择inactive 的rss进行swap 。

* cache memory

缓存到内存中的硬盘文件的大小。比如你读写文件的时候，或者使用mapped file的时候，这个内存都会增加。这类内存也可以再细分为active和inactive的cache，即active\_file和inactive\_file。如果系统需要更多内存，则inactive的cache会被优先重用。

CPU

* cpuacct.stat文件
* docker.cpu.system
* docker.cpu.user

但是比较遗憾，Docker 不会报告nice，idle和iowait等事件。

System也叫kernel时间，主要是系统调用所耗费的部分，而user则指自己程序的耗费CPU，如果User时间高，则需要好好检查下自己的程序是否有问题，可能需要进行优化。

Blkio

优先从CFQ（Completely Fair Queuing 完全公平的排队）拿数据，拿不到从这两个文件拿：

blkio.throttle.io\_service\_bytes，读写字节数 blkio.throttle.io\_serviced，读写次数

Throttle这个单纯可能有误导，实际这些都不是限制值，而是实际值。

每个文件的第一个字段是 major:minor 这样格式的device ID。

网络数据

* iptables
* 伪文件系统
* 网络设备接口
* Virtual Ethernet

针网络的监控要精确到接口级别，即网卡级别。每个容器在host上都有一个对应的virtual Ethernet，我们可以从这个设备获得tx和rx信息。

不过找到容器在主机上对应的虚拟网卡比较麻烦。这时候可以在宿主机上通过 ip netns 命令从容器内部取得网络数据。

为了在容器所在网络命名空间中执行 ip netns 命令，我们首先需要找到这个容器进程的PID。

|  |  |
| --- | --- |
|  | $ CONTAINER\_PID=`docker inspect -f '' $CONTAINER\_ID`  $ mkdir -p /var/run/netns  $ ln -sf /proc/$CONTAINER\_PID/ns/net /var/run/netns/$CONTAINER\_ID  $ ip netns exec $CONTAINER\_ID netstat -i |

或者：

|  |  |
| --- | --- |
|  | $ CONTAINER\_PID=`docker inspect -f '' nginx `  $ cat /proc/$CONTAINER\_PID/net/dev |

实际上Docker的实现也是从伪文件系统中读取网络metric的：

|  |  |
| --- | --- |
|  | $ pwd  /sys/class/net/veth559b656/statistics  $ ls  collisions rx\_crc\_errors rx\_frame\_errors rx\_packets tx\_compressed tx\_heartbeat\_errors  multicast rx\_dropped rx\_length\_errors tx\_aborted\_errors tx\_dropped tx\_packets  rx\_bytes rx\_errors rx\_missed\_errors tx\_bytes tx\_errors tx\_window\_errors  rx\_compressed rx\_fifo\_errors rx\_over\_errors tx\_carrier\_errors tx\_fifo\_errors |

Docker监控方案实现

* 自己动手 + 开源软件
* SaaS

评价标准

* 功能
  + 满足
  + 信息详细程度
  + 查询的灵活程度
  + 报警 + API
* 灵活性
  + 定制
* 成本
  + 学习、开发
  + 维护
* 运维
  + 部署复杂程度
* 高可用

需要考虑的基本要素如上所示，不多述。

自己动手

* 灵活性强
* 成本高

这里的成本包括开发成本，开发成本可能包括招人和培训，开发时间和填坑时间。开发完了还需要维护成本，而且随着Docker的升级，可能还需要对metric的采集实现进行升级，以及各种bugfix。

自己动手打造监控方案

* 采集
* 存储
* 展示
* 报警（动作）

StatsD 是 Flickr 公司首先提出来的，后来由 Esty 公司发扬光大的一个轻量级的指标采集模块。

简单来讲，StatsD 就是一个简单的网络守护进程，基于 Node.js 平台（Esty实现，其实也有其他语言版本），通过 UDP 或者 TCP 方式侦听各种统计信息，包括计数器和定时器，可以用来采集操作系统、不同数据库、中间件的数据指标，进行缓存、聚合，并发送到Graphite 等存储和可视化系统中。

StatsD 具有以下优点：

* 简单

首先安装部署简单，且StatsD 协议是基于文本的，可以直接写入和读取，方便实现各种客户端和SDK。

Cloud Insight的探针也是采用这些方式，我们有些SDK也是基于StatsD的，目前有Ruby、Python和Java的，在[GitHub](https://github.com/cloudinsight)上可以看到。

* 低耦合性

StatsD 守护进程采取 UDP 这种无状态的协议，收集指标和应用程序本身之间没有依赖，不会阻塞应用，不管StatsD的状态是运行中，还是没在运行，都不会影响应用程序，应用程序也不关心StatsD是否收到数据。

* 易集成

StatsD非常容易整合其他组件，可以自己编写采集业务逻辑，发送到StatsD守护进程即可。也就是说用户的工作很简单，只需要按定义好的规则采集数据发送到Stats，然后用Graphite存储、展示，通过使用Riemann进行报警。

Tcollector

* 来源于OpenTSDB

Tcollector 是一个采集指标数据并保存到OpenTSDB的框架，你可以使用该框架自己编写采集的业务逻辑。类似StatsD，运行在客户端，收集本地的metric信息，推送到OpenTSDB。

Collectd

* System statistics collection daemon
* 存储到RRD
* 插件机制（input/output）
* 简单报警功能

Collectd即是一个守护进程，也是一个框架，类似StatsD，它性能非常好，采用C语言编写。Collectd不直接支持从Docker中取数据，但是我们可以自己编写插件来采集性能指标数据。

Collectd有强大的插件机制，已经实现了包括amqp、rrdtool、graphite、http、kafka、redis、mongodb、OpenTSDB以及CSV文件等在内的各种插件。

在4.3版本之后还支持简单的基于阈值检查的报警机制。

cAdvisor（Container Advisor）



cAdvisor是一个用于收集、聚合处理和输出容器运行指标的守护进程。而且cAdvisor基本算是一个获取Docker性能数据的标配了吧。

|  |  |
| --- | --- |
|  | sudo docker run \  --volume=/:/rootfs:ro \  --volume=/var/run:/var/run:rw \  --volume=/sys:/sys:ro \  --volume=/var/lib/docker/:/var/lib/docker:ro \  --publish=8080:8080 \  --detach=true \  --name=cadvisor \  google/cadvisor |

一句命令就可以启动cAdvisor容器，访问8080端口即可看到性能指标数据。cAdvisor可以通过storage\_driver参数将数据存到influxdb，同时也可以将metric输出为Prometheus的格式，所以很多自定义Docker监控系统都会采取cAdvisor + Prometheus 的组合。

存储TSDB

* OpenTSDB
* Influxdb
* RRDTool
* Graphite

关于时序列数据库，可以看附录中相关的介绍文章。推荐使用OpenTSDB或者Influxdb，简单对比一下各自特点如下：

* OpenTSDB
  + Java & HBase
  + 易扩展（集群功能强大）
  + 机器多，运维稍显麻烦
* Influxdb
  + Golang
  + 集群功能不太成熟
  + 有类SQL的查询语句
  + 单台即可工作

这两者都支持自由模式和多维度，非常适合用于采用tag机制的数据模式建模。

开源可视化工具

* Graphite
* Influxdb + Grafana
* Prometheus

光有数据是不够的，raw data没有任何意义，我们需要良好的可视化组件来展示数据和数据的内在意义，发挥数据的作用。

我们也可以将数据存储和展示交给其他开源软件。

如果你的数据采集和存储都是自己来完成的，只想使用一个外部的图形化界面的话，选Grafana应该没错，Grafana展现形式非常丰富，配置也很灵活。

http://liubin.org/images/2016/04/docker-monitor/grafana.png

开源方案

* cAdvisor（经典）+ InfluxDB + Grafana
* Zabbix/Nagios/Hawkular
* Fluentd
* Prometheus
* Riemann
* ATSD（Axibase Time Series Database）

Hawkular 是一个来自于RedHat开源的监控解决方案，也包括报警等系统，可以说功能非常强大。

Hawkular基于Cassandra存储，它认为这虽然增加了运维的复杂程度，但是扩展变得容易（参考前面关于OpenTSDB和Influxdb的对比）。

Zabbix

* 最经典（SaaS软件的最大敌人）
* 架构简单、清晰
* 文档丰富
* 包括采集、触发、告警
* Agent支持用户自定义监控项
* 通过SNMP、ssh、telnet、IPMI、JMX监控
* 一套HTTP + JSON的接口

现在Zabbix已经实现了探针的自动注册，也支持了基于角色的监控对象自动发现，但是你还是需要管理这些自动配置的项目。而且监控的服务多了，Zabbix探针在监控对象上运行的脚本也会变多，需要更多的进程，可能会对正常业务产生影响。

在Zabbix中支持Docker，可以使用 Zabbix Docker Monitoring ，首先需要下载两个xml的模板文件，导入到管理界面，然后在docker host，也就是zabbix的agent的机器上，安装zabbix模块（.so），然后还得在Docker主机上启动cadvisor容器。

Graphite

主要功能：

* 存储数值型时序列数据
* 根据请求对数据进行可视化（画图）

Graphite是一个经典的时序列数据存储，容易扩展，提供了功能强大的画图Web API以及大量的函数和输出方式。Graphite有自己的可视化实现，也可以支持Grafana。

Graphite本身不带数据采集功能，你需要选择其他第三方插件，比如Collectd等。



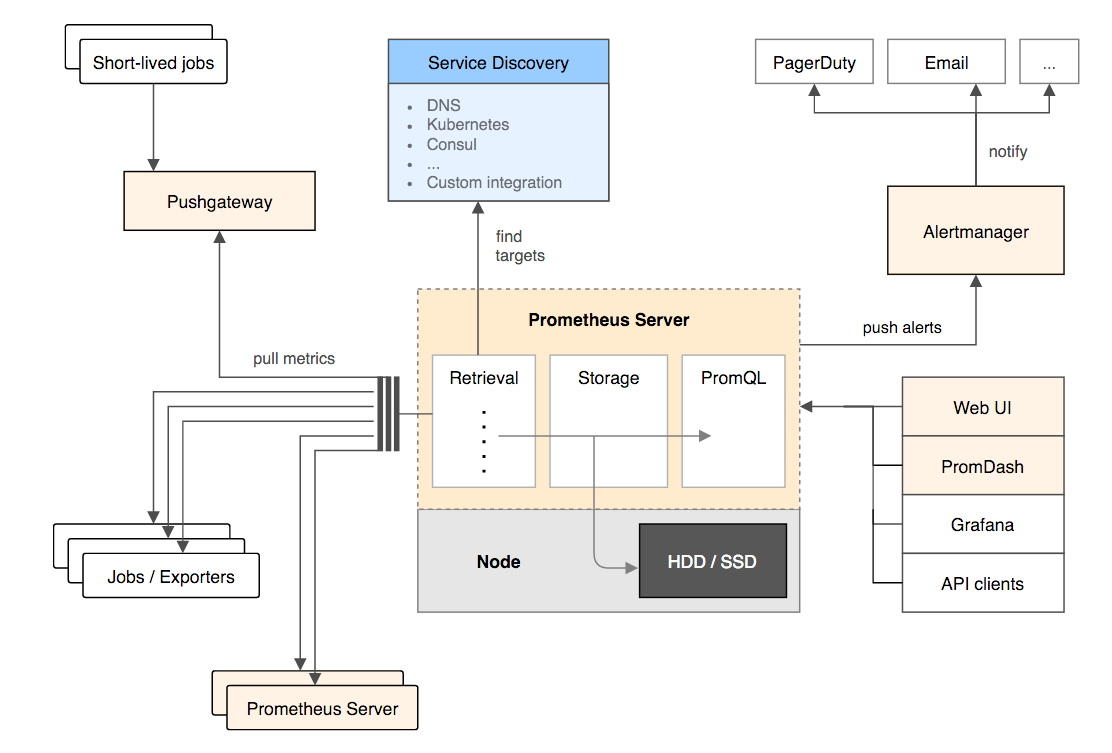
Graphite架构自下而上分为3个模块：

* whisper：创建、更新RRD文件
* carbon：以守护进程的形式运行，接收数据写入请求
  + carbon-cache：数据存储
  + carbon-relay：分区和复制，位于carbon-cache之前，类似carbon-cache的负载均衡
  + carbon-aggregator：数据集计，用于减轻carbon-cache的负载
* graphite-web：用于读取、展示数据的Web应用

Whisper使用了类似RRDtool的RRD文件格式，不像C/S结构的软件，whisper没有服务进程，只是作为library来使用，直接对文件进行create/update/fetch等操作。

Prometheus

* 一体化、仪表盘和告警
* 多维度
* 灵活查询语言
* 非分布式、单机自治
* 基于HTTP的pull模式（push需要中间网关）
* LevelDB



Prometheus是一个全套监控和预测方案，包括数据采集、存储、查询和可视化，以及报警功能。由社交音乐平台SoundCloud在2012年开发，最近也非常火。

在集群流行的时候，很少有软件专为单机、本地存储而设计。Prometheus就是这样一个软件，它采用了基于LevelDB的本地存储，并没有使用成熟的面向列的数据库。不过Prometheus 也在实验将数据存储到OpenTSDB。

Prometheus 采用了pull模式，即服务器通过探针端的exporter来主动拉取数据，而不是探针主导上报。

Prometheus部署和运维起来可能比较麻烦，需要很多组件单独安装，比如dashboard（PromDash），虽然这虽然显得更加开放和包容。不过使用Docker的话将会简单很多，通过一个Docker Compose配置文件就能建立全功能的Prometheus监控环境，网上有很多这样的例子。

Heapster

* 经典组合：heapster + Influxdb + grafana
* Sink：Kafka、stdout、gcm（Google Cloud Monitoring）、hawkular、monasca、riemann、opentsdb

Heapster是k8s的一个子项目，用于获取集群的性能数据。现在Heapster原生支持k8s和CoreOS，也可以很容易扩展来支持其他集群管理软件。

Heapster除了支持，基本的metric之外，还支持事件类型的数据，比如容器生命周期事件。

Riemann

* 事件处理
* Clojure实现

Riemann 是一个分布式监控（数据处理）系统，但它做的事情其实非常简单，就是接收事件->按定义规则处理->转发或存储到外部系统。

我们可以将Riemann 与时间序列数据库，或者基于 StastD 的方案集成使用，来弥补其他方案在报警、事件流处理这方面上的不足。

Riemann采用Clojure语言编写，这是一门Lisp方言，函数式编程语言，对于大多数习惯于过程式或者面向对象的现代人来说，理解起来有一定难度，有一定的学习曲线。

开源软件的问题点

* 灵活性受限于upstream
* 维护成本高
* 定制难度大
* 技术栈

要选择自己熟悉或者能投入精力去熟悉的技术栈产品，否则有问题没人调查，监控产品将沦为摆设。

SaaS

* turnkey解决方案
* 维护成本 ~ Zero
* 适合中小企业

对于中小型企业尤其创业公司来说，自主开发或者直接利用现有的开源工具进行监控都有一些问题，主要是成本和风险的问题。对于中小企业，应该先把精力集中在发展核心业务，能外包的就先不自己做。而且很多中小公司大家都是全栈，没有专门的运维人员，都是临时抱佛脚，随时都会变成救火队员。

SaaS最大的优点是什么？那就是免运维，开箱即用，修改的代码少甚至不需要修改代码，或者只需要简单的安装一个agent就可以工作了。很多SaaS软件的开场白都是运行一条 yum install ，然后倒上一杯咖啡等几分钟，就能看到数据了。

你的初期投入非常少，上手非常快，而且成本比较低（如果你的公司已经有数百上千服务器了，则这部分成本可能会变高，就跟是自建机房还是用云主机的对比一样）。

SaaS

* 传统APM
  + New Relic
  + AppDynamics
  + Dynatrace（Ruxit）
* 基础设施监控
  + Datadog
  + SysDig
  + Cloud Insight
  + clusterup
  + Scout

RancherLab公司有人写了篇文章，本讲稿的最后有链接，大家可以参考下。这篇文章名为《Comparing Seven Monitoring Options for Docker》，即对比了七种不同的监控Docker的方案，包括使用 docker stats命令, cAdvisor, Prometheus ，Sensu，以及saas服务 Scout, Sysdig Cloud and DataDog等方案，作为结论作者觉得Datadog是这其中最优秀的方案。

RancherLab有一个产品叫RancheOS，是一个专门为了运行Docker准备的微型Linux版本，它的口号是“The perfect place to run Docker”。跟CoreOS类似，但是貌似功能不如CoreOS多，也没有CoreOS有名，也没有CoreOS中fleet、flannel和etcd这样的组件，尤其是etcd，可以说是CoreOS的副产品，但是几乎成了Docker业界标准的kv store和服务发现组件了。

Datadog

* 国外最好
* 功能很强大
* 安装很简单

国外最流行的SaaS解决方案是Datadog，国内可能比较成熟、规模较大的应该算是Cloud Insight了。

这类服务用户只需要注册一个账号，按照安装过程通过一条命令来安装探针即可在web展示端看到数据。

要说到Datadog的不足，那就是在国外，网络延迟需要考虑，万一哪天不科学了也需要有所准备。

价格方面Datadog也比较贵。免费plan支持5台机器，而且只保留一天的数据，而且没有报警功能。收费版15美元一台主机，支持报警功能，数据存储13个月。

说道SaaS，不得不提客服，直接面对非母语客服人员交流起来肯定会有诸多不顺吧。

Cloud Insight

* 实时数据
* 历史数据
* 仪表盘
* 混合监控
* 报警功能

当然，最便宜的还是Cloud Insight，有多便宜呢，官方定价的话3个探针以下是免费的，支持超过3台主机的话，平均每天1快钱。

这只是官方定价，实际上还会便宜，貌似联系客服就可以知道有多便宜了。

Cloud Insight还支持ChatOps集成，包括国内的 BearyChat 和简聊。随着devops文化的普及，相信这些工具的重要性也会与日俱增。

Cloud Insight的图表功能也很丰富，能对任何指标以图表的方式展示，还能在图表上叠加事件，比如报警通知、服务启动停止等，能在观察到metric变动趋势的同时，看到相应的时间，了解metric发生变动的原因。比如CPU load超过一定值时，可能触发报警，这时你能在图表上看到相应的事件，同样，如果CPU load一直不是很低，但是从某一时间点开始变低了，你可能也能从一次新的代码部署中了解原因。

Sysdig

* 免费工具
* SaaS服务 Sysdig Cloud
* 拓扑可视化

可以认为sysdig是strace + tcpdump + htop + iftop + lsof + 众多linux常用的系统监控命令的合体。

如果你熟悉tcpdump，那么你知道它能还原整个网络流量，而sysdig则是操作系统级别的监控工具，能捕捉到所有OS事件和数据。

而且sysdig原生支持Linux container，包括Docker和LXC，提供了基本的指标监控信息。除了性能指标，sysdig还能采集trace等日志信息，用于以后的问题分析和解决。

Sysdig Cloud是sisdig的SaaS版，除了基本的单机sysdig功能之外，还提供了跨平台跨基础设施的组件间依赖关系的可视化。

Librato

* 数据聚合平台
* 简单探针
* 图表和报警
* 价格不贵

Librato是一个数据聚合平台，而不是严格意义的监控系统。

Librato很容易从AWS CloudWatch和Heroku获得数据，如果是自己监控主机或者Docker，需要使用Collectd框架。它也有很多插件，可以从StatsD、Riemann等数据源采集数据。

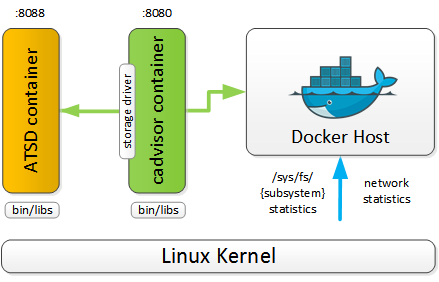
Librato的探针虽然功能不是强大，但是他提供了丰富的实时在线数据处理功能，用户可以使用DSL对任意时间序列数据组合进行数学运算，比如加减乘除、比率导数等。还支持和时间窗口滑动功能，即跟过去某一段时间进行比较。

Librato也支持报警，基于metric和条件，设置报警信息。

Librato是按照metric的个数和时间分辨率来收费的，在官网的主页上有一个大概的估算，如果你有20个metric，并且时间间隔为60秒，则一台服务器的价格只有2美元1个月，这比datadog要便宜。

Axibase（ATSD）

* 非开源TSDB
* 支持报警
* 预测功能



作为TSDB，ATSD支持长时间存储高精度的metric数据。ATSD支持多种数据采集工具和协议，比如tcollector, Collectd，当然ATSD也支持从多台Docker主机手机指标数据，并长期保存，进行可视化和分析。

除了传统的时间序列数据，ATSD还支持属性（Properties）和消息这两种类型的数据。属性一般用于保存meta data，这有点类似标签。

和OpenTSDB一样，ATSD也支持tag，我们可以使用这些tag进行过滤和聚合。

ATSD内置了自动回归推断算法（holt-winters，arima），可以提早预测故障。预测功能的准确性取决于数据的采集频率，保存时间（也就是数据量大小）和算法。

最大的遗憾，就是ATSD不是开源，也不是免费的软件，不过他们提供了一个社区版可以免费使用，但是你只能在一个节点上安装ATSD，而且不能用于盈利性服务，也不能对软件进行修改以及再发布。

如果我们只是评估一下，或者想自己构建监控方案，它的产品设计还是非常值得我们来借鉴一下。

SaaS的挑战

* 数据敏感性

采用SaaS，意味着你的数据都将会保存到公网，可能会带来心理不安全感。实际上SaaS反而会更安全些，尤其是对中小公司没有专门的安全运维团队的情况下。

* 成本（迁移和使用成本）

一般来说这是一个一次性投入成本。

* 内部抵抗（观念、个人爱好）

来自技术人员自身的抵抗，不是每个人都喜欢自己变得轻松，使用SaaS可能会给一些人带来工作上的不充实感。

趋势

* 标签机制

一种观点：监控服务状态胜过监控个别容器，通过tag机制对服务整体的性能指标进行聚合。

Tag可以是任何维度，比如BU，地区，服务，甚至个别容器。

Docker和Kubernetes等都支持label机制，即tag机制。

* docker daemon –label com.example.group=“webserver”
* docker run –label com.example.group=“webserver”
* Dockerfile: LABEL com.example.group=“webserver”
* 通过API打通

通过API化实现共赢。开源软件比如fluentd、Collectd等，都支持插件功能。包括Docker本身，也在1.11版本中，采用了runC作为容器运行时，在上面通过containerD来统一控制，来支持符合OCI标准的容器。

* Total解决方案

包括从探针到展示、告警，就是现在类似Datadog和Cloud Insight这样的产品，以及支持中间件的详细程度，就像一个大市场，什么都能买到，不管你用什么软件，平台都能提供监控。

这就像是去了酒吧突然想吃碗拉面，然后竟然酒吧能给你做出来的那种感觉。

另一个层面就是从RUEM（实时用户体验管理）到基础设施层的 **打通** 。比如你看到某URL的用户HTTP响应较慢，如果不能跟后端的APM打通，你尽管能识别出问题，但是你不知道如何解决。如果和后端的APM以及基础设施监控打通，你就能定位到HTTP响应慢时，相应的后端代码的位置，并根据后端代码的位置从而进一步找到MySQL的监控数据以及系统异常事件，立刻知道问题的根本原因所在并解决问题，可以说效率应该能提高几个数量级。

