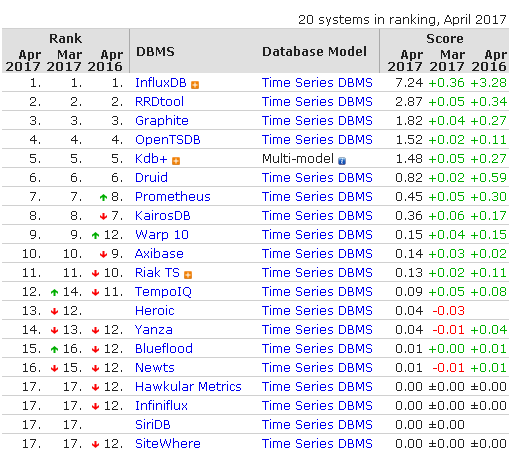
# DB-Engines Ranking - popularity ranking of time Series DBMS

https://db-engines.com/en/ranking/time+series+dbms



# TICK技术栈 -- DevOps轻量级监控解决方案