系统监控工具如果能够做到 All in One，真的对解决人力和时间成本上有非常大的帮助。

* 拓扑可视化

跨组件、跨基础设施和应用，自动识别组件以及组件之间的依赖关系，以帮助更好的发现问题和解决问题。

* Weave Scope
* Ruxit
* Sysdig

参考：

* [本次分享全部资料](https://github.com/liubin/presentations/tree/master/20160423-docker-monitoring)
* [Comparing Seven Monitoring Options for Docker](http://rancher.com/comparing-monitoring-options-for-docker-deployments/)
* [How to collect Docker metrics](https://www.datadoghq.com/blog/how-to-collect-docker-metrics/)
* [时序列数据库武斗大会](http://liubin.org/blog/2016/02/18/tsdb-intro/)
* [Fluentd Docker Metrics Input Plugin](https://github.com/kiyoto/fluent-plugin-docker-metrics)
* [Docker Runtime metrics](https://docs.docker.com/v1.8/articles/runmetrics/)

[Tweet](http://twitter.com/share)

# 2@统一标准？超越APM统一性能监控优势\_安防监控新闻-中关村在线

http://security.zol.com.cn/546/5465796.html

2015-10-20 05:40:00

工具供应商希望能够以更少、更有效的轻量级监控工具来监控整个环境，据研究副总裁JonahKowall在出席Gartner公司的IT基础架构和运营管理峰会的统一管理讨论期间表示。

　　如下，我们将分析传统性能监测工具的局限性及下一代统一监控工具潜在的优势。

　　传统性能监控工具的缺点

　　每家企业都在使用软件工具来测量性能。这其中包括应用程序性能监控(APM)，网络性能管理(NPM)，服务器性能管理，存储性能管理以及数据中心的其他元素。但是，传统的工具给IT设计师和架构师们带来了一系列难以克服的问题。

　　例如，APM，是用户体验和应用程序为企业提供的服务质量的一个重要指标。但传统的APM工具分散在数据中心的各处，可能无法为企业IT领导们提供他们所需要了解的可用信息。

　　通常，它们是带来麻烦最多的工具。许多企业依靠筒仓执行其核心任务，如服务器，网络，存储，甚至虚拟化管理。由于大多数工具都是基于专有软件，他们通常不能整合或共享数据，从而限制了孤岛筒仓之间的视觉角度。

　　而工具的数量是成倍增长的，因为每个功能分区都会引入了新的软件工具。

　　kowall在现场抽查了与会人员，发现40%的受访者所在的企业均依赖6到10款不同的工具来监控他们的环境。

　　数据收集方法带来了另一个问题。代理是最为普遍的从监测要素查询数据的方法，但代理会影响性能，而且较旧的代理可以成为攻击的潜在媒介。这两个因素经常被忽略，因为数据中心的架构缺乏一个连贯的计划，以适应或维持监控代理。不过，有供应商锁定的元素，因为许多工具供应商继续在推动基于代理的数据收集和传递。

　　但这些抱怨并不能解决[现代](http://detail.zol.com.cn/Image_record_index/subcate570_32925_list_1.html)IT企业不断变化的现实问题。企业的孤岛筒仓不再是静态的，而应用程序也越来越相互依存和动态。例如，应用程序可能会依赖于数据库或其他应用程序，并可能随着用户需求的变化需要大规模计算。这样，配置应用程序就会涉及到网络，更大的存储，服务器等之间的相互依存。

　　应用监控必须更好地收集和报告数据在一个广阔的数据中心功能区的情况——但同时也需要施以较少的需求和更有效的操作环境。

　　统一的监测获得长足的进展

　　统一的性能监控的理念已经在监测工具供应商和用户之间获得了广泛的好评。我们的目标是提供一个简单且具有成本效益的工具，能够跨功能区自动实现优越的监控水平，同时没有代理的陷阱。

　　统一的监控可以利用应用程序编程接口(API)，现在可查询系统以收集和报告范围日益扩大的信息，Kowall说。如此高的水平的工具也可以洞察连接系统，以便更好地发现和拓扑分析。一些统一的监测工具也可以执行日志文件分析来关联事件，并在设备复杂的环境进行数十个甚至上百个故障排除。

　　这一的影响是显著的。考虑到一台虚拟机(VM)哪怕中断短短几分钟，但监测新的VM往往需要人工手动安装代理，以及昂贵的许可和配置添加虚拟机的监视范围的耗时成本。理想的统一监控能够发现和应对新的虚拟机，自动使用现有的API，不需要再安装软件，很少(如果有的话)进行设置。

　　统一监控工具的供应商队伍在不断发展壮大。特别是一些成熟的供应商已经在生产轻巧，API驱动的监控，以及日志文件的分析能力。Kowall指出，包括AccelOps公司，CATechnologies公司，Centerity系统公司，GroundWorkInc.公司，Ipswitch公司，Kaseya，ManageEngine，ScienceLogic公司，SolarWinds公司和Zenoss公司等供应商均采用日志文件分析包括惠普，IBM，Elasticsearch，VMware，Splunk公司和SumoLogic。

　　统一性能监控增长带来的烦恼

　　虽然统一监控为企业带来的好处越来越多，但该技术没有完全准备好取代所有其他的APM或NPM工具。当然，采用统一管理肯定可以降低工具的总数量，大大降低旧的传统工具的负担和风险，同时配合企业精心挑选的APM和NPM产品，作为补充，Kowall说。

　　例如，设计用于测量最终用户体验，并提供早期问题预警的应用性能的工具可能会继续依赖于定制，代理驱动的工具。这是因为代理可以容忍中断连接，悄悄地收集数据，当连接可用时与集中收集的服务器共享。

　　基于API的监控工具不会很快消失。企业也可以继续像监控关键业务交易一样使用其他工具监控，实施复杂的故障排除，性能调整和容量规划等具体任务。

　　对于统一监控工具而言，一个持续的挑战在于对API的依赖需要其他软件供应商在他们的产品开发和测试方面做更多的工作。尽管有着很好的前景，但没有统一的标准就不能保证一套API套件是合适可用的，某些监控功能可能根本无法使用或无法实现跨异构平台的一致性。

　　数据中心企业所面临的承诺和陷阱是非常明确的。引入一款统一的监测工具已被证明是需要精心策划的项目，并需要针对不同的产品路线图进行大量的测试，因为其可能会影响API的可用性。只有这样，你才能在您的数据中心推出统一的性能监控。

# 2@Application performance management - Wikipedia

https://en.wikipedia.org/wiki/Application\_performance\_management

In the fields of [information technology](https://en.wikipedia.org/wiki/Information_technology) and [systems management](https://en.wikipedia.org/wiki/Systems_management), **application performance management**  (**APM**)  is the monitoring and management of performance and availability of [software](https://en.wikipedia.org/wiki/Software) applications. APM strives to detect and diagnose complex application performance problems to maintain an expected [level of service](https://en.wikipedia.org/wiki/Level_of_service). APM is "the translation of [IT metrics](https://en.wikipedia.org/wiki/Performance_metric) into business meaning ([i.e.] value)."[[1]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-1)

Contents

  [hide]

* [1Measuring application performance](https://en.wikipedia.org/wiki/Application_performance_management#Measuring_application_performance)
* [2Current issues](https://en.wikipedia.org/wiki/Application_performance_management#Current_issues)
* [3The APM conceptual framework](https://en.wikipedia.org/wiki/Application_performance_management#The_APM_conceptual_framework)
  + [3.1End user experience – (primary)](https://en.wikipedia.org/wiki/Application_performance_management#End_user_experience_.E2.80.93_.28primary.29)
  + [3.2Runtime application architecture (secondary)](https://en.wikipedia.org/wiki/Application_performance_management#Runtime_application_architecture_.28secondary.29)
  + [3.3Business transaction (primary)](https://en.wikipedia.org/wiki/Application_performance_management#Business_transaction_.28primary.29)
  + [3.4Deep Dive Component Monitoring (secondary)](https://en.wikipedia.org/wiki/Application_performance_management#Deep_Dive_Component_Monitoring_.28secondary.29)
  + [3.5Analytics/reporting (primary)](https://en.wikipedia.org/wiki/Application_performance_management#Analytics.2Freporting_.28primary.29)
* [4See also](https://en.wikipedia.org/wiki/Application_performance_management#See_also)
* [5References](https://en.wikipedia.org/wiki/Application_performance_management#References)
* [6External links](https://en.wikipedia.org/wiki/Application_performance_management#External_links)

**Measuring application performance[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=1)**]**

Two sets of [performance metrics](https://en.wikipedia.org/wiki/Performance_metrics) are closely monitored. The first set of performance metrics defines the performance experienced by end users of the application. One example of performance is average response times under peak load. The components of the set include load and response times.

* The load is the volume of transactions processed by the application, e.g., transactions per second (tps), requests per second, pages per second. Without being loaded by computer-based demands for searches, calculations, transmissions, etc., most applications are fast enough, which is why programmers may not catch performance problems during development.
* The response times are the times required for an application to respond to a user's actions at such a load.[[2]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-2)

The second set of performance metrics measures the [computational resources](https://en.wikipedia.org/wiki/Resource_(computer_science)) used by the application for the load, indicating whether there is adequate capacity to support the load, as well as possible locations of a performance bottleneck. Measurement of these quantities establishes an empirical performance baseline for the application. The baseline can then be used to detect changes in performance. Changes in performance can be correlated with external events and subsequently used to predict future changes in application performance.[[3]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-3)

The use of APM is common for Web applications, which lends itself best to the more detailed monitoring techniques.[[4]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-4) In addition to measuring response time for a user, response times for components of a Web application can also be monitored to help pinpoint causes of delay. There also exist [HTTP](https://en.wikipedia.org/wiki/HTTP) appliances that can decode transaction-specific [response times](https://en.wikipedia.org/wiki/Round-trip_delay_time) at the Web server layer of the application.

In their *APM Conceptual Framework*, [Gartner](https://en.wikipedia.org/wiki/Gartner) Research describes five dimensions of APM:[[5]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-5)[[6]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-6)[[7]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-7)[[8]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-8)

* End [user experience](https://en.wikipedia.org/wiki/User_experience) monitoring – ([active](https://en.wikipedia.org/wiki/Synthetic_monitoring) and [passive](https://en.wikipedia.org/wiki/Passive_monitoring))
* Application runtime architecture discovery and modeling
* User-defined transaction profiling (also called [business transaction management](https://en.wikipedia.org/wiki/Business_transaction_management))
* Application component monitoring
* Reporting & Application [data analytics](https://en.wikipedia.org/wiki/Data_analytics)

In 2016, [Gartner](https://en.wikipedia.org/wiki/Gartner) Research has updated its definition, into three main functional dimensions:[[9]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-9)

* End-user experience monitoring (EUEM) has been evolved to Digital experience monitoring (DEM);
* A new dimension, Application discovery, tracing and diagnostics (ADTD), combines three formerly separate dimensions (Application topology [runtime architecture] discovery and visualization, User-defined transaction profiling, and Application component deep-dive), since all three are primarily focused on problem remediation and are interlinked;
* Application analytics (AA).

**Current issues[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=2)**]**

Since the first half of 2013, APM has entered into a period of intense competition of technology and strategy with a multiplicity of vendors and viewpoints.[[10]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-10)This has caused an upheaval in the marketplace with vendors from unrelated backgrounds (including network monitoring, systems management, application instrumentation, and web performance monitoring) adopting messaging around APM[[*which?*](https://en.wikipedia.org/wiki/Wikipedia:Avoid_weasel_words)]. As a result, the term APM has become diluted and has evolved into a concept for managing application performance across many diverse computing platforms, rather than a single market.[[*clarification needed*](https://en.wikipedia.org/wiki/Wikipedia:Please_clarify)][[11]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-11)

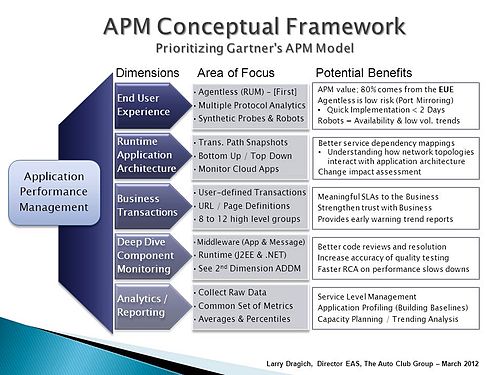
Two challenges for implementing APM are (1) it can be difficult to instrument an application to monitor application performance, especially among components of an application, and (2) applications can be [virtualized](https://en.wikipedia.org/wiki/Platform_virtualization), which increases the variability of the measurements.[[12]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-12)[[13]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-13) To alleviate the first problem [application service management](https://en.wikipedia.org/wiki/Application_service_management) (ASM) provides an application-centric approach, where business service performance visibility is a key objective. The second aspect present in distributed, virtual and [cloud-based](https://en.wikipedia.org/wiki/Cloud_computing) applications poses a unique challenge for application performance monitoring because most of the key system components are no longer hosted on a single machine. Each function is now likely to have been designed as an Internet service that runs on multiple virtualized systems. The applications themselves are very likely to be moving from one system to another to meet service-level objectives and deal with momentary outages.[[14]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-14)

**The APM conceptual framework[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=3)**]**

Applications themselves are becoming increasingly difficult to manage as they move toward highly distributed, multi-tier, multi-element constructs that in many cases rely on application development frameworks such as .NET or Java.[[15]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-15) The APM Conceptual Framework was designed to help prioritize an approach on what to focus on first for a quick implementation and overall understanding of the five-dimensional APM model. The framework slide outlines three areas of focus for each dimension and describes their potential benefits. These areas are referenced as "*Primary*" below, with the lower priority dimensions referenced as "*Secondary.* "[[16]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-16)

End user experience – (primary)**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=4)**]**

Measuring the transit of traffic from user request to data and back again is part of capturing the end-user-experience (EUE).[[17]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-17) The outcome of this measuring is referred to as Real-time Application monitoring (aka Top Down monitoring), which has two components, passive and active. [Passive monitoring](https://en.wikipedia.org/wiki/Passive_monitoring) is usually an agentless appliance implemented using [network port mirroring](https://en.wikipedia.org/wiki/Port_mirroring). A key feature to consider in this solution is the ability to support multiple protocol analytics (e.g., XML, SQL, PHP) since most companies have more than just web-based applications to support. [Active monitoring](https://en.wikipedia.org/wiki/Synthetic_monitoring), on the other hand, consists of synthetic probes and web robots predefined to report system availability and business transactions. Active monitoring is a good complement to passive monitoring; together, these two components help provide visibility into application health during off peak hours when transaction volume is low.

[](https://en.wikipedia.org/wiki/File:APM_Conceptual_Framework.jpg)

This slide outlines three *areas of focus* for each dimension and describes their potential benefits.

*User experience management* (UEM) is a subcategory that emerged from the EUE dimension to monitor the behavioral context of the user. UEM, as practiced today, goes beyond availability to capture latencies and inconsistencies as human beings interact with applications and other services.[[18]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-18) UEM is usually agent-based and may include JavaScript injection to monitor at the end user device. UEM is considered another facet of Real-time Application monitoring.#

Runtime application architecture (secondary)**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=5)**]**

Application Discovery and Dependency Mapping (ADDM) solutions exist to automate the process of mapping transactions and applications to underlying infrastructure components.[[19]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-19) When preparing to implement a runtime application architecture, it is necessary to ensure that up/down monitoring is in place for all nodes and servers within the environment (aka, bottom-up monitoring). This helps lay the foundation for event correlation and provides the basis for a general understanding on how network topologies interact with application architectures.

Business transaction (primary)**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=6)**]**

Focus on user-defined transactions or the URL page definitions that have some meaning to the business community. For example, if there are 200 to 300 unique page definitions for a given application, group them together into 8-12 high-level categories. This allows for meaningful SLA reports, and provides trending information on application performance from a business perspective: start with broad categories and refine them over time. For a deeper understanding, see[Business transaction management](https://en.wikipedia.org/wiki/Business_transaction_management).

Deep Dive Component Monitoring (secondary)**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=7)**]**

Deep Dive Component Monitoring (DDCM) requires an agent install and is generally targeted in the [middleware](https://en.wikipedia.org/wiki/Middleware) space focusing on the web, application, and messaging servers. It should provide a real-time view of the [J2EE](https://en.wikipedia.org/wiki/J2EE) and [.NET](https://en.wikipedia.org/wiki/.NET_Framework) stacks, tying them back to the user-defined business transactions. A robust solution shows a clear path from a code execution standpoint (e.g., Spring, Struts, etc.) to the URL rendered and finally to the user request. Since DDCM is closely related to the second dimension in the APM model, most products in this space also provide application discovery dependency mapping (ADDM) as part of their broader solution.

Analytics/reporting (primary)**[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=8)**]**

It is important to arrive at a common set of metrics to collect and report on for each application, then standardize on a common view on how to present the application performance data. Collecting raw data from the other tool sets across the APM model provides flexibility in application reporting. This allows for answering a wide variety of performance questions as they arise, despite the different platforms each application may be running on. Too much information is overwhelming. That is why it is important to keep reports simple or they won’t be used.[[20]](https://en.wikipedia.org/wiki/Application_performance_management#cite_note-20)

**See also[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=9)**]**

* [Application Response Measurement](https://en.wikipedia.org/wiki/Application_Response_Measurement)
* [Application service management](https://en.wikipedia.org/wiki/Application_service_management)
* [Business transaction performance](https://en.wikipedia.org/wiki/Business_transaction_performance)
* [List of performance analysis tools](https://en.wikipedia.org/wiki/List_of_performance_analysis_tools)
* [Network management](https://en.wikipedia.org/wiki/Network_management)
* [Website monitoring](https://en.wikipedia.org/wiki/Website_monitoring)
* [User Experience Rating](https://en.wikipedia.org/wiki/UXR)

**References[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=10)**]**

* 1. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-1) [*"The Anatomy of APM"*](http://apmdigest.com/the-anatomy-of-apm-4-foundational-elements-to-a-successful-strategy). APM Digest. 4 April 2012.
  2. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-2) Dubie, Denise.[*"Performance management from the client's point of view"*](http://www.networkworld.com/buzz/2006/111306-application-performance-management.html). NetworkWorld*. Retrieved 22 March 2013*.
  3. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-3) [*"APM and MoM -Symbiotic Solution Sets"*](http://www.apmdigest.com/apm-and-mom-symbiotic-solution-sets). APM Digest. 11 May 2012.
  4. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-4) [*"What You Should Know About APM – Part 1"*](http://nexus.realtimepublishers.com/content/?tip=what-you-should-know-about-application-performance-management-part-1). Realtime NEXUS. 2013.
  5. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-5) [*"Keep the Five Functional Dimensions of APM Distinct"*](http://www.gartner.com/DisplayDocument?id=1436734&ref=g_sitelink). Gartner Research (ID Number=G00206101). 16 September 2010.
  6. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-6) [*"Analytics vs. APM"*](http://www.apmdigest.com/gartner-analytics-vs-application-performance-management-1). APM Digest. 28 January 2013.
  7. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-7) [*"A Comparison of Application Performance Management Suites from CA, HP and Oracle"*](http://www.oracle.com/us/products/enterprise-manager/crimson-apm-comp-400465.pdf)(PDF). Crimson consulting group*. Retrieved 22 March 2013*.
  8. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-8) [*"Magic Quadrant for Application Performance Monitoring"*](https://www.gartner.com/doc/2639025/magic-quadrant-application-performance-monitoring). Gartner*. Retrieved 18 December 2013*.
  9. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-9) [*"Magic Quadrant for Application Performance Monitoring Suites, 2016"*](https://www.gartner.com/doc/3551918). Gartner Research (ID Number=G00298377). 21 December 2016.
  10. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-10) [*"APM Convergence: Monitoring vs. Management"*](http://apmdigest.com/apm-convergence-monitoring-vs-management). APM Digest. 6 March 2013.
  11. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-11) [*"Application Performance Management Spectrum"*](http://trac-research.com/reports/2013SpectrumAPMES.pdf)(PDF). TRAC Research. 11 March 2013.
  12. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-12) Khanna, Gunjan; Beaty, Kirk A.; Kar, Gautam; Kochut, Andrzej (2006).[*"Application Performance Management in Virtualized Server Environment"*](http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1687567&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1687567).Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP: 373–381.[*doi*](https://en.wikipedia.org/wiki/Digital_object_identifier):[*10.1109/NOMS.2006.1687567*](https://doi.org/10.1109%2FNOMS.2006.1687567)*. Retrieved 22 March 2013*.
  13. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-13) Matchett, Mike.[*"Is Virtualization Stalled On Performance?"*](http://virtualizationreview.com/articles/2013/02/20/virtualization-stalled-performance.aspx). Virtualization Review*. Retrieved 22 March 2013*.
  14. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-14) [*"Differences between approaches to APM - a chat with Jesse Rothstein of Extrahop"*](http://www.zdnet.com/blog/virtualization/differences-between-approaches-to-apm-a-chat-with-jesse-rothstein-of-extrahop/4303). ZDNet. 9 December 2011.
  15. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-15) [*"The Five Essential Elements of Application Performance Monitoring"*](http://www.realtimepublishers.com/book?id=168). Realtime NEXUS. 2010.
  16. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-16) [*"Priorizing Gartner's APM Model: The APM Conceptual Framework"*](http://www.apmdigest.com/prioritizing-gartners-apm-model). APM Digest. 15 March 2012.
  17. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-17) [*"Application performance monitoring tools: Three vendor strategies"*](http://searchnetworking.techtarget.com/tip/Application-performance-monitoring-tools-Three-vendor-strategies). SearchNetworking. 25 March 2013.
  18. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-18) [*"Insight from the User Experience Management Panel in Boston"*](http://apmdigest.com/insight-from-the-user-experience-management-panel-in-boston). APM Digest. 23 March 2012.
  19. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-19) [*"Research and Markets: Radar for Application Discovery and Dependency Mapping (ADDM)"*](http://www.businesswire.com/news/home/20110519005682/en/Research-Markets-Radar-Application-Discovery-Dependency-Mapping). Business Wire. 19 May 2011.
  20. [**Jump up^**](https://en.wikipedia.org/wiki/Application_performance_management#cite_ref-20) [*"Big Data and Advanced Analytics: Success Stories From the Front Lines"*](http://www.forbes.com/sites/mckinsey/2012/12/03/big-data-advanced-analytics-success-stories-from-the-front-lines/). Forbes. 3 December 2012.

**External links[**[**edit**](https://en.wikipedia.org/w/index.php?title=Application_performance_management&action=edit&section=11)**]**

* [The Updated List of 50+ Application Performance Management Services](https://haydenjames.io/50-top-server-monitoring-application-performance-monitoring-apm-solutions/)

[Categories](https://en.wikipedia.org/wiki/Help:Category):

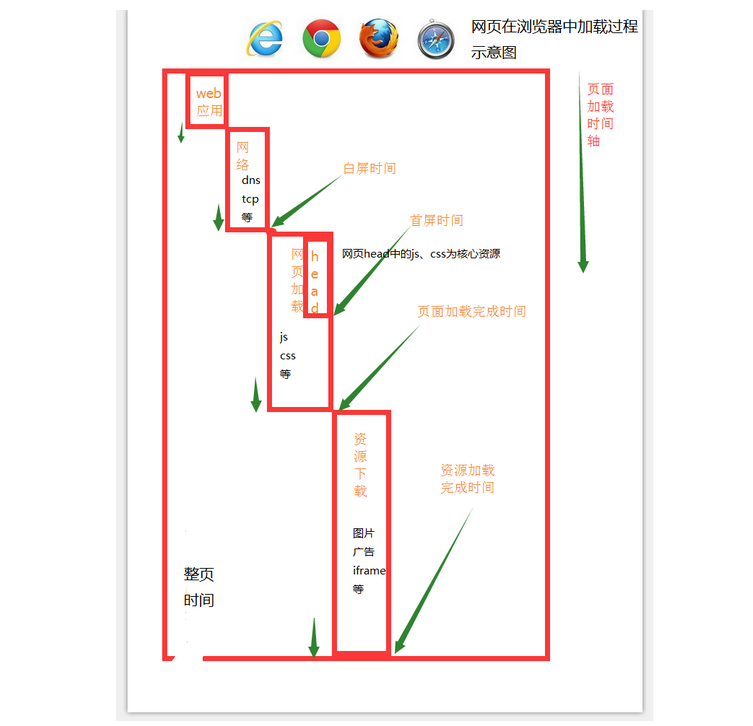
* [System administration](https://en.wikipedia.org/wiki/Category:System_administration)
* [Performance management](https://en.wikipedia.org/wiki/Category:Performance_management)

# 是谁拖了网站访问速度的「后腿」 ？ - OneAPM 博客

http://blog.oneapm.com/apm-tech/240.html

**对做前端开发的同学来说，请求排队、网络、Web 应用程序、页面加载、资源下载这些针对网站的性能指标是很熟悉的。对白屏时间、首屏时间、页面加载完成时间、资源下载完成时间以及整页时间这些性能指标也不算陌生。**

但是这10个指标分别都是什么意思？跟网站又有什么关系？为什么 TOPN 页面会惊现平均时间为1分钟？是谁在拖网站访问速度的「后腿」？本文希望能够帮助大家解决心中的困惑。



其实，性能指标不在多，找到适合自己业务的，作为判断标准才是最好的。

**用户打开一个页面的过程**

从用户在浏览器地址栏里面输入一个网址，到用户最终看到页面，在页面上可以进行各种操作，简单分为一下几个过程:

1、用户输入网址，浏览器发出请求。（主要是网络耗时）   
2、Web 服务器接到请求，处理请求，（主要是 Web 服务器耗时）   
3、Web 服务器返回数据，浏览器开始接受数据（主要是网络耗时）   
4、浏览器边接受数据边加载页面中的脚本（下载 js 脚本等耗时）   
5、浏览器下载页面中的样式、图片、视频等资源（下载图片等耗时）

[**OneAPM**](http://www.oneapm.com/)[**Browser Insight**](http://www.oneapm.com/bi/feature.html)**指标介绍**

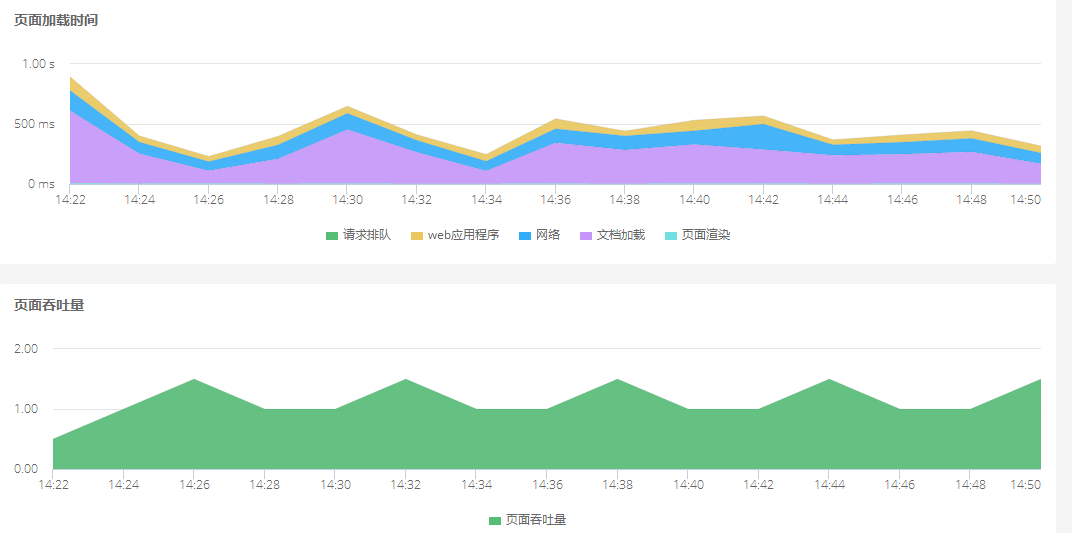
[OneAPM](http://www.oneapm.com/index.html?utm_source=Community&utm_medium=Article&utm_term=backleg&utm_campaign=NovArti&from=matefinolg) 基于浏览器端的性能监控产品 [Browser Insight](http://www.oneapm.com/bi/feature.html?utm_source=Community&utm_medium=Article&utm_term=backleg&utm_campaign=NovArti&from=matefinolg) 针对这10个指标，分别从两个方面描述了网页加载过程：

第一个方面 页面加载5阶段的耗时，用于分片运营（请求排队、网络、Web 应用程序、页面加载、资源下载）

第二个方面 页面加载过程中的5个时间点，分别对应用户感知网页加载过程中的5个关键时间点和页面状态

* 分段运营的5个指标中，请求排队和 Web 应用程序 是从 Ai 中获取到的
* 网络 = 总网络时间- Web 应用程序耗时
* 页面加载时间段=页面加载完成时间-白屏时间
* 资源下载时间段=资源下载完成时间-面加载完成时间
* 在试用的过程中，当应用程序异常时候，web应用程序面积就会增大
* 当网络异常时候，网络的面积就会增大

**平时大家看到页面加载时段或者资源下载时段面积大，是因为在一般情况下，Web 和网络都不是性能的瓶颈，性能瓶颈往往在复杂的前端页面和资源的加载过程。**

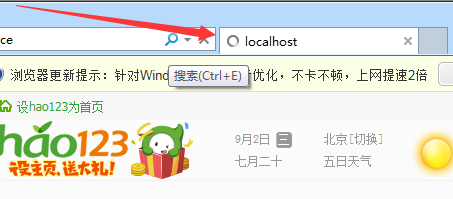


**页面加载过程中 5 个关键的时间点**

白屏时间：打开一个页面感觉屏幕从白色开始变化时刻 首屏时间：页面刚开始加载是杂乱无章，无样式的（在网速缓慢时候特别明显），当页面样式加载完毕的时候，页面就好看多了

页面加载完成时间：

* 页面加载完成时间是和业务关系最为密切的时间点，大量 js 业务逻辑都在这个时间点触发；
* 如果你发现页面上一个按钮开始无法点击，过一会可以点击了，那么可以点击的时间点就是页面加载完成时间；
* 对于移动端或者 PC 端，有很多应用是 js 控制出现一个 loading 动画，当有数据了在去掉 loading，js 控制出现 loading 的时刻就是页面加载完成时间；
* 如果页面很长，开始没有滚动条，当滚动条出现的时候，就是页面加载完成时间；
* 资源下载完成时间：可以看到页面上的图片、看到 iframe 嵌套页面里面的内容，浏览器标签不在出现loding不断旋转的时刻，就是资源加载完成的时间点；



* 整页时间：页面彻底加载完成的时间，一般情况下资源下载完成时间和整页面时间很接近。

**影响页面加载时段的主要因素**

1： 基础的网络质量   
2： 网页 Html 文件的大小 例如 你是10k，1M 10M   
3：是页面中通过 script 标签引入的脚本下载和执行，包括 head 中的也包括 body 中

因为一个浏览器同时可以可以下载的文件是有限的，同一个域名下可以同时下载的文件也是有限的

总的来说一般的浏览器为2~4 性能好点的浏览器是8个左右

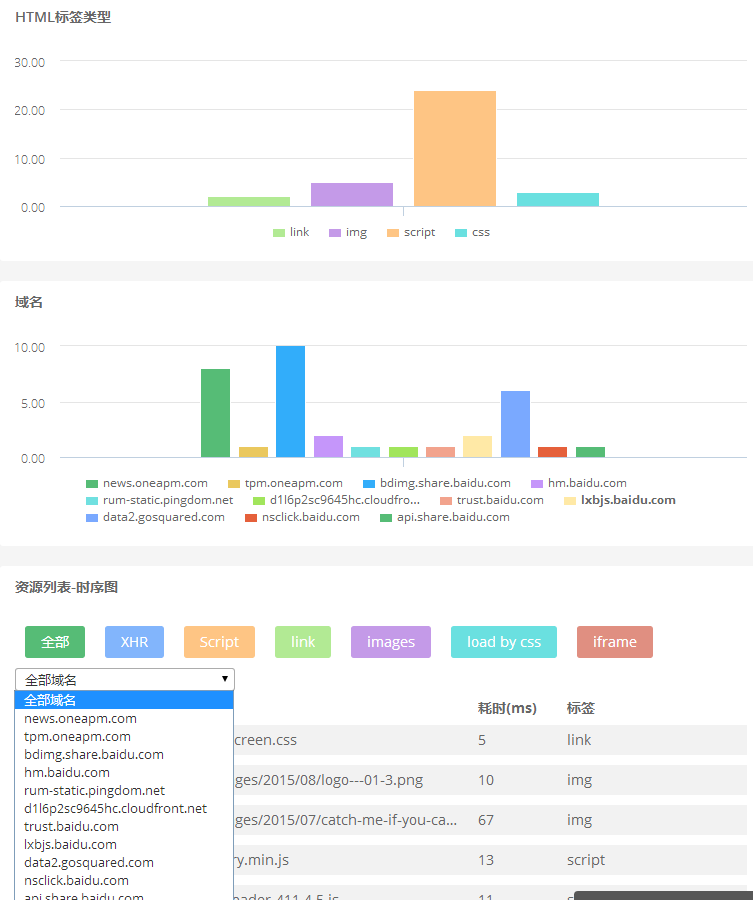
所以这个阶段网页中下载 CSS，图片等资源，都会影响页面加载时段的面积

因为css影响页面样式，所以一般放在 head 里面，公共的 js 文件放在 head 里面，业务的 js 建议通过动态加载

**影响资源下载时段的主要因素**

1：基础的网络质量   
2：页面中图片、视频、iframe 嵌套页，广告等，都在这个阶段下载   
3：下载资源个数和域名数，因为浏览器从同一个域名下同时下载资源是有限的，所以域名个数个和下载文件的个数都会影响整体性能   
所以一般都会把一个域名下载资源分散到3个左右域名下，同时下载，提高页面性能 4：第三方资源 例如百度统计、谷歌统计、 百度地图 一般都会在这个阶段加载，如果第三方资源处现在问题，资源下载时段面积也会增大

**查看那些资源的下载拖慢了整个页面的速度**



[Browser Insight](http://www.oneapm.com/bi/feature.html?utm_source=Community&utm_medium=Article&utm_term=backleg&utm_campaign=NovArti&from=matefinolg) 的 [Trace](http://blog.oneapm.com/tags-Trace.html) 会收集缓慢页面的资源加载信息，可以作为[性能优化](http://www.oneapm.com/brand/apm.html)的参考。欢迎大家访问官方网站，免费试用一下这款专业的[前端性能优化](http://www.oneapm.com/bi/feature.html)产品。

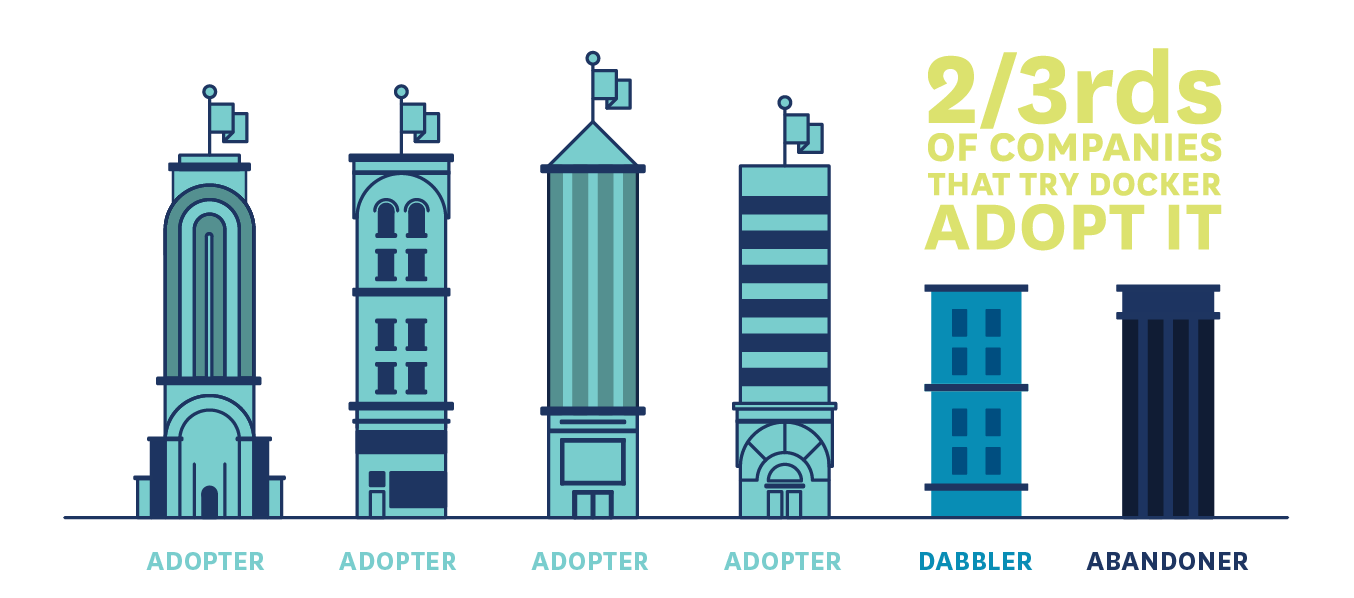
# Docker 监控实战：聚合+分组&gt;N? - OneAPM 博客

http://blog.oneapm.com/apm-tech/281.html

如今，越来越多的公司开始使用 [Docker](http://www.oneapm.com/ci/docker.html) 了，现在来给大家看几组数据：

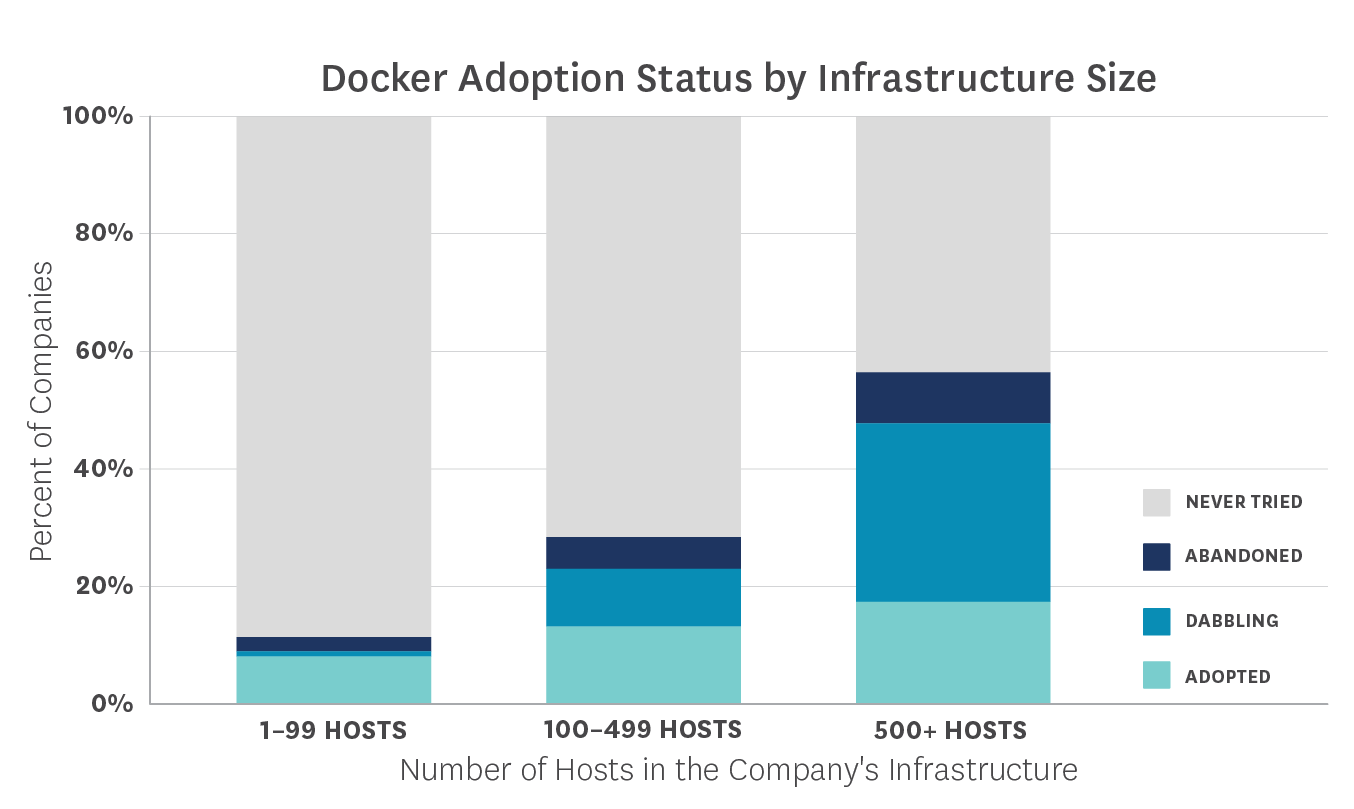
2 / 3 的公司在尝试了 Docker 后最终使用了它

也就是说 [Docker](http://news.oneapm.com/tag/docker/) 的转化率达到了 67%，而且转化时长也控制在 60 天内。



越大型的公司越早开始使用 Docker

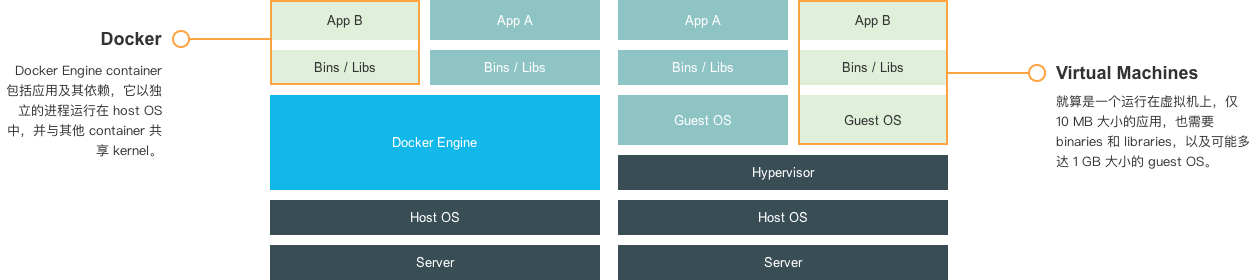
研究发现主机数量越多的公司，越早开始使用 Docker。而主机数量多，在这个研究里就默认等同于是大型公司了。



**Docker 优势**

那为什么 Docker 越来越火呢？一谈起 Docker 总是会跟着让人联想到轻量这个词，甚至会有一种通过 Docker 启动一个服务会节省很多资源的错觉。然而 Docker 的「轻」也只是相对于传统虚拟机而已。

传统虚拟机和 Docker 的对比如图：



从图中可以看出 Docker 和 虚拟机的差异，虚拟机的 Guest OS 和 Hypervisor 层在 Docker 中被 Docker Engine 层所替代，Docker 有着比虚拟机更少的抽象层。

由于 Docker 不需要通过 Hypervisor 层实现硬件资源虚拟化，运行在 Docker 容器上的程序直接使用实际物理机的硬件资源。因此在 CPU、内存利用率上 Docker 略胜一筹。

Docker 利用的是宿主机的内核，而不需要 Guest OS，因此，当新建一个容器时，Docker 不需要和虚拟机一样重新加载一个操作系统内核，因此新建一个 Docker 容器只需要几秒钟。

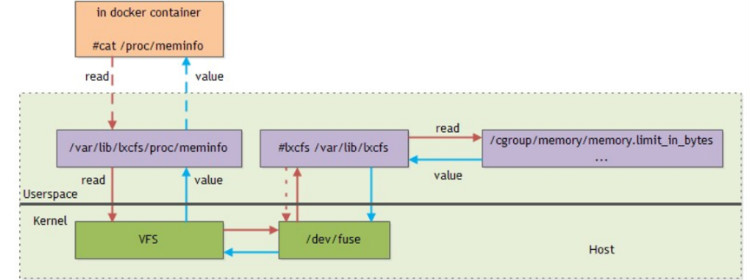
**总结一下 Docker 容器相对于 VM 有以下几个优势：启动速度快、资源利用率高、性能开销小。**

[**Docker 监控**](http://www.oneapm.com/ci/docker.html)**方案**

那么，既然 Docker 这么火，[Docker 监控](http://www.oneapm.com/ci/docker.html)是不是也该提上日程？或许具体问题要具体分析，但是似乎大家都在使用开源的监控方案，来解决 Docker监控的问题。

就拿腾讯游戏来说吧，我们看看尹烨（腾讯互娱运营部高级工程师， [干货 | 腾讯游戏是如何使用 Docker 的？](http://www.infoq.com/cn/articles/how-tencent-game-use-docker) ）怎么说：

容器的监控问题也花了我们很多精力。监控、告警是运营系统最核心的功能之一，腾讯内部有一套很成熟的监控告警平台，而且开发运维同学已经习惯这套平台，如果我们针对 Docker 容器再开发一个监控告警平台，会花费很多精力，而且没有太大的意义。所以，我们尽量去兼容公司现有的监控告警平台。每个容器内部会运行一个代理，从 /proc 下面获取 CPU、内存、IO 的信息，然后上报公司的监控告警平台。但是，默认情况下，容器内部的 proc 显示的是 Host 信息，我们需要用 Host 上 cgroup 中的统计信息来覆盖容器内部的部分 proc 信息。我们基于开源的 lxcfs，做了一些改造实现了这个需求。



这些解决方案都是基于开源系统来实现的，当然，我们也会把我们自己觉得有意义的修改回馈给社区，我们给 Docker、Kubernetes 和 lxcfs 等开源项目贡献了一些 patch。融入社区，与社区共同发展，这是一件很有意义的事情。

在没有专业运维团队来监控 Docker 的情况下，并且还想加快 Docker 监控的进程，该怎么办呢？

为了能够更精确的分配每个容器能使用的资源，我们想要实时获取容器运行时使用资源的情况，怎样对 Docker 上的应用进行监控呢？Docker 的结构会不会加大监控难度？

我们都了解， container 相当于小型 host，可以说存在于 hosts 与应用之间的监控盲区，无论是传统的[基础组件监控](http://www.oneapm.com/ci/feature.html)还是[应用性能监控](http://www.oneapm.com/)的方式，都很难有效地监控 Docker。了解了一下现有的 Docker 相关监测 App 和服务，包括简单的开源工具和复杂的企业整体解决方案，下面列举其中的几种作为参考：

**1. cAdvisor**

谷歌的 container introspection 解决方案是 cAdvisor，这是一个 Docker 容器内封装的工具，能够采集、处理和导出运行中的容器的数据。通过它可以看到 CPU 的使用率、内存使用率、网络吞吐量以及磁盘空间利用率等。同时，你还可以通过点击在网页顶部的 Docker Containers 链接，选择某个容器来详细了解它的使用情况。cAdvisor 部署和使用简单，但它只可以监视在同一个 host 上运行的容器，对多节点部署不是太管用。

**2.**[**Cloud Insight**](http://www.oneapm.com/ci/feature.html)

在我们列举的几个监控 Docker 的服务或平台中，这是唯一一款国内产品。[Cloud Insight](http://www.oneapm.com/ci/feature.html) 支持多种操作系统、云主机、数据库和中间件的监控，原理是在平台服务仪表盘和自定义仪表盘中，采集并处理 Metric，对数据进行聚合与分组等计算，提供曲线图、柱状图等多样化的展现形式。优点是监控的指标很全，简单易用，可以期待一下。

**3. Scout**

Scout 是一款监视服务，并不是一个独立的开源项目。它有大量的插件，除了 Docker 信息还可以吸收其他有关部署的数据。因此 Scout 算是一站式监控系统，无需对系统的各种资源来安装各种不同的监控系统。 Scout 的一个缺点是，它不显示有关每个主机上单独容器的详细信息。此外，每个监控的主机十美元这样略微昂贵的价格也是是否选择 Scout 作为监控服务的一个考虑因素，如果运行一个有多台主机的超大部署，成本会比较高。

**4. Sematext**

Sematext 也是一款付费监控解决方案，计划收费方案是3.5美分/小时。同样也支持 [Docker 监控](http://www.oneapm.com/ci/docker.html)，还包括对容器级事件的监测（停止、开始等等）和管理容器产生的日志。

**5. Prometheus**

Prometheus 由 SoundCloud 发明，适合于监控基于容器的基础架构。支持监控容器的资源和运行特性，支持多维度查询，能聚合 Docker 监控数据。

**Docker 监控实践**

数据的聚合&分组，是运维2.0时代的重头戏，因此我们重点选取其中比较有这个方面代表性的两个监控方案来看看具体的 Docker 监控过程。

先借鉴「Monitor Docker Containers with Prometheus](http://5pi.de/2015/01/26/monitor-docker-container」一文中的介绍，来说说这套开源的 Docker 监控方案：[*Prometheus*](https://prometheus.github.io/)；而此篇文字的原文地址：[*Monitor Docker Containers with Prometheus*](http://5pi.de/2015/01/26/monitor-docker-containers-with-prometheus/)。

Prometheus 由 SoundCloud 发明，可用于监控基于容器的基础架构。Prometheus 特点是高维度数据模型，时间序列是通过一个度量值名字和一套键值对识别。灵活的查询语言允许查询和绘制数据。它采用了先进的度量标准类型像汇总（summaries），从指定时间跨度的总数构建比率或者是在任何异常的时候报警并且没有任何依赖，中断期间使它成为一个可靠的系统进行调试。

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

可以创建一个简单的 container-exporter 来收集 Docker 容器的指标以及输出给 Prometheus 来消费。这个输出器使用容器的名字，id 和 镜像作为维度。额外的 per-exporter 维度可以在 prometheus.conf 中设置。

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

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

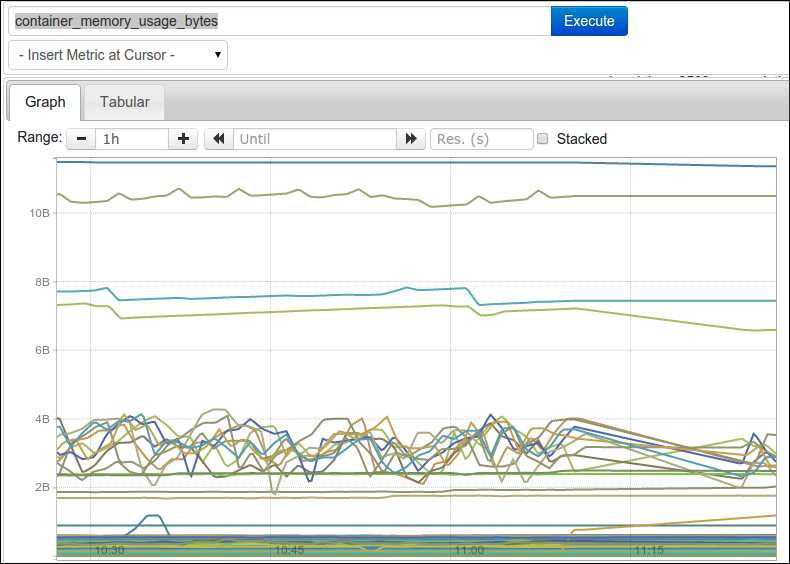
container\_memory\_usage\_bytes{env="prod",id="57690ddfd3bb954d59b2d9dcd7379b308fbe999bce057951aa3d45211c0b5f8c",instance="http://1.2.3.5:9088/metrics",job="container-exporter",name="haproxy-exporter",image="prom/haproxy-exporter:latest"} 16809984.000000

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

...

...

如果你运行了许多容器，这个看起来像这样：



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

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

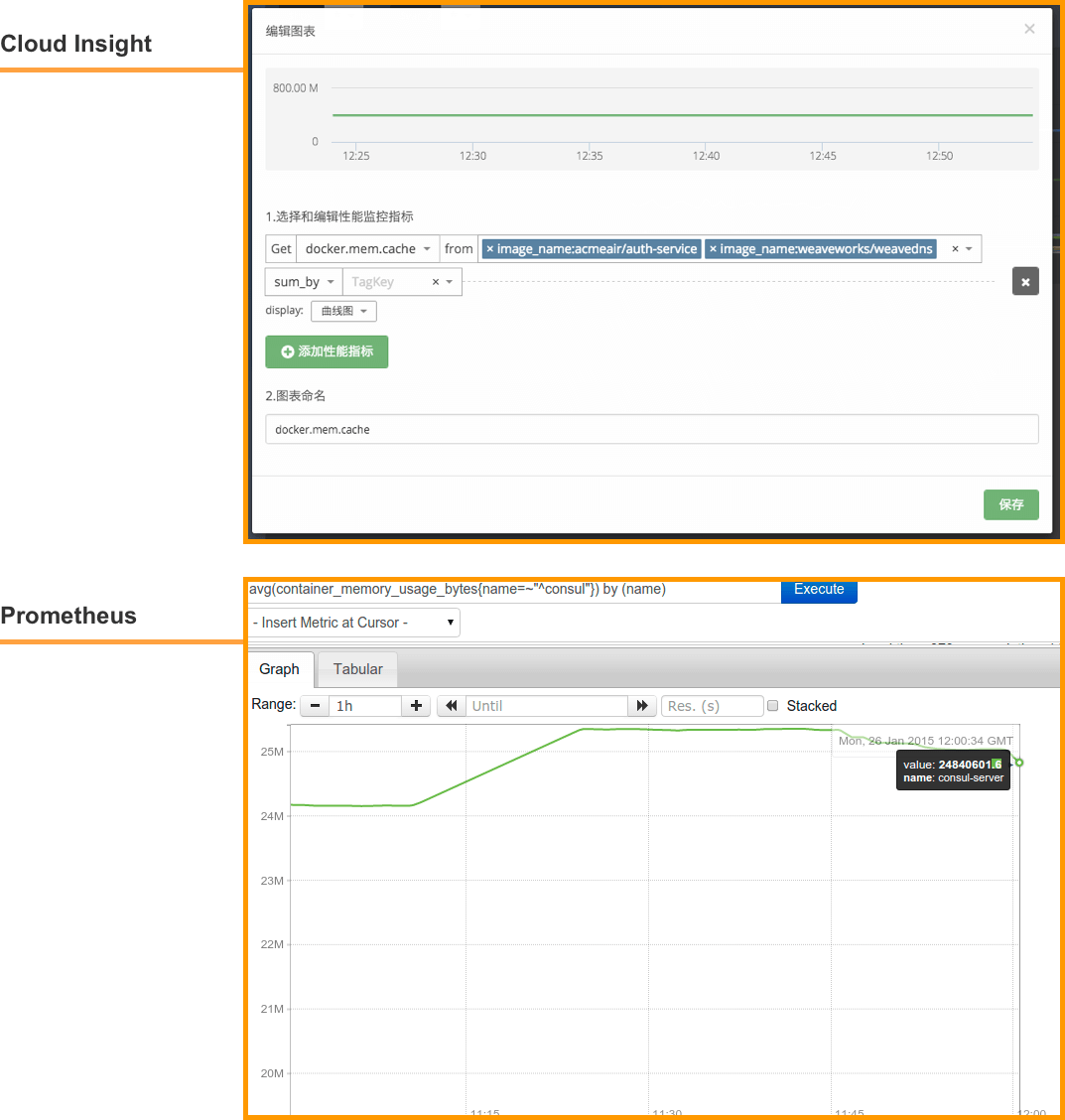
像多维度的数据模型，来实现数据聚合、分组、过滤，不单单是 Prometheus。OpenTSDB 和 InfluxDB 这些时间序列数据库和系统监控工具的结合，让系统监控这件事情变得更加的多元。

如果你是阿里云的用户，或者使用过 [Zabbix](http://www.oneapm.com/ci/zabbix.html)，那么你会明显感受到一个痛点：没有办法对数据做聚合，只能挨个查看主机的性能指标，更不用说有管理的功能。正因为如此我们才对数据聚合&分组抱有极大兴趣。如果想更多地了解数据聚合和分组的相关内容也可以阅读一下[数据聚合 & 分组：新一代系统监控的核心功能](http://news.oneapm.com/query-metric/)这篇文章。

现在我们来对比 Prometheus 和 Cloud Insight 在数据聚合、分组（切片）上的展现效果和功能。

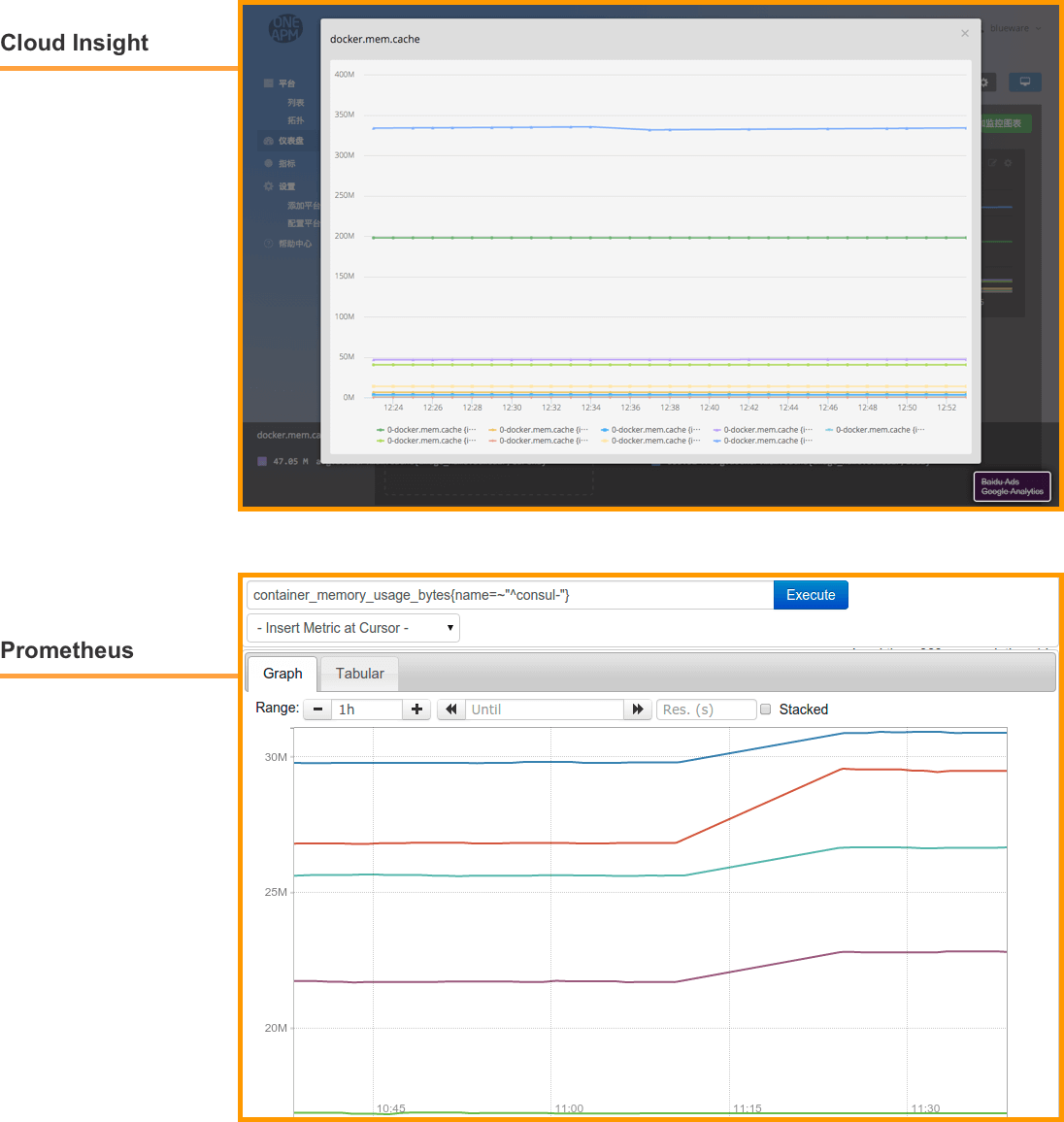
**数据聚合**

根据不同的 Container Name 或 Image Name 对内存使用量或 Memeory Cache 进行聚合。



**数据分组（切片)**

根据不同的 Container Name 或 Image Name 对内存使用量或 Memeory Cache进行分组（切片）。



单方面监控 Docker 可能并不太适合与业务挂钩的应用，当业务量上涨，不单单是 Docker 的负载上升，其他 JVM 指标也能也会出现上升的趋势。如果我们利用 Acme 模拟真实应用环境，再通过 Cloud Insight 进行监控的话，需要新建一个用于此次监控的仪表盘，依次将想要获取的指标统统添加进去。

可以添加以下指标：

docker.cpu.user

docker.cpu.sysytem

docker.containers.running

jvm.heap\_memory

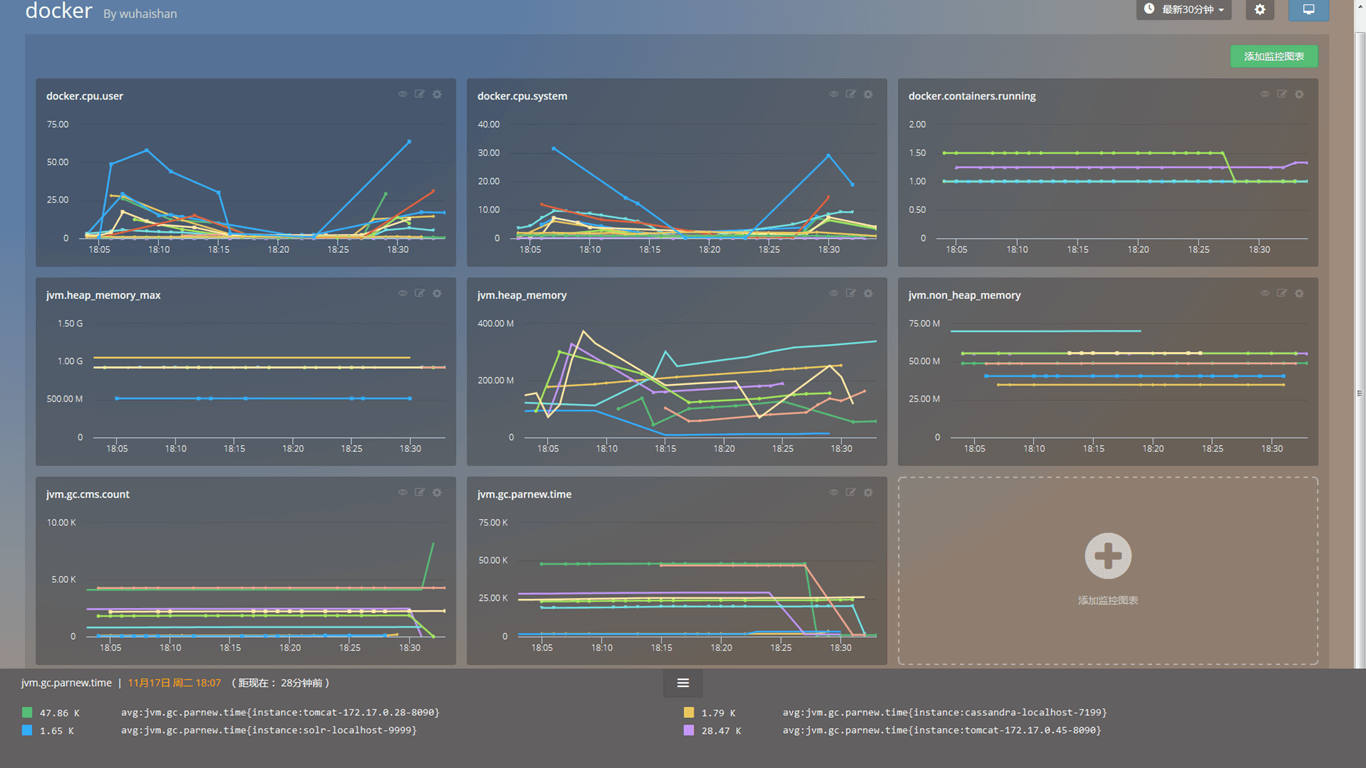
jvm.non\_heap\_memory

jvm.gc.cms.count

jvm.heap\_memory\_max

jvm.gc.parnew.time

应用 Acme 部署在四台 servers 上，我们开启四台 servers, 然后用 JMeter 给应用加压。随着时间 JMeter 不断给应用加压，当 users 人数达到 188 时，我们来看一下仪表盘的视图。



根据 JMeter 里的数据，CPU 占用和错误率都有所提升；与此同时，在指标 docker.cpu.user 这幅图中，蓝色的线所代表的 Container CPU 占用率已经超过 50%，逐渐接近 75%，系统剩余的 CPU 资源逐渐下降。

而指标 docker.cpu.system 图中同样可以看到蓝色的那条数据在 18：29 左右出现了一个波峰，代表系统 CPU 资源消耗突然增大。通过这两幅图，我们可以定位到 CPU 占用率过高的 Container ，及时而主动地去了解性能瓶颈，从而优化性能，合理分配资源。

用 jvm.heap\_memory 的值去比左图 jvm.heap\_memory\_max 的值，将能清楚的反映 JVM 堆内存的消耗情况。而 jvm.gc.parnew.time 图中显示了新生代并行 GC 的时间数据。GC 是需要时间和资源的，不好的 GC 会严重影响系统的系能，良好的 GC 是 JVM 高性能的保证。

无法被监控的软件是很危险的，通过解读这张 Docker 仪表盘总览图，我们也许可以了解到 Docker 实时性能状况，精准定位到性能薄弱的环节，从而优化我们的应用。

**总结**

Docker 兼容相比其他的数据库、系统、中间件监控，要复杂一些。由于需要表现不同 Container 的性能消耗，来了解不同应用的运行情况，所以数据的聚合、切片（分组）和过滤，在 Docker 监控中成为了必备功能。

所以推荐大家使用时间序列数据库，或者类似设计逻辑的监控方案，如：Prometheus 和 Cloud Insight。而 Docker 单方面的监控，可能不太满足一些大型公司的需求，如果一个工具在监控 Docker 同时能够监控其他组件，那就更好了。国外的 Graphite、Grafana 和 Host Graphite，以及国内的 Cloud Insight，都是能够让用户将不同数据来源都集中在同一个地方进行展现，大家页可以对比和试用一下。

参考文章：

[Monitor Docker Containers with Prometheus](http://5pi.de/2015/01/26/monitor-docker-containers-with-prometheus/)

[8 Surprising Fact about Real Docker](https://www.datadoghq.com/docker-adoption/)

[六个下一代 Docker 监控工具](http://blog.tenxcloud.com/?p=496)

# 六大下一代Docker监测工具 – 时速云技术博客

http://blog.tenxcloud.com/?p=496

容器：这里面什么才是最重要的？container monitoring，一项可以帮助你了解容器内发生什么以及让你获得某种支持的技术，随着Docker自身的monitoring 和 introspection技术API的成熟，已经有最新一波的容器监测技术利用到它们。

以下列举6个Docker相关的监测App和服务，从简单的开源工具到复杂的企业整体解决方案。

**cAdvisor**

谷歌的container introspection解决方案是cAdvisor，其本身也是在Docker容器内封装的实用工具，它能够快速的产生，运行容器的基本行为等有用信息。它部署和使用很简单，但是它只可以监视在同一个host上运行的其他容器，所以它对多节点部署不是太管用。

**Prometheus.io**

Soundcloud实验室出品，Prometheus.io是一个系统监控和报警的开源框架。它的强大之处在于它可以摄取来自一个巨大范围数据源的数据包括容器。更厉害的是，容器的数据来源并不局限cAdvisor、Kubernetes、CoreOS的Etcd。

**Scout**

Scout是一款监视服务，它并不是一个独立的开源项目。但是它比单独的cAdvisor涵盖了更多的领域——换句话说，它从跨多主机聚合数据，并且它有一个插件架构允许更广范围的集成。但它也是一个付费服务，10美元每主机。所以那些寻求一个完全免费的解决方案的开发者一定会失望。

**DataDog**

同样作为“container monitoring as a service”的是 DataDog。15美元每主机，相比Scout拥有更详细的和灵活的报表功能。

**Logentries**

许多Docker-monitoring的服务都是长时间的系统监测maven。Logentries是其中之一，该公司似乎已经感觉到这波容器浪潮。它们最近推出了一个免费的Docker容器日志服务，30天试用版本甚至还包括Logentries的其他付费服务，但容器日志服务本身仍然是免费的。小提醒：如果你运行Docker在一个受限制的环境中，例如Google Compute Engine，你需要运行在特许模式下运行Logentries容器。

**Sematext**

Sematext是一款云端on-prem多个集成监控解决方案，Sematext最近添加Docker监测在其SPM解决方案中。像其他监测方案一样，该监控是通过一个容器管理，通过事件回滚到各种各样的软件源，一个代理可以用来监视多个服务器。还包括对容器级事件的监测（停止、开始等等）和管理容器产生的日志。计划收费方案是3.5美分每服务小时。

本文由丁麒伟编译整理，转载请注明出自时速云。原文链接：http://www.infoworld.com/article/2976930/application-virtualization/6-monitoring-tools-docker-containers.html

# 2@Docker 监控- Prometheus VS Cloud Insight - OneAPM 博客

http://blog.oneapm.com/apm-tech/259.html

**如今，越来越多的公司开始使用**[**Docker**](http://www.oneapm.com/ci/docker.html)**了，2 / 3 的公司在尝试了 Docker 后最终使用了它。为了能够更精确的分配每个容器能使用的资源，我们想要实时获取容器运行时使用资源的情况，怎样对 Docker 上的应用进行监控呢？Docker 的结构会不会加大监控难度？**

可是在没有专业运维团队来监控 Docker 的情况下，并且还想加快 [Docker 监控](http://www.oneapm.com/ci/docker.html)的日程，怎么办呢？

我们通过调查了解到几种不错的 [Docker 监控](http://www.oneapm.com/ci/docker.html)方法，其中 Prometheus 和 [Cloud Insight](http://www.oneapm.com/ci/feature.html) 让人很感兴趣。

**Prometheis**

先来说说一套开源的 Docker 监控方案：[Prometheus](http://prometheus.io/)；而此篇文字的原文地址：[Monitor Docker Containers with Prometheus](http://5pi.de/2015/01/26/monitor-docker-containers-with-prometheus/)。

Prometheus 由 SoundCloud 发明，适合于监控基于容器的基础架构。Prometheus 特点是高维度数据模型，时间序列是通过一个度量值名字和一套键值对识别。灵活的查询语言允许查询和绘制数据。它采用了先进的度量标准类型像汇总（summaries），从指定时间跨度的总数构建比率或者是在任何异常的时候报警并且没有任何依赖，中断期间使它成为一个可靠的系统进行调试。

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

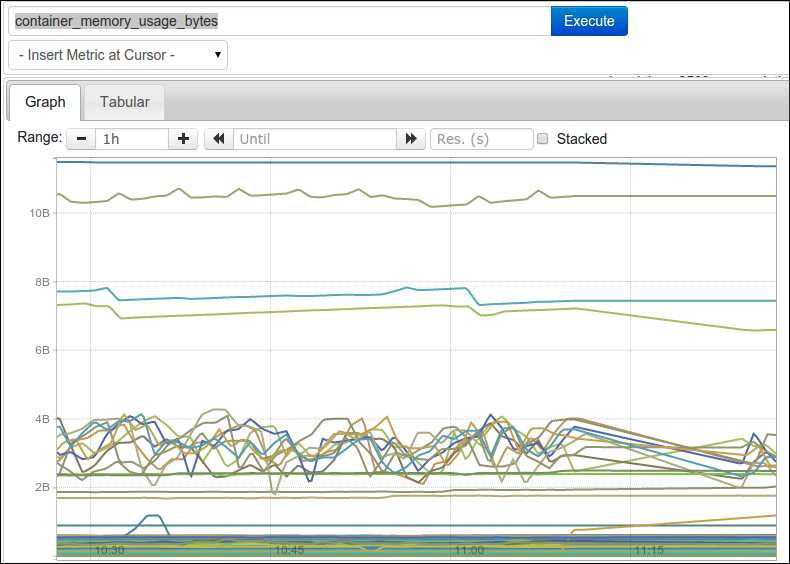
我已经创建了一个简单的 container-exporter 来收集 [Docker 容器](http://news.oneapm.com/tag/docker/)的指标以及输出给 Prometheus 来消费。这个输出器使用容器的名字，id 和 镜像作为维度。额外的 per-exporter 维度可以在prometheus.conf 中设置。

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

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

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

container\_memory\_usage\_bytes{env="prod",id="907ac267ebb3299af08a276e4ea6fd7bf3cb26632889d9394900adc832a302b4",instance="http://1.2.3.2:9088/metrics",job="container-exporter",name="node-exporter",image="prom/container-exporter:latest"} ... ... 如果你运行了许多容器，这个看起来像这样：



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

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

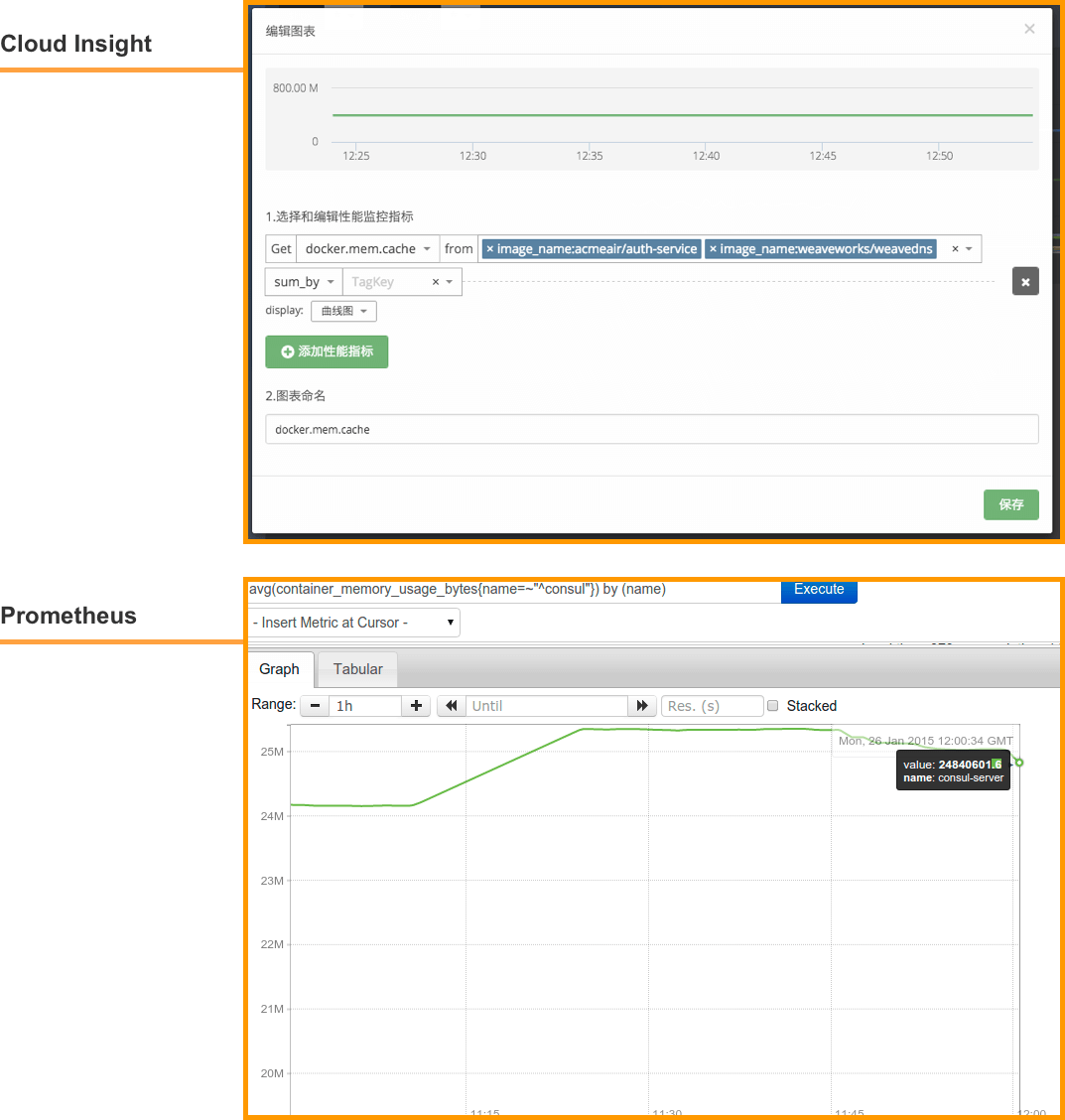
像多维度的数据模型，来实现数据聚合、分组、过滤，不单单是 Prometheus。OpenTSDB 和 InfluxDB 这些时间序列数据库和系统监控工具的结合，让系统监控这件事情变得更加的多元。

接下来，我们为大家介绍国内一家同样提供该功能的监控方案：[Cloud Insight](http://www.oneapm.com/ci/feature.html?utm_source=Community&utm_medium=Article&utm_term=201512DockerPrometheus&utm_campaign=CiArti&from=jscwftdq)。有关其数据聚合的功能可以阅读：[数据聚合 & 分组：新一代系统监控的核心功能](http://news.oneapm.com/query-metric/)。

现在我们来对比 Prometheus 和 Cloud Insight 在数据聚合、分组（切片）上的展现效果和功能。

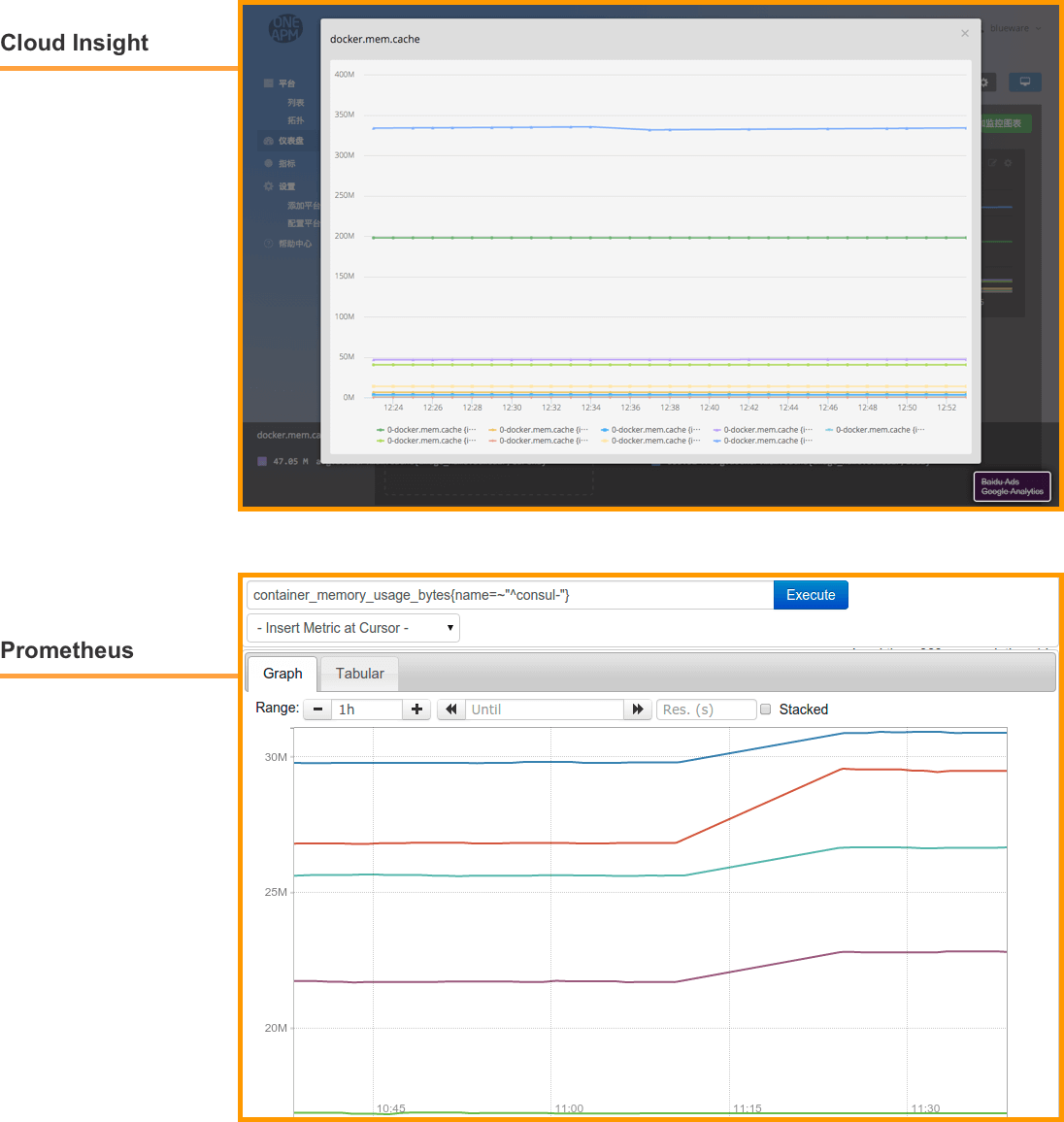
**数据聚合**

根据不同的 Container Name 或 Image Name 对内存使用量或 Memeory Cache 进行聚合。



**数据分组（切片）**

根据不同的 Container Name 或 Image Name 对内存使用量或 Memeory Cache进行分组（切片）。



**Cloud Insight**

[Cloud Insight](http://www.oneapm.com/ci/feature.html?utm_source=Community&utm_medium=Article&utm_term=201512DockerPrometheus&utm_campaign=CiArti&from=jscwftdq) 支持多种操作系统、云主机、数据库和中间件的监控，原理是在平台服务仪表盘和自定义仪表盘中，采集并处理 Metric，对数据进行聚合与分组等计算，提供曲线图、柱状图等多样化的展现形式。优点是监控的指标很全，简单易用，也可以期待一下。

**Cloud Insight 监控 Docker 试验**

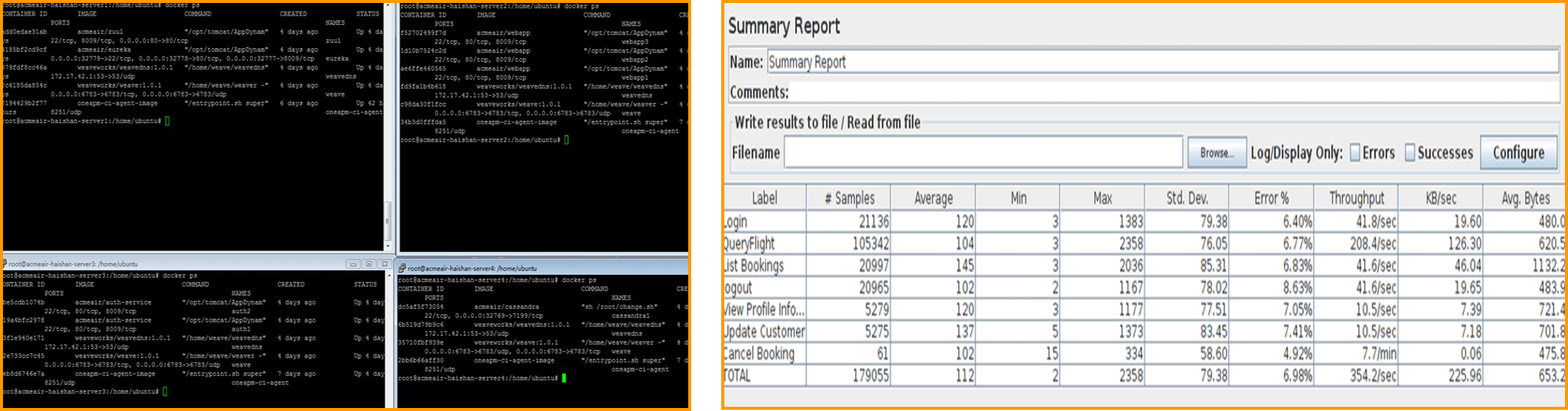
我们用 AcmeAir 作为试验的应用，AcmeAir 是一款由原 IBM 新技术架构部资深工程师 Andrew Spyker，利用 Netflix 开源的 Netflix OSS 打造的开源电子商务应用。

首先，我们要打开 Cloud Insight 监控，还好 [Cloud Insight](http://www.oneapm.com/ci/feature.html?utm_source=Community&utm_medium=Article&utm_term=201512DockerPrometheus&utm_campaign=CiArti&from=jscwftdq) 安装简单，一条命令即可。接着，我们新建一个用于此次监控的仪表盘，依次将想要获取的指标统统添加进去。

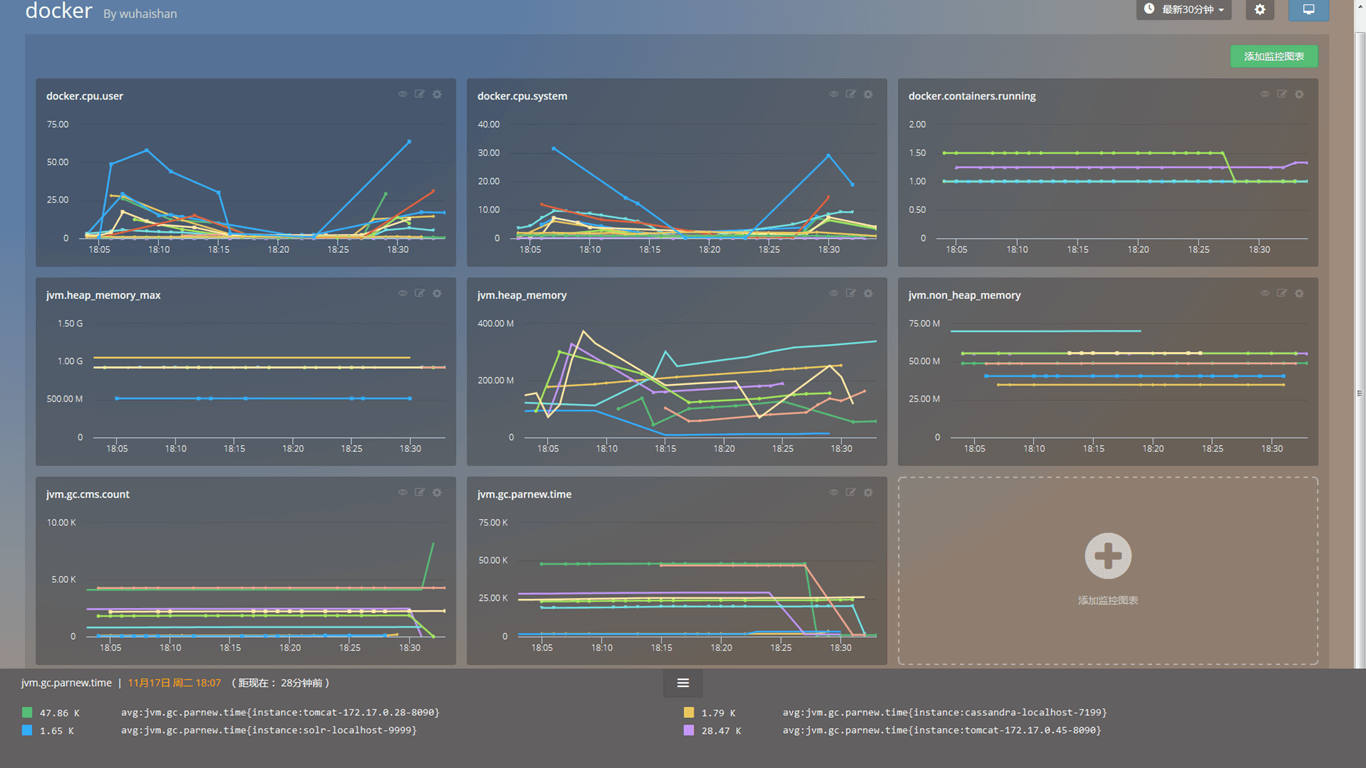
我们添加了以下指标：

* docker.cpu.user
* docker.cpu.sysytem
* docker.containers.running
* jvm.heap\_memory
* jvm.nonheapmemory
* jvm.gc.cms.count
* jvm.heapmemorymax
* jvm.gc.parnew.time

应用 Acme 部署在四台 servers 上，我们开启四台 servers, 然后用 JMeter 给应用加压。



随着时间 JMeter 不断给应用加压，当 users 人数达到188时，我们再来看一下仪表盘的视图。



从图中可以看到，性能数据发生了变化，根据 JMeter 里的数据，此时 CPU 占用超过了50%，错误率也有所提升；对比来看，根据 Cloud Insight 里的曲线显示，蓝色的线所代表的 Container CPU 占用率已经超过50%，逐渐接近75%，系统剩余的 CPU 资源逐渐下降，该 Container 的系统 CPU 资源消耗也突然增大。我们可以通过这些定位到 CPU 占用率过高的 Container ，及时而主动地去了解性能瓶颈，从而优化性能，合理分配资源。[Cloud Insight](http://www.oneapm.com/ci/feature.html?utm_source=Community&utm_medium=Article&utm_term=201512DockerPrometheus&utm_campaign=CiArti&from=jscwftdq) 所抓取的性能指标算是较为全面，部署和展现方式都是相当简单易懂的。

**总结**

Docker 兼容相比其他的数据库、系统、中间件监控，要复杂一些。由于需要表征不同 Container 的性能消耗，来了解不同应用的运行情况，所以数据的聚合、切片（分组）和过滤，在 Docker 监控中成为了必备功能。

所以我们推荐使用了时间序列数据库，或者类似设计逻辑的监控方案，如：Prometheus 和 Cloud Insight。

而 Docker 单方面的监控，可能不太满足一些大型公司的需求，如果一个工具在监控 Docker 同时能够监控其他组件，那就更好了。

国外出现了 Graphite、Grafana 和 Host Graphite，能够让用户将不同数据来源都集中在同一个地方进行展现；而国内 [Cloud Insight](http://www.oneapm.com/ci/feature.html?utm_source=Community&utm_medium=Article&utm_term=201512DockerPrometheus&utm_campaign=CiArti&from=jscwftdq) 似乎也是这样的思路。

# 关于OneAPM | OneAPM是什么

- 北京蓝海讯通科技股份有限公司

http://www.oneapm.com/info/about.html

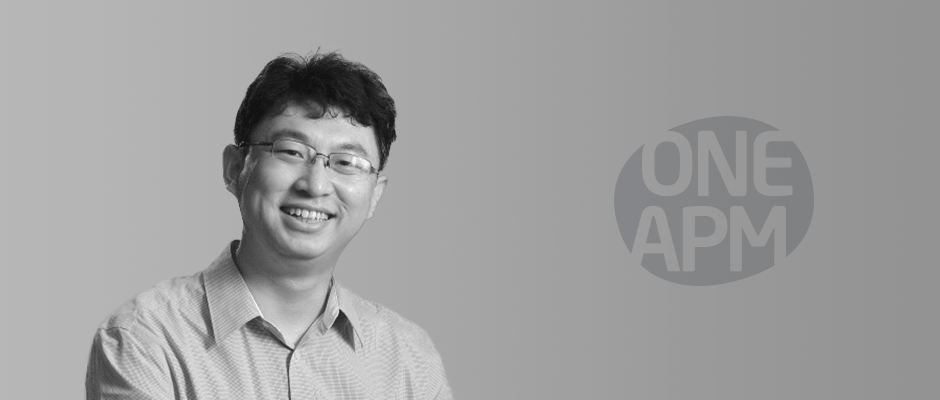
OneAPM，即北京蓝海讯通科技股份有限公司，是中国基础软件领域的新兴领军企业。专注于提供新一代 ITOM（IT 运维管理）软件和服务。OneAPM 一直秉承「为客户创造价值」的经营理念，2016年8月15日，正式挂牌新三板（股票代码 838699）。

OneAPM 是全球首家可以同时从系统服务层、应用层、用户体验层、业务交易层提供性能管理服务的公司。经过8年的技术与产品积累与沉淀，已经能够提供本地化部署和 SaaS 部署模式，支持所有主流的编程语言和框架。

目前，OneAPM 拥有 5 条产品线，包括应用性能管理（Application Insight、Browser Insight，Mobile Insight、Cloud Test），IT 事件管理及关联分析（Cloud Insight、OneAlert、.Log Insight），数据中心管理（Infrastructure Insight、Network Insight），性能压测（Cloud Performance Test）以及应用安全管理平台（OneASP），能够为企业 IT 系统的高效、稳定、安全运行提供全方位的支持和保障服务。

OneAPM 目前拥有约 450 名员工，核心研发团队由美国硅谷资深研发人员组成，总部位于中国北京，同时在上海、广州、成都、西安、沈阳等各大城市设有分公司，目前公司业务已经拓展至台湾、日本、韩国、东南亚等国家和地区。

当前，全球有数十万开发者选择了OneAPM，其中企业级用户超过 50000 家，全面覆盖了运营商、金融、政府、交通、能源、电商、O2O、旅游、教育、游戏等多个领域。



何晓阳，OneAPM 创始人。对字节码和类装载技术有持续深入研究，著有「Java 虚拟机技术与应用性能管理实战」等三本专业书籍，被誉为「中国 APM 行业第一人」。



「何晓阳•读书笔记」是 OneAPM 董事长：何晓阳先生的随笔，记录对科技、创业的观点，以及读书的心得。  
[「中国 SaaS 创业者，你需要种更多的土豆」](http://news.oneapm.com/start-up-as-farmer/)

# Comparing Seven Monitoring Options for Docker | Rancher Labs

http://rancher.com/comparing-monitoring-options-for-docker-deployments/

As Docker is used for larger deployments it becomes more important to get visibility into the status and health of docker environments. In this article I aim to go over some of the common tools used to monitor containers. I will be evaluating these tools based on the following criteria: 1) ease of deployment, 2) level of detail of information presented, 3) level of aggregation of information from entire deployment, 4) ability to raise alerts from the data and 5)  Ability to monitor non-docker resources 6) cost. This list is by no means comprehensive however I have tried to highlight the most common tools and tools that optimize our six evaluation criteria.

**Docker Stats**

All commands in this article have been specifically tested on a RancherOS instance running on Amazon Web Services EC2. However, all tools presented today should be usable on any Docker deployment.

The first tool I will talk about is Docker itself, yes you may not be aware that docker client already provides a rudimentary command line tool to inspect containers’ resource consumption. To look at the container stats run docker statswith the name(s) of the running container(s) for which you would like to see stats. This will present the CPU utilization for each container, the memory used and total memory available to the container. Note that if you have not limited memory for containers this command will post total memory of your host. This does not mean each of your container has access to that much memory. In addition you will also be able to see total data sent and received over the network by the container.

$ docker stats determined\_shockley determined\_wozniak prickly\_hypatia

CONTAINER             CPU %               MEM USAGE/LIMIT       MEM %               NET I/O

determined\_shockley   0.00%               884 KiB/1.961 GiB     0.04%               648 B/648 B

determined\_wozniak    0.00%               1.723 MiB/1.961 GiB   0.09%               1.266 KiB/648 B

prickly\_hypatia       0.00%               740 KiB/1.961 GiB     0.04%               1.898 KiB/648 B

For a more detailed look at container stats you may also use the Docker Remote API via netcat (See below). Send an http get request for/containers/[CONTAINER\_NAME]/statswhere CONTAINER\_NAME is name of the container for which you want to see stats. You can see an example of the complete response for a container stats request [here](https://gist.github.com/usmanismail/0c4922ffec4a0220d385). This will present details of the metrics shown above for example you will get details of caches, swap space and other details about memory. You may want to peruse  the [Run Metrics](https://docs.docker.com/articles/runmetrics/%20) section of the Docker documentation to get an idea of what the metrics mean.

echo -e "GET /containers/[CONTAINER\_NAME]/stats HTTP/1.0\r\n" | nc -U /var/run/docker.sock

 Score Card:

1. Easy of deployment: \*\*\*\*\*
2. Level of detail: \*\*\*\*\*
3. Level of aggregation: none
4. Ability to raise alerts: none
5. Ability to monitor non-docker resources: none
6. Cost: Free

**CAdvisor**

The docker stats command and the remote API are useful for getting information on the command line, however, if you would like to access the information in a graphical interface you will need a tool such as [CAdvisor](https://github.com/google/cadvisor). CAdvisor provides a visual representation of the data shown by the docker stats command earlier.  Run the docker command below and go to http://<your-hostname>:8080/in the browser of your choice to see the CAdvisor interface. You will be shown graphs for overall CPU usage, Memory usage, Network throughput and disk space utilization. You can then drill down into the usage statistics for a specific container by clicking the Docker Containers link at the top of the page and then selecting the container of your choice.  In addition to these statistics CAdvisor also shows the limits, if any, that are placed on container, using the Isolation section.

docker run \

--volume=/:/rootfs:ro \

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

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

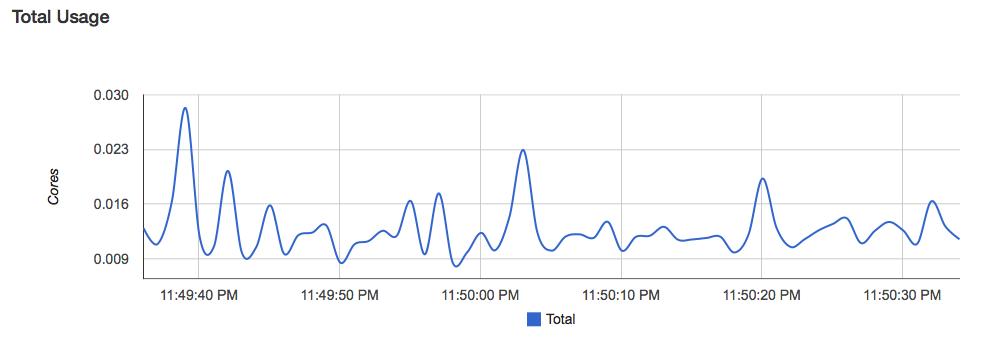
--volume=/var/lib/docker/:/var/lib/docker:ro \

--publish=8080:8080 \

--detach=true \

--name=cadvisor \

google/cadvisor:latest

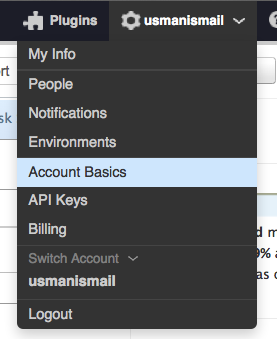
[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-19-at-11.50.29-PM.png)

CAdvisor is a useful tool that is trivially easy to setup, it saves us from having to ssh into the server to look at resource consumption and also produces graphs for us. In addition the pressure gauges provide a quick overview of when a your cluster needs additional resources. Furthermore, unlike other options in this article CAdvisor is free as it is open source and also it runs on hardware already provisioned for your cluster, other than some processing resources there is no additional cost of running CAdvisor. However, it has it limitations; it can only monitor one docker host and hence if you have a multi-node deployment  your stats will be disjoint and spread though out your cluster. Note that you can use[heapster](https://github.com/GoogleCloudPlatform/heapster) to monitor multiple nodes if you are running Kubernetes.  The data in the charts is a moving window of one minute only and there is no way to look at longer term trends. There is no mechanism to kick-off alerting if the resource usage is at dangerous levels. If you currently do not have any visibility in to the resource consumption of your docker node/cluster then CAdvisor is a good first step into container monitoring however, if you intend to run any critical tasks on your containers a more robust tool or approach is needed.  Note that[Rancher](http://rancher.com/rancher-io/) runs CAdvisor on each connected host, and exposes a limited set of stats through the UI, and all of the system stats through the API.

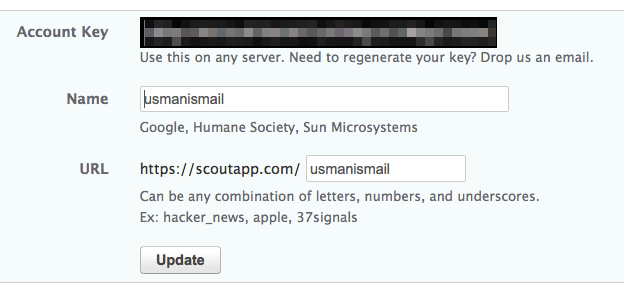
 Score Card (Ignoring heapster because only supported on Kubernetes):

1. Easy of deployment: \*\*\*\*\*
2. Level of detail: \*\*
3. Level of aggregation: \*
4. Ability to raise alerts: none
5. Ability to monitor non-docker resources: none
6. Cost: Free

**Scout**

**[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-21-at-9.30.08-AM.png)**

The next approach for docker monitoring is Scout and it addresses several of limitations of CAdvisor. Scout is a hosted monitoring service which can aggregate metrics from many hosts and containers and present the data over longer time-scales. It can also create alerts based on those metrics. The first step  to getting scout running is to sign up for a Scout account at <https://scoutapp.com/>, the free trial account should be suitable for testing out integration.  Once you have created your account and logged in, click on your account name in the top right corner and then Account Basicsand take note of your Account Key as you will need this to send metrics from our docker server.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/accountid.png)Now on your host, create a file called scoutd.yml and copy the following text into the the file, replacing the account\_key with the key you took note of earlier. You can specify any values that make sense for the host, display\_name, environment and roles properties. These will be used to separate out the metrics when they are presented in the scout dashboard. I am assuming an array of web-servers is run on docker so will use the values shown below.

# account\_key is the only required value

account\_key: YOUR\_ACCOUNT\_KEY

hostname: web01-host

display\_name: web01

environment: production

roles: web

You can now bring up your scout agent with the scout configuration file by using the docker scout plugin.

docker run -d  --name scout-agent                              \

    -v /proc:/host/proc:ro                                     \

    -v /etc/mtab:/host/etc/mtab:ro                             \

    -v /var/run/docker.sock:/host/var/run/docker.sock:ro      \

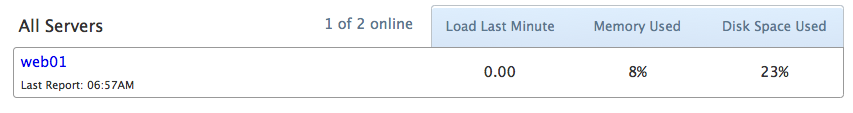
    -v `pwd`/scoutd.yml:/etc/scout/scoutd.yml                  \

-v /sys/fs/cgroup/:/host/sys/fs/cgroup/ \

    --net=host --privileged                                  \

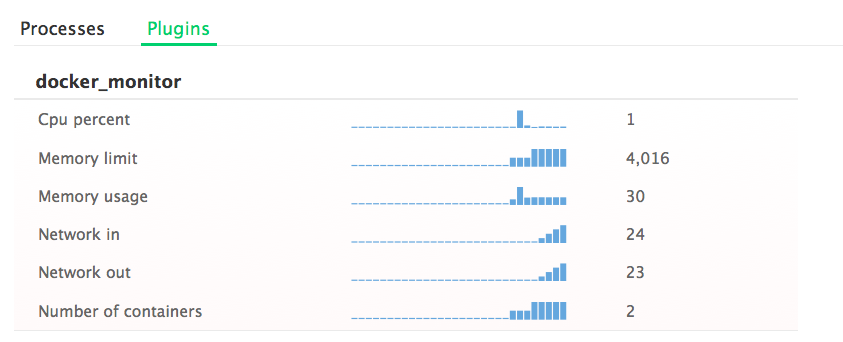
    soutapp/docker-scout

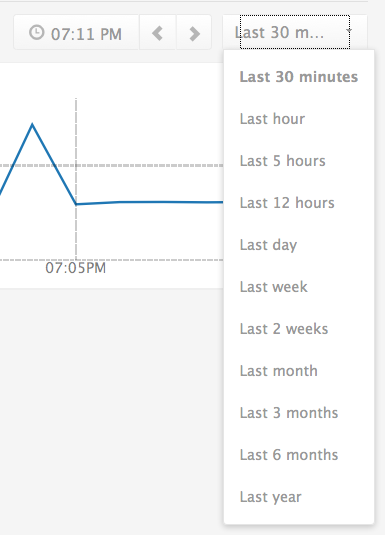
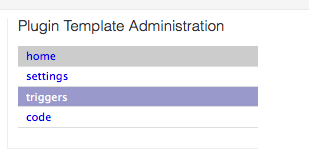
Now go back to the Scout web view and you should see an entry for your agent which will be keyed by the display\_name parameter (web01) that you specified in your scoutd.yml earlier.

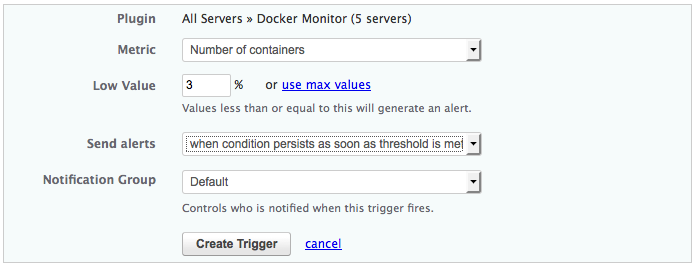
[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-21-at-9.58.40-AM.png)If you click the display name it will display detailed metrics for the host. This includes the process count, CPU usage and memory utilization for everything running on your host. Note these are not limited to processes running inside docker.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-21-at-10.00.47-AM.png)

To add docker monitoring to your servers click the Roles tab and then select All Servers.Now click the + Plugin Template Button and then  Docker Monitor from the following screen to load the details view. Once you have the details view up select Install Plugin to add the plugin to your hosts. In the following screen give a name to the plugin installation and specify which containers you want to monitor. If you leave the field blank the plugin will monitor all of the containers on the host.  Click complete installation and after a minute or so you can go to [Server Name] > Plugins to see details from the docker monitor plugin. The plugin shows the CPU usage, memory usage network throughput and the number of containers for each host.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-20-at-10.11.06-PM.png)

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-20-at-10.11.39-PM.png)If you click on any of the graphs you can pull a detailed view of the metrics and this view allows you to see the trends in the metric values across a longer time span. This view also allows you to filter the metrics based on environment and server role. In addition you can create “Triggers” or alerts to send emails to you if metrics go above or below a configured threshold. This allows you to setup automated alerts to notify you if for example some of your containers die and the container count falls below a certain number. You can also setup alerts for average CPU utilization so if for example if your containers are running hot you will get an alert and you can launch more add more hosts to your docker cluster. To create a trigger select[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-22-at-6.30.25-PM.png)Roles > All Servers from the top menu and then docker monitor from the plugins section. Then select triggersfrom thePlugin template Administration menu on the right hand side of the screen.  You should now see an option to “Add a Trigger” which will apply to the entire deployment.  Below is an example of a trigger which will send out an alert if the number of containers in the deployment falls below 3. The alert was created for “All Servers” however you could tag your hosts with different roles using the scoutd.yml created on the server. Using the roles you can apply triggers to a sub-set of the servers on your deployment. For example you could setup an alert for when the number of containers on your web nodes falls below a certain number. Even with the role based triggers I still feel that Scout alerting could be better. This is because many docker deployments have heterogeneous containers on the same host. In such a scenario it would be impossible to setup triggers for specific types of containers as roles are applied to all containers on the host.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-22-at-6.33.12-PM.png)

Another advantage of using Scout over CAdvisor is that it has a [large set of plugins](https://scoutapp.com/plugin_urls) which can pull in other data about your deployment in addition to docker information. This allows Scout to be your one stop monitoring system instead of having a different monitoring system for various resources in your system.

One drawback of Scout is that it does not present detailed information about individual containers on each host like CAdvisor can. This is problematic, if your are running heterogeneous containers on the same server. For example if you want a trigger to alert you about issues in your web containers but not about your Jenkins containers Scout will not be able to support that use case. Despite the drawbacks Scout is a significantly more useful tool for monitoring your docker deployments. However this does come at a cost, ten dollars per monitored host. The cost could be a factor if you are running a large deployment with many hosts.

 Score Card:

1. Easy of deployment: \*\*\*\*
2. Level of detail: \*\*
3. Level of aggregation: \*\*\*
4. Ability to raise alerts: \*\*\*
5. Ability to monitor non-docker resources: Supported
6. Cost: $10 / host

**Data Dog**

From Scout lets move to another monitoring service, DataDog, which addresses several of the short-comings of Scout as well as all of the limitations of CAdvisor. To get started with DataDog, first sign up for a DataDog account at<https://www.datadoghq.com/>. Once you are signed into your account you will be presented with list of supported integrations with instructions for each type.  Select docker from the list and you will be given a docker run command (show below) to copy into your host. The command will have your API key preconfigured and hence can be run the command as listed. After about 45 seconds your agent will start reporting metrics to the DataDog system.

docker run -d --privileged --name dd-agent \

-h `hostname` \

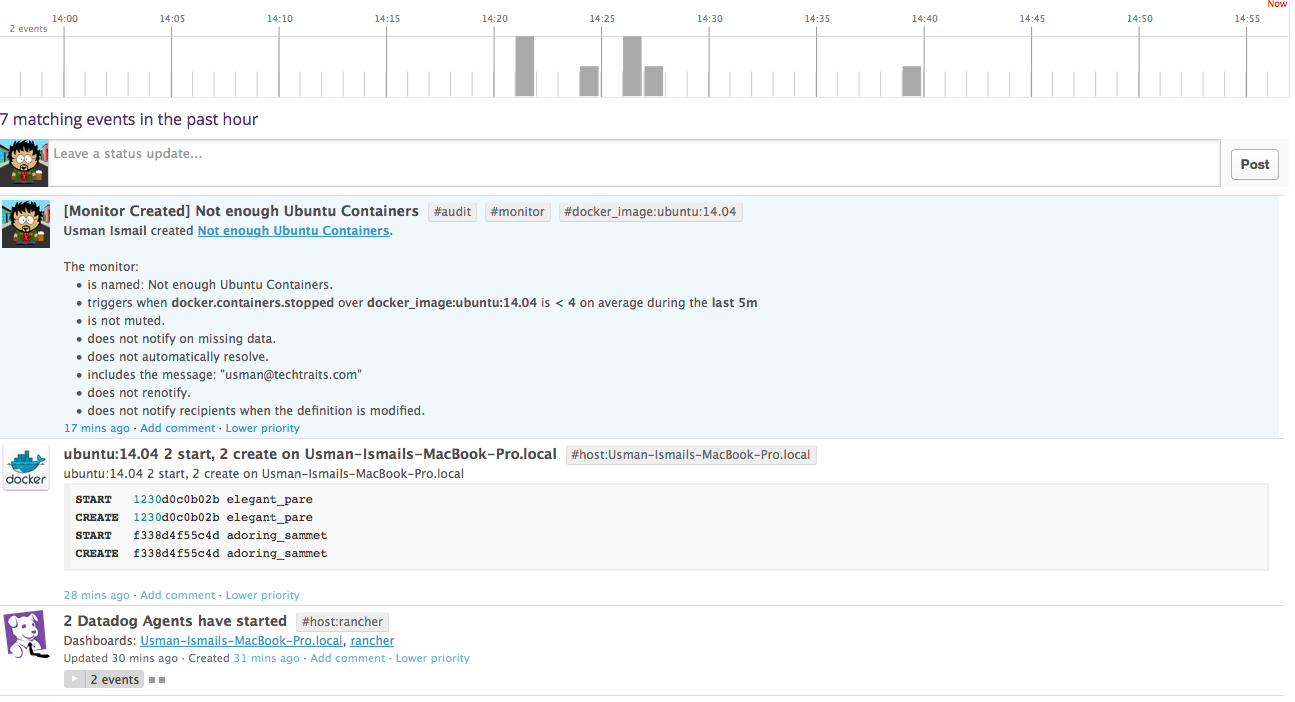
-v /var/run/docker.sock:/var/run/docker.sock \

-v /proc/mounts:/host/proc/mounts:ro \

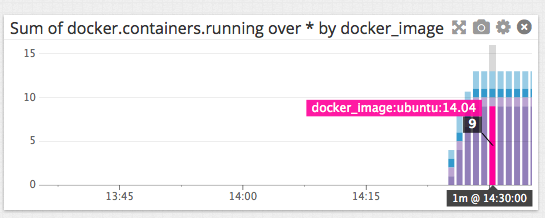
-v /sys/fs/cgroup/:/host/sys/fs/cgroup:ro \

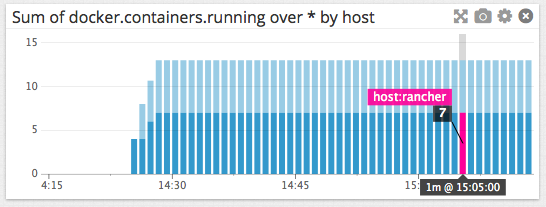
-e API\_KEY=YOUR\_API\_KEY datadog/docker-dd-agent \

Now that your containers are connected you can go to the Eventstab in the DataDog web console and see all events pertaining to your cluster. All container launches and terminations will be part of this event stream.

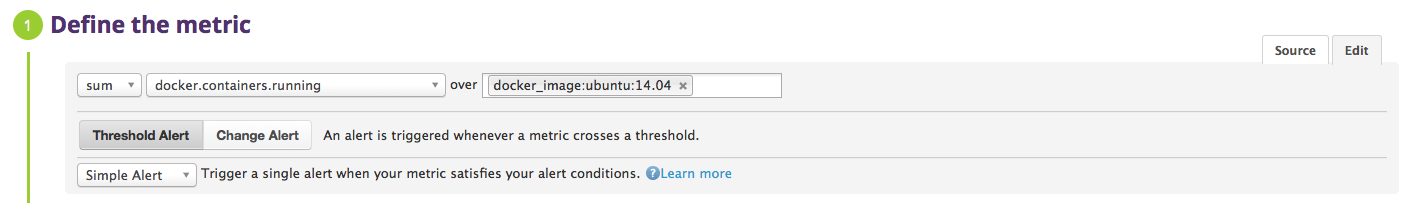
[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-21-at-2.56.04-PM.png)

You can also click the Dashboardstab and hit create dashboards to aggregate metrics across your entire cluster. Datadog collects metrics about CPU usage, memory and I/O for all containers running in the system. In addition you get counts of running and stopped containers as well as counts of docker images. The dashboard view allows you to create graphs for any metric or set of metrics across the entire deployment or grouped by host or container image. For example the graph below shows the number of running containers broken down by the image type, I am running 9 ubuntu:14.04 containers in my cluster at the moment.

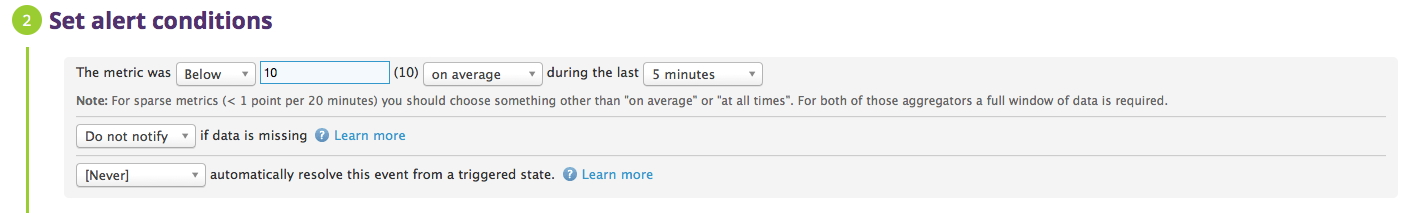
[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-21-at-2.35.21-PM.png)You could also split the same data by Hosts, as the second graph shows, 7 of the containers are running on my Rancher host and the remaining ones on my local laptop.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-21-at-3.14.10-PM.png)

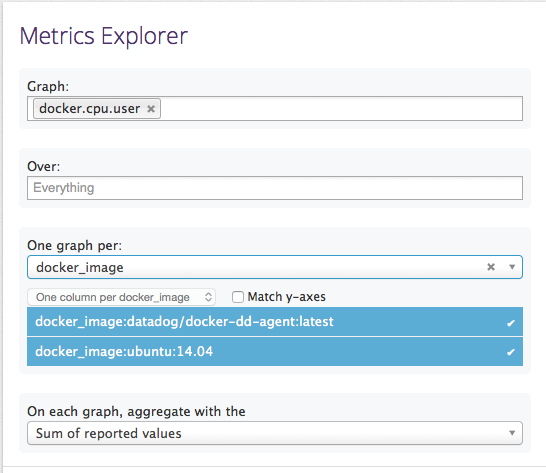
Data Dog also supports alerting using a feature called Monitors. A monitor is DataDog’s equivalent to a Scout trigger and allows you to define thresholds for various metrics. DataDog’s alerting system is a lot more flexible and detailed then Scout’s. The example below shows how to specify that you are concerned about Ubuntu containers terminating hence you would monitor the docker.containers.running metric for containers created from the ubuntu:14.04 docker image.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-22-at-6.49.53-PM.png)

Then specify the alert conditions to say that if there are fewer than ten ubuntu containers in our deployment (on average) for the last 5 minutes, you would like to be alerted. Although not shown here you will also be asked to specify the text of the message which is sent out when this alert is triggered as well as the target audience for this alert. In the current example I am using a simple absolute threshold. You can also specify a delta based alert which triggers if say the avg stopped container count was four over the last five minutes raise an alert.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-22-at-6.49.58-PM.png)

Lastly, using the Metrics Explorer tab you can make ad-hoc aggregations over your metrics to help debug issues or extract specific information from your data. This view allows your to graph any metric over a slice based on container image or host. You may combine output into a single graph or generate a set of graphs by grouping across images or hosts.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-21-at-2.40.30-PM.png)

DataDog is a significant improvement  over scout in terms feature set, easy of use and user friendly design. However this level of polish comes with additional cost as each DataDog agent costs $15.

Score Card:

1. Easy of deployment: \*\*\*\*\*
2. Level of detail: \*\*\*\*\*
3. Level of aggregation: \*\*\*\*\*
4. Ability to raise alerts: Supported
5. Ability to monitor non-docker resources: \*\*\*\*\*
6. Cost: $15 / host[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/sensu.png)

**Sensu Monitoring Framework**

Scout and Datadog provide centralized monitoring and alerting however both are hosted services that can get expensive for large deployments. If you need a self-hosted, centralized metrics service,  you may consider the [sensu open source monitoring framework](http://sensuapp.org/). To run the Sensu server you can use the [hiroakis/docker-sensu-server](https://registry.hub.docker.com/u/hiroakis/docker-sensu-server/)container. This container installs sensu-server, the uchiwa web interface, redis, rabbitmq-server, and  the sensu-api. Unfortunately sensu does not have any docker support out of the box. However, using the plugin system you can configure support for both container metrics as well as status checks.

Before launch your sensu server container you must define a check that you can load into the server. Create a file called check-docker.json and add the following contents into the file.  In this file you are telling the Sensu server to run a script called load-docker-metrics.sh every ten seconds on all clients which are subscribed to the docker tag. You will define this script a little later.

{

"checks": {

"load\_docker\_metrics": {

"type": "metric",

"command": "load-docker-metrics.sh",

"subscribers": [

"docker"

],

"interval": 10

}

}

}

Now you can run the sensu server docker container with our check configuration file using the command below. Once you run the command you should be able to launch the uchiwa dashboard at http://YOUR\_SERVER\_IP:3000 in your browser.

docker run -d --name sensu-server \

-p 3000:3000 \

-p 4567:4567 \

-p 5671:5671 \

-p 15672:15672 \

-v $PWD/check-docker.json:/etc/sensu/conf.d/check-docker.json \

hiroakis/docker-sensu-server

Now that the sensu server is up you can launch sensu clients on each of the hosts running our docker containers. You told the server that the containers will have a script called load-docker-metrics.sh so lets create the script and insert it into our client containers. Create the file and add the text shown below into the file, replacing HOST\_NAME with a logical name for your host . The script below is using the [Docker Remote API](https://docs.docker.com/reference/api/docker_remote_api_v1.17/)to pull in the meta data for running containers, all containers and all images on the host. It then prints the values out using sensu’s key value notation. The sensu server will read the output values from the STDOUT and collect those metrics. This example only pulls these three values but you could make the script as detailed as required. Note that you could also add multiple check scripts such as thos, as long as you reference them in the server configuration file you created earlier. You can also define that you want the check to fail if the number of running containers ever falls below three. You can make a check fail by returning a non-zero value from the check script.

#!/bin/bash

set -e

# Count all running containers

running\_containers=$(echo -e "GET /containers/json HTTP/1.0\r\n" | nc -U /var/run/docker.sock \

| tail -n +5 \

| python -m json.tool \

| grep \"Id\" \

| wc -l)

# Count all containers

total\_containers=$(echo -e "GET /containers/json?all=1 HTTP/1.0\r\n" | nc -U /var/run/docker.sock \

| tail -n +5 \

| python -m json.tool \

| grep \"Id\" \

| wc -l)

# Count all images

total\_images=$(echo -e "GET /images/json HTTP/1.0\r\n" | nc -U /var/run/docker.sock \

| tail -n +5 \

| python -m json.tool \

| grep \"Id\" \

| wc -l)

echo "docker.HOST\_NAME.running\_containers ${running\_containers}"

echo "docker.HOST\_NAME.total\_containers ${total\_containers}"

echo "docker.HOST\_NAME.total\_images ${total\_images}"

if [ ${running\_containers} -lt 3 ]; then

exit 1;

fi

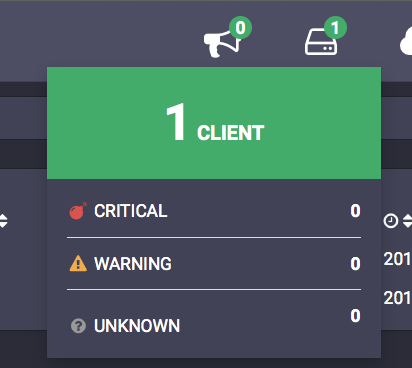
Now that you have defined your load docker metrics check you need to to start the sensu client using the [usman/sensu-client](https://registry.hub.docker.com/u/usman/sensu-client) container I defined for this purpose. You can use the command shown below to launch sensu client. Note that the container must run as privileged in order to be able to access unix sockets, it must have the docker socket mounted in as a volume as well as the load-docker-metrics.sh script you defined above. Make sure the load-docker-metrics.sh script is marked as executable in your host machine as the permissions carry through into the container. The container also takes in SENSU\_SERVER\_IP, RABIT\_MQ\_USER, RABIT\_MQ\_PASSWORD, CLIENT\_NAME and  CLIENT\_IP as parameters, please specify the value of these parameters for your setup. The default values for the RABIT\_MQ\_USER RABIT\_MQ\_PASSWORD are sensu andpassword.

docker run -d --name sensu-client --privileged \

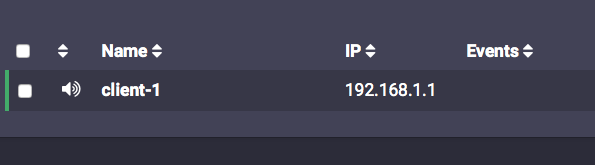
-v $PWD/load-docker-metrics.sh:/etc/sensu/plugins/load-docker-metrics.sh \

-v /var/run/docker.sock:/var/run/docker.sock \

usman/sensu-client SENSU\_SERVER\_IP RABIT\_MQ\_USER RABIT\_MQ\_PASSWORD CLIENT\_NAME CLIENT\_IP

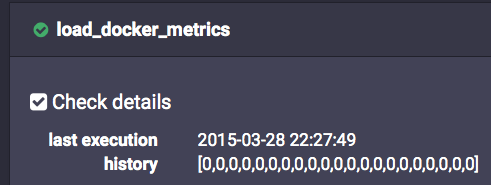
[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-28-at-10.13.48-PM.png)

A few seconds after running this command you should see  the client count increase to 1 in the  uchiwa dashboard. If you click the clients icon you should see a list of your clients including the client that you just added. I named my clientclient-1 and specified the host IP as 192.168.1.1.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-28-at-10.13.54-PM.png)

If you click on the client name your should get further details of the checks. You can see that the load\_docker\_metrics check was run at 10:22 on the 28th of March.

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-28-at-10.14.00-PM.png)If you Click on the check name you can see further details of check runs. The zeros indicate that there were no errors, if the script had failed (if for example your docker Daemon dies) you would see an error code (non zero) value. Although it is not covered this in the current article you can also setup sensu to alert you when these checks fail using [Handlers](http://sensuapp.org/docs/0.11/adding_a_handler). Furthermore, uchiwa only shows the values of checks and not the metrics collected. Note that sensu does not store the collected metrics, they have to be forwarded to a time series database such as InfluxDB or Graphite. This is also done through Handlers. Please find details of how to configure metric forwarding to graphite [here](http://www.joemiller.me/2012/02/02/sensu-and-graphite/).

[](http://cdn.rancher.com/wp-content/uploads/2015/03/13030215/Screen-Shot-2015-03-28-at-10.27.59-PM.png)

Sensu ticks all the boxes in our evaluation criteria; you can collect as much detail about our docker containers and hosts as ypu want. In addition you are able to aggregate the values of all of out hosts in one place and raise alerts over those checks. The alerting is not as advanced as DataDog or Scout, as  you are only able to alert on checks failing on individual hosts. However, the big drawback of Sensu is difficulty of deployment. Although I have automated many steps in the deployment using docker containers, Sensu remains a complicated system requiring us to install, launch and maintain separate processes for Redis, RabitMQ, Sensu API, uchiwa and Sensu Core. Furthermore, you would require still more tools such as Graphite to present metric values and a production deployment would require customizing the containers I have used today for secure passwords and custom ssl certificates. In addition were you to add more checks after launching the container you would have to restart the Sensu server as that is the only way for it start collecting new metrics. For these reasons I rate Sensu fairly low for ease of deployment.

Easy of deployment: \*  
Level of detail: \*\*\*\*  
Level of aggregation: \*\*\*\*  
Ability to raise alerts: Supported but limited  
Ability to monitor non-docker resources: \*\*\*\*\*  
Cost: $Free

I also evaluated two other monitoring services, Prometheus and Sysdig Cloud in a[*second article*](http://rancher.com/docker-monitoring-continued-prometheus-and-sysdig/), and have included them in this post for simplicity.

**Prometheus**

First lets take a look at Prometheus; it is a self-hosted set of tools which collectively provide metrics storage, aggregation, visualization and alerting. Most of the tools and services we have looked at so far have been push based, i.e. agents on the monitored servers talk to a central server (or set of servers) and send out their metrics. Prometheus on the other hand is a pull based server which expects monitored servers to provide a web interface from which it can scrape data. There are several [exporters available](http://prometheus.io/docs/instrumenting/exporters/) for Prometheus which will capture metrics and then expose them over http for Prometheus to scrape. In addition there are [libraries](http://prometheus.io/docs/instrumenting/clientlibs/) which can be used to create custom exporters. As we are concerned with monitoring docker containers we will use the [container\_exporter](https://github.com/docker-infra/container_exporter)capture metrics. Use the command shown below to bring up the container-exporter docker container and browse tohttp://MONITORED\_SERVER\_IP:9104/metrics to see the metrics it has collected for you. You should launch exporters on all servers in your deployment. Keep track of the respective MONITORED\_SERVER\_IPs as we will  be using them later in the configuration for Prometheus.

docker run -p 9104:9104 -v /sys/fs/cgroup:/cgroup -v /var/run/docker.sock:/var/run/docker.sock prom/container-exporter

Once we have got all our exporters running we are can launch Prometheus server. However, before we do we need to create a configuration file for Prometheus that tells the server where to scrape the metrics from. Create a file calledprometheus.conf and then add the following text inside it.

global:

  scrape\_interval: 15s

  evaluation\_interval: 15s

  labels:

    monitor: exporter-metrics

rule\_files:

scrape\_configs:

- job\_name: prometheus

  scrape\_interval: 5s

  target\_groups:

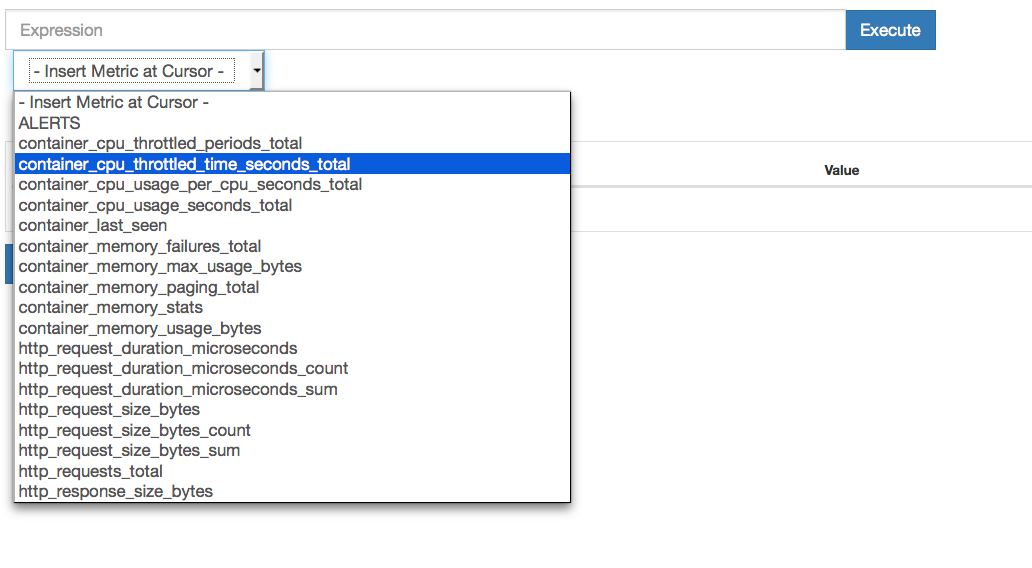
    # These endpoints are scraped via HTTP.

    - targets: ['localhost:9090','MONITORED\_SERVER\_IP:9104']

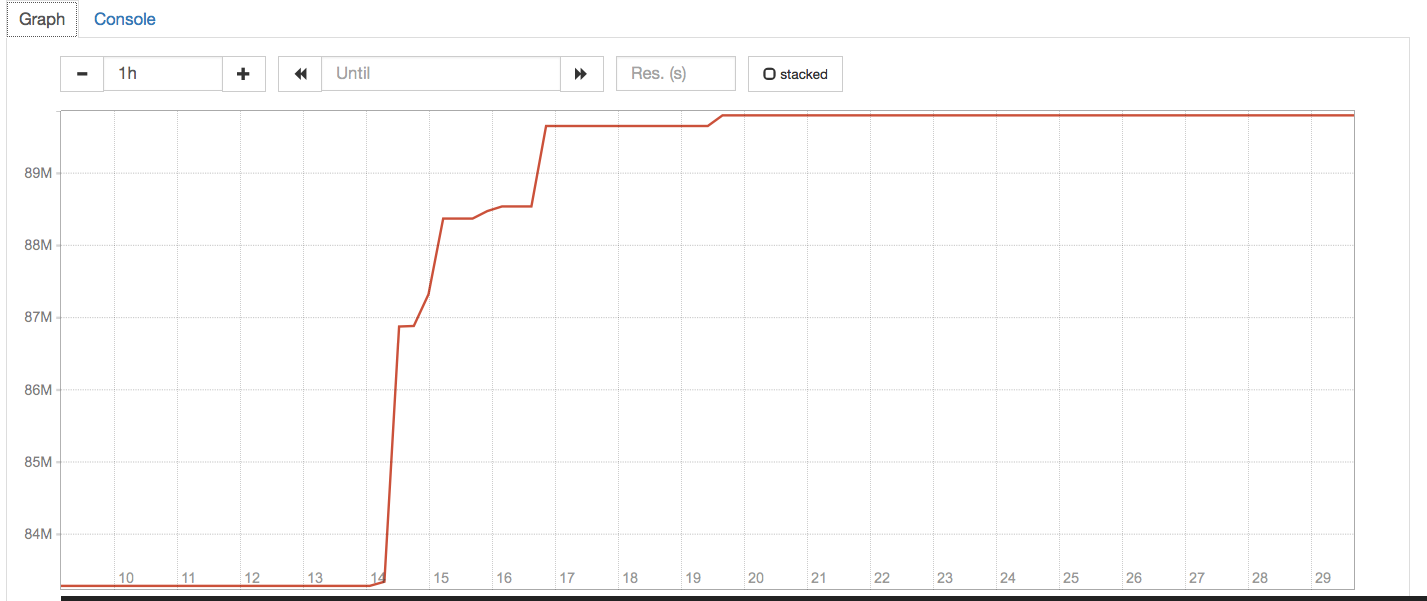
In this file there are two sections, global and job(s). In the global section we set defaults for configuration properties such as data collection interval (scrape\_interval).  We can also add labels which will be appended to all metrics. In the jobs section we can define one or more jobs that each have a name, an optional override scraping interval as well as one or more targets from which to scrape metrics. We are adding two targets, one is the Prometheus server itself and the second is the container-exporter we setup earlier. If you setup more than one exporter your can setup additional targets to pull metrics from all of them. Note that the job name is available as a label on the metric hence you may want to setup separate jobs for your various types of servers. Now that we have a configuration file we can start a Prometheus server using the [prom/prometheus](https://registry.hub.docker.com/u/prom/prometheus/) docker image.

docker run -d --name prometheus-server -p 9090:9090 -v $PWD/prometheus.conf:/prometheus.conf prom/prometheus -config.file=/prometheus.conf

After launching the container, Prometheus server should be available in your browser on the port 9090 in a few moments. Select Graph from the top menu and select a metric from the drop down box to view its latest value. You can also write queries in the expression box which can find matching metrics. Queries take the form **METRIC\_NAME{LABEL\_NAME=LABEL\_VALUE, …}**. You can find more details of the query syntax [here](http://prometheus.io/docs/querying/).



We are able to drill down into the data using queries to filter out data from specific server types (jobs) and containers. All metrics from containers are labeled with the image name, container name and the host on which the container is running. Since metric names do not encompass container or server name we are able to  easily aggregate data across our deployment. For example we can filter for the **container\_memory\_usage\_bytes {image=”ubuntu”}** to get information about the memory usage of all ubuntu containers in our deployment. Using the built in functions we can also aggregate the resulting set of of metrics. For example**average\_over\_time(container\_memory\_usage\_bytes {image=”ubuntu”}[5m])**will show the memory used by ubuntu containers, averaged over the last five minutes. Once you are happy with with a query you can click over to the Graph tab and see the variation of the metric over time.



Temporary graphs are great for ad-hoc investigations but you also need to have persistent graphs for dashboards. For this you can use the [Prometheus Dashboard Builder](https://registry.hub.docker.com/u/prom/promdash/). To launch Prometheus Dashboard Builder you need access to an SQL database which you can create using the official MySQL  [Docker image](https://registry.hub.docker.com/_/mysql/). The command to launch the MySQL container is shown below, note that you may select any value for database name, user name, user password and root password however keep track of these values as they will be needed later.

docker run -p 3306:3306 --name promdash-mysql   \

-e MYSQL\_DATABASE=<database-name>            \

-e MYSQL\_USER=<database-user>                \

-e MYSQL\_PASSWORD=<user-password>            \

-e MYSQL\_ROOT\_PASSWORD=<root-password>       \

-d mysql

Once you have the database setup, use the rake installation inside the promdash container to initialize the database. You can then run the Dashboard builder by running the same container. The command to initialize the database and bring up the Prometheus Dashboard Builder are shown below.

# Initialize Database

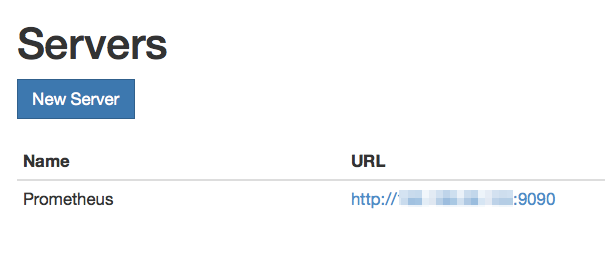
docker run --rm -it --link promdash-mysql:db \

-e DATABASE\_URL=mysql2://<database-user>:<user-password>@db:3306/<database-name> prom/promdash ./bin/rake db:migrate

# Run Dashboard

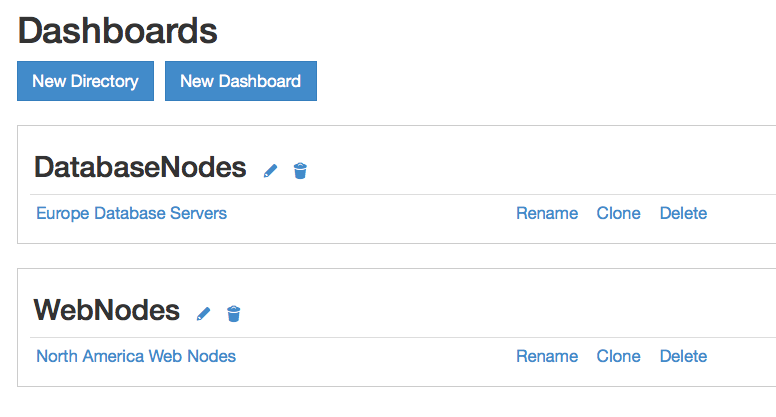
docker run -d --link promdash-mysql:db -p 3000:3000 --name prometheus-dash \

-e DATABASE\_URL=mysql2://<database-user>:<user-password>@db:3306/<database-name> prom/promdash

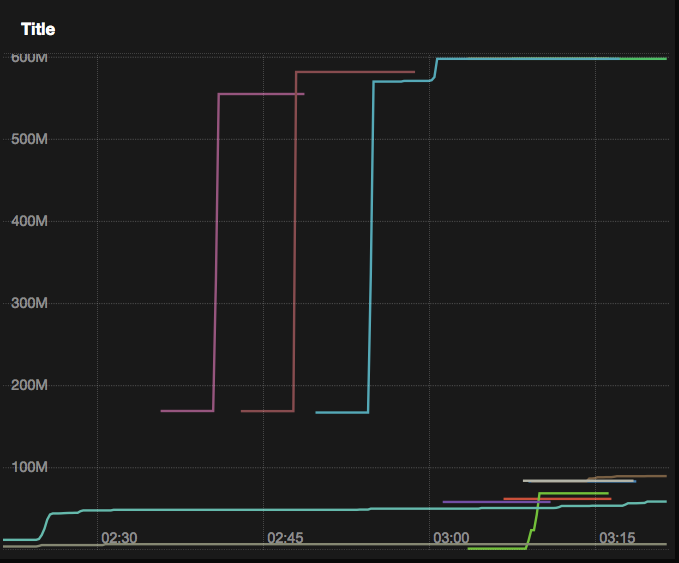


Once your container is running you can browse to port 3000 and load up the dashboard builder UI. In the UI you need to click Servers in the top menu and New Serverto add your Prometheus Server as a datasource for the dashboard builder. Addhttp://PROMETHEUS\_SERVER\_IP:9090 to the list of servers and hit Create Server.

Now click Dashboards in the top menu, here you can create Directories (Groups of Dashboards) and Dashboards. For example we created a directory for Web Nodes and one for Database Nodes and in each we create a dashboard as shown below.



Once you have created a dashboard you can add metrics by mousing over the title bar of a graph and selecting the data sources icon (Three Horizontal lines with an addition sign following them ). You can then select the server which you added earlier, and a query expression which you tested in the Prometheus Server UI. You can add multiple data sources into the same graph in order to see a comparative view.



You can add multiple graphs (each with possibly multiple data sources) by clicking the **Add Graph** button. In addition you may select the time range over which your dashboard displays data as well as a refresh interval for auto-loading data. The dashboard is not as polished as the ones from Scout and DataDog, for example there is no easy way to explore metrics or build a query in the dashboard view. Since the dashboard runs independently of the Prometheus server we can’t ‘pin’ graphs generated in the Prometheus server into a dashboard. Furthermore several times we noticed that the UI would not update based on selected data until we refreshed the page. However, despite its issues the dashboard is feature competitive with DataDog and because Prometheus is under heavy development, we expect the bugs to be resolved over time. In comparison to other self-hosted solutions Prometheus is a lot more user friendly than Sensu and allows you present metric data as graphs without using third party visualizations. It also is able to provide much better analytical capabilities than CAdvisor.

Prometheus also has the ability to apply alerting rules over the input data and displaying those on the UI. However, to be able to do something useful with alerts such send emails or notify [pagerduty](http://www.pagerduty.com/) we need to run the the [Alert Manager](https://registry.hub.docker.com/u/prom/alertmanager/). To run the Alert Manager you first need to create a configuration file. Create a file called alertmanager.conf and add the following text into it:

notification\_config {

    name: "ubuntu\_notification"

pagerduty\_config {

        service\_key: "<PAGER\_DUTY\_API\_KEY>"

    }

email\_config {

        email: "<TARGET\_EMAIL\_ADDRESS>"

    }

hipchat\_config {

        auth\_token: "<HIPCHAT\_AUTH\_TOKEN>"

room\_id: 123456

    }

}

aggregation\_rule {

    filter {

        name\_re: "image"

value\_re: "ubuntu:14.04"

    }

repeat\_rate\_seconds: 300

notification\_config\_name: "ubuntu\_notification"

}

In this configuration we are creating a notification configuration calledubuntu\_notification, which specifies that alerts must go to the PagerDuty, Email and HipChat. We need to specify the relevant API keys and/or access tokens for the HipChat and PagerDutyNotifications to work. We are also specifying that the alert configuration should only apply to alerts on metrics where the label **image**has the value **ubuntu:14.04**. We specify that a triggered alert should not retrigger for at least 300 seconds after the first alert is raised. We can bring up the Alert Manager using the docker image by volume mounting our configuration file into the container using the command shown below.

docker run -d -p 9093:9093 -v $PWD:/alertmanager prom/alertmanager -logtostderr -config.file=/alertmanager/alertmanager.conf

Once the container is running you should be able to point your browser to port 9093 and load up the Alarm Manger UI. You will be able to see all the alerts raised here, you can ‘silence’ them or delete them once the issue is resolved. In addition to setting up the Alert Manager we also need to create a few alerts. Add **rule\_file: “/prometheus.rules”** in a new line into the global section of the prometheus.conffile you created earlier. This line tells Prometheus to look for alerting rules in theprometheus.rules file. We now need to create the rules file and load it into our server container. To do so create a file called prometheus.rules in the same directory where you created prometheus.conf. and add the following text to it:

ALERT HighMemoryAlert

IF container\_memory\_usage\_bytes{image="ubuntu:14.04"} > 1000000000

FOR 1m

WITH {}

SUMMARY "High Memory usage for Ubuntu container"

DESCRIPTION "High Memory usage for Ubuntu container on {{$labels.instance}} for container {{$labels.name}} (current value: {{$value}})"

In this configuration we are telling Prometheus to raise an alert called**HighMemoryAlert** if the **container\_memory\_usage\_bytes metric** for containers using the Ubuntu:14.04 image goes above 1 GB for 1 minute. The summary and the description of the alerts is also specified in the rules file. Both of these fields can contain placeholders for label values which are replaced by Prometheus. For example our description will specify the server instance (IP) and the container name for metric raising the alert. After launching the Alert Manager and defining your Alert rules, you will need to re-run your Prometheus server with new parameters. The commands to do so are below:

# stop and remove current container

docker stop prometheus-server && docker rm prometheus-server

# start new container

docker run -d --name prometheus-server -p 9090:9090         \

-v $PWD/prometheus.conf:/prometheus.conf                  \

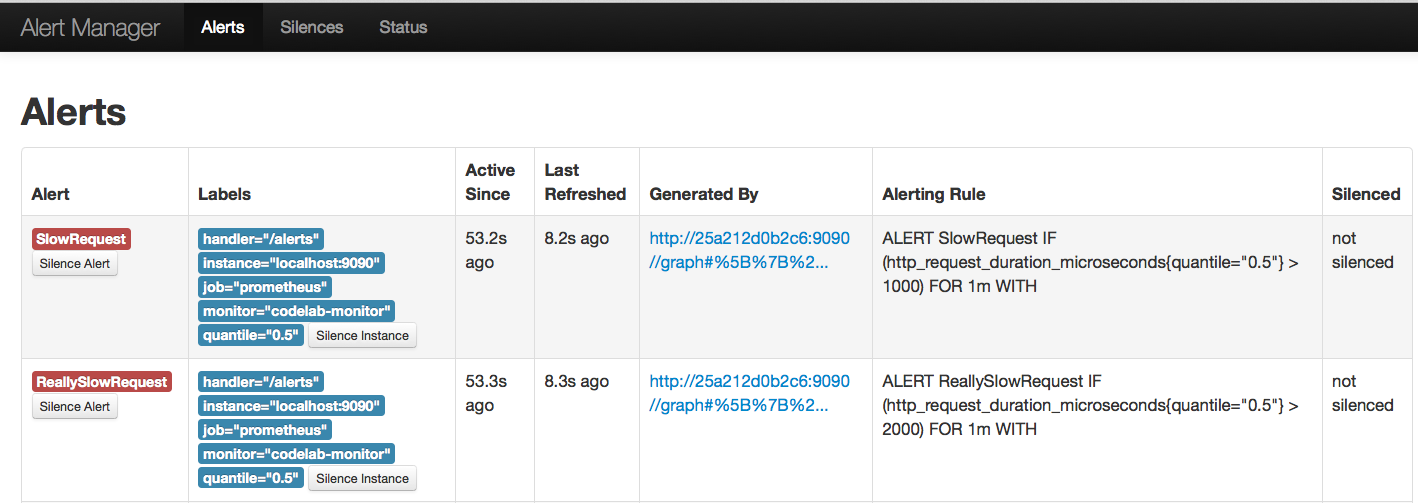
-v $PWD/prometheus.rules:/prometheus.rules                \

prom/prometheus                                           \

-config.file=/prometheus.conf                             \

-alertmanager.url=http://ALERT\_MANAGER\_IP:9093

Once the Prometheus Server is up again you can click Alerts in the top menu of the Prometheus Server UI to bring up a list of alerts and their statuses. If and when an alert is fired you will also be able to see it in the Alert Manager UI and any external service defined in the alertmanager.conf file.



Collectively the Prometheus tool-set’s feature set is on par with DataDog which has been our best rated Monitoring tool so far. Prometheus uses a very simple format for input data and can ingest from any web endpoint which presents the data. Therefore we can monitor more or less any resource with Prometheus, and there are already several libraries defined to monitor common resources. Where Prometheus is lacking is in level of polish and ease of deployment. The fact that all components are dockerized is a major plus however, we had to launch 4 different containers each with their own configuration files to support the Prometheus server. The project is also lacking detailed, comprehensive documentation for these various components. However, in caparison to self-hosted services such as CAdvisor and Sensu, Prometheus is a much better toolset. It is significantly easier setup than sensu and has the ability to provide visualization of metrics without third party tools. It is able has much more detailed metrics than CAdvisor and is also able to monitor non-docker resources.  The choice of using pull based metric aggregation rather than push is less than ideal as you would have to restart your server when adding new data sources. This could get cumbersome in a dynamic environment such as cloud based deployments. Prometheus does offer the [Push Gateway](https://github.com/prometheus/pushgateway) to bridge the disconnect.  However, running yet another service will add to the complexity of the setup. For these reasons I still think DataDog is probably easier for most users, however, with some polish and better packaging Prometheus could be a very compelling alternative, and out of self-hosted solutions Prometheus is my pick.

Score Card:

1. Easy of deployment: \*\*
2. Level of detail: \*\*\*\*\*
3. Level of aggregation: \*\*\*\*\*
4. Ability to raise alerts: \*\*\*\*
5. Ability to monitor non-docker resources: Supported
6. Cost: Free

**Sysdig Cloud**

Sysdig cloud is a hosted service that provides metrics storage, aggregation, visualization and alerting. To get started with sysdig sign up for a trial account at[https://app.sysdigcloud.com](https://app.sysdigcloud.com/). and complete the registration form. Once you complete the registration form and log in to the account, you will be asked toSetup your Environment and be given a curl command similar to the shown below. Your command will have your own secret key after the -s switch. You can run this command on the host running docker and which you need to monitor. Note that you should replace the [TAGS] place holder with tags to group your metrics. The tags are in the format TAG\_NAME:VALUE so you may want to add a tag role:web or deployment:production. You may also use the containerized sysdig agent.

# Host install of sysdig agent

curl -s https://s3.amazonaws.com/download.draios.com/stable/install-agent | sudo bash -s 12345678-1234-1234-1234-123456789abc [TAGS]

# Docker based sysdig agent

docker run --name sysdig-agent --privileged --net host \

  -e ACCESS\_KEY=12345678-1234-1234-1234-123456789abc   \

  -e TAGS=os:rancher                                   \

  -v /var/run/docker.sock:/host/var/run/docker.sock    \

  -v /dev:/host/dev -v /proc:/host/proc:ro             \

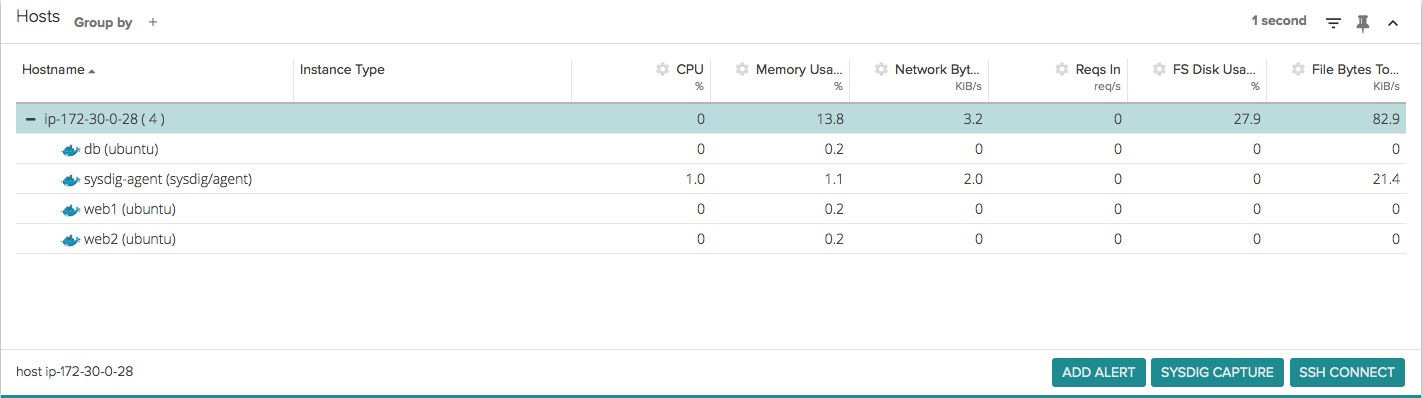
  -v /boot:/host/boot:ro                               \

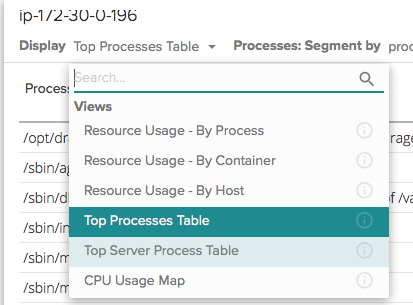
  -v /lib/modules:/host/lib/modules:ro                 \

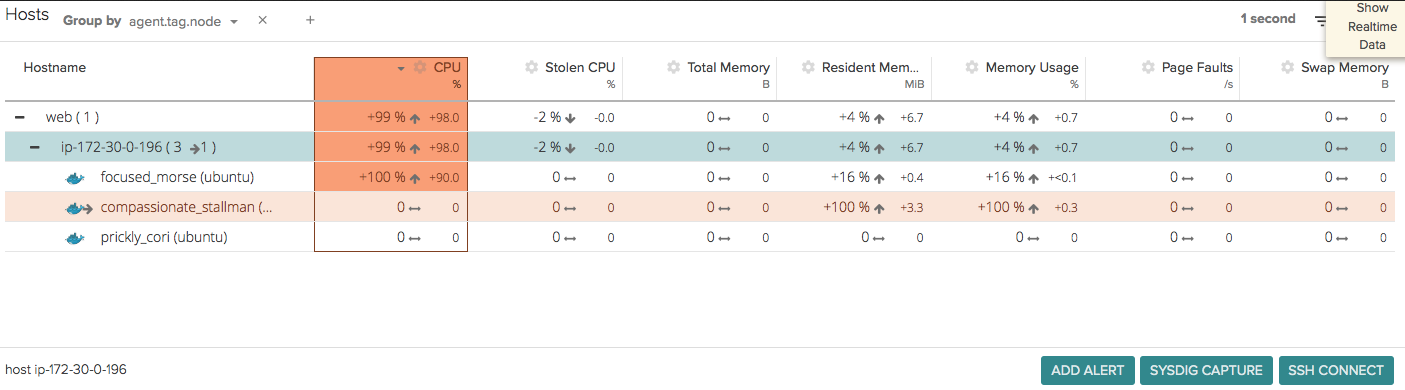
  -v /usr:/host/usr:ro sysdig/agent

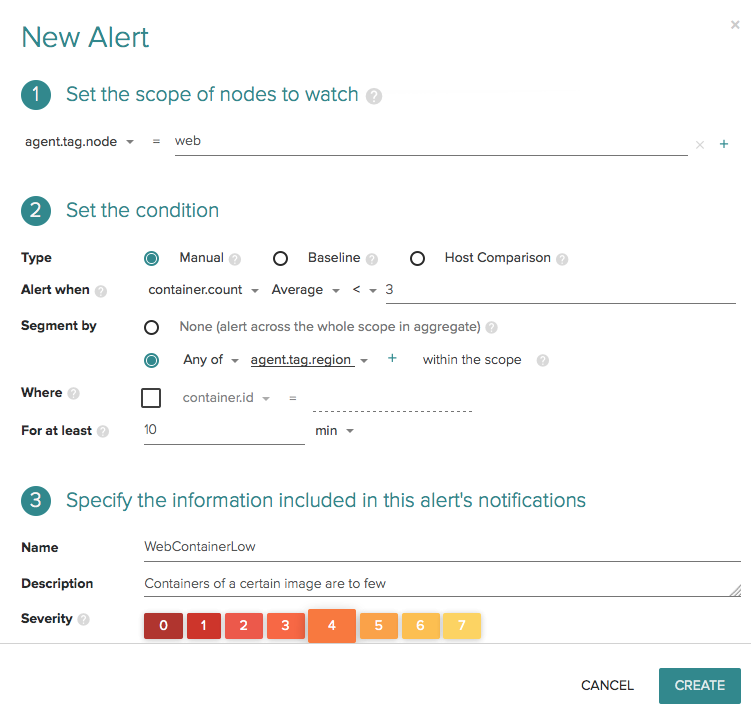
Even if you use docker you will still need to install Kernel headers in the host OS. This goes against Docker’s philosophy of isolated micro services. However, installing kernel headers is fairly benign. Installing the headers and getting sysdig running is trivial if you are using a mainstream kernel such us CentOS, Ubuntu or Debian. Even the Amazon’s custom kernels are supported however RancherOS’s custom kernel presented problems for sysdig as did the tinycore kernel. So be warned if you would like to use Sysdig cloud on non-mainstream kernels you may have to get your hands dirty with some system hacking.

After you run the agent you should see the Host in the Sysdig cloud console in the**Explore** tab. Once you launch docker containers on the host those will also be shown. You can see basic stats about the CPU usage, memory consumption, network usage. The metrics are aggregated for the host as well as broken down per container.

[](http://cdn.rancher.com/wp-content/uploads/2015/04/20181622/Screen-Shot-2015-04-14-at-12.06.36-PM.png)By selecting one of the hosts or containers you can get a whole host of other metrics including everything provided by the docker stats API. Out of all the systems we have seen so far sysdig certainly has the most comprehensive set of metrics out of the box. You can also select from several pre-configured dashboards which present a graphical or tabular representation of your deployment.

[](http://cdn.rancher.com/wp-content/uploads/2015/04/20181622/Screen-Shot-2015-04-16-at-11.26.53-AM.png)

You can see live metrics, by selecting **Real-time Mode** (Target Icon) or select a window of time over which to average values. Furthermore, you can also setup comparisons which will highlight the delta of current values and values at a point in the past. For example the table below shows values compared with those from ten minutes ago. If the CPU usage is significantly higher than 10 minutes ago you may be experiencing load spikes and need to scale out. The UI is at par with, if not better than DataDog for identifying and exploring trends in the data.[](http://cdn.rancher.com/wp-content/uploads/2015/04/20181622/Screen-Shot-2015-04-19-at-4.59.09-PM.png)

In addition to exploring data on an ad-hoc basis you can also create persistent dashboards. Simply click the pin icon on any graph in the explore view and save it to a named dashboard. You can view all the dashboards and their associated graphs by clicking the **Dashboards tab.**You can also select the bell icon on any graph and create an alert from the data. The Sysdig cloud supports detailed alerting criteria and is again one of the best we have seen. The example below shows an alert which triggers if the count of containers labeled web falls below three on average for the last ten minutes. We are also segmenting the data by theregion tag, so there will be a separate check for web nodes in North America and Europe. Lastly, we also specify a Name, description and Severity for the alerts. You can control where alerts go by going to **Settings** (Gear Icon) > Notifications and add email addresses or SNS Topics to send alerts too. Note all alerts go to all notification endpoints which may be problematic if you want to wake up different people for different alerts.[](http://cdn.rancher.com/wp-content/uploads/2015/04/20181622/Screen-Shot-2015-04-19-at-4.55.35-PM.png)

I am very impressed with Sysdig cloud as it was trivially easy to setup, provides detailed metrics with great visualization tools for real-time and historical data. The requirement to install kernel headers on the host OS is troublesome though and lack of documentation and support for non-standard kernels could be problematic in some scenarios. The alerting system in the Sysdig cloud is among the best we have seen so far, however, the inability to target different email addresses for different alerts is problematic. In a larger team for example you would want to alert a different team for database issues vs web server issues. Lastly, since it is in beta the pricing for Sysdig cloud is not easily available. I have reached out to their sales team and will update this article if and when they get back to me. If sysdig is price competitive then Datadog has serious competition in the hosted service category.

Score Card:

1. Easy of deployment: \*\*\*
2. Level of detail: \*\*\*\*\*
3. Level of aggregation: \*\*\*\*\*
4. Ability to raise alerts: \*\*\*\*
5. Ability to monitor non-docker resources: Supported
6. Cost: Must Contact Support

**Conclusion**

Today’s article has covered several options for monitoring docker containers, ranging from free options; docker stats, CAdvisor, Prometheus or Sensu to paid services such as Scout, Sysdig Cloud and DataDog. From my research so far DataDog seems to be the best-in-class system for monitoring docker deployments. The setup was complete in seconds with a one-line command, all hosts were reporting metrics in one place, historical trends were apparent in the UI and Datadog supports deep diving into metrics as well as alerting. However, at $15 per host the system can get expensive for large deployments. For larger scale, self-hosted deployments Sensu is able to fulfill most requirements however the complexity in setting up and managing a Sensu cluster may be prohibitive. Obviously, there are plenty of other self-hosted options, such as Nagios or Icinga, which are similar to Sensu.

Hopefully this gives you an idea of some of the options for monitoring containers available today.  I am continuing to investigate other options, including a more streamlined self-managed container monitoring system using CollectD, Graphite or InfluxDB and Grafana. Stay tuned for more details.

ADDITIONAL INFORMATION: After publishing this article I had some suggestions to also evaluate Prometheus and Sysdig Cloud, two other very good options for monitoring Docker. We’ve now included them in this article, for ease of discovery.  You can find the original second part of my post[here](http://rancher.com/docker-monitoring-continued-prometheus-and-sysdig/).

To learn more about monitoring and managing Docker, please join us for our next Rancher online meetup.

[**REGISTER NOW**](http://cta-service-cms2.hubspot.com/ctas/v2/public/cs/c/?cta_guid=8cf87b05-c26a-43ff-b5b4-efffd7991fd3&placement_guid=fec158c0-1df5-4fbd-9ff6-ffcd3c0acd02&portal_id=468859&redirect_url=APefjpHGm2lF04Fy4qpsI8etijP0iGCc7vHBI3pv0RYr2ojlakYZTrMX9LpHfI4jxBiejGZiziSLkHje4g8P2zCHjVaYZFM8YLCyOjz-TBnnWV5epaoRsbiozPiJmJyKSnFm3hbz-B-E9uhhNMrjQlOfjgCgFlMeVg&hsutk=&canon=http%3A%2F%2Francher.com%2Fcomparing-monitoring-options-for-docker-deployments%2F&__hstc=263286291.08af67b757c6d0281fc3d1e8bf9f9c7e.1494381410693.1494381410693.1494381410693.1&__hssc=263286291.1.1494381410694&__hsfp=2964040773)

Usman is a server and infrastructure engineer, with experience in building large scale distributed services on top of various cloud platforms. You can read more of his work at[*techtraits.com*](http://techtraits.com/), or follow him on twitter[*@usman\_ismail*](https://twitter.com/usman_ismail)or on[*GitHub*](https://github.com/usmanismail).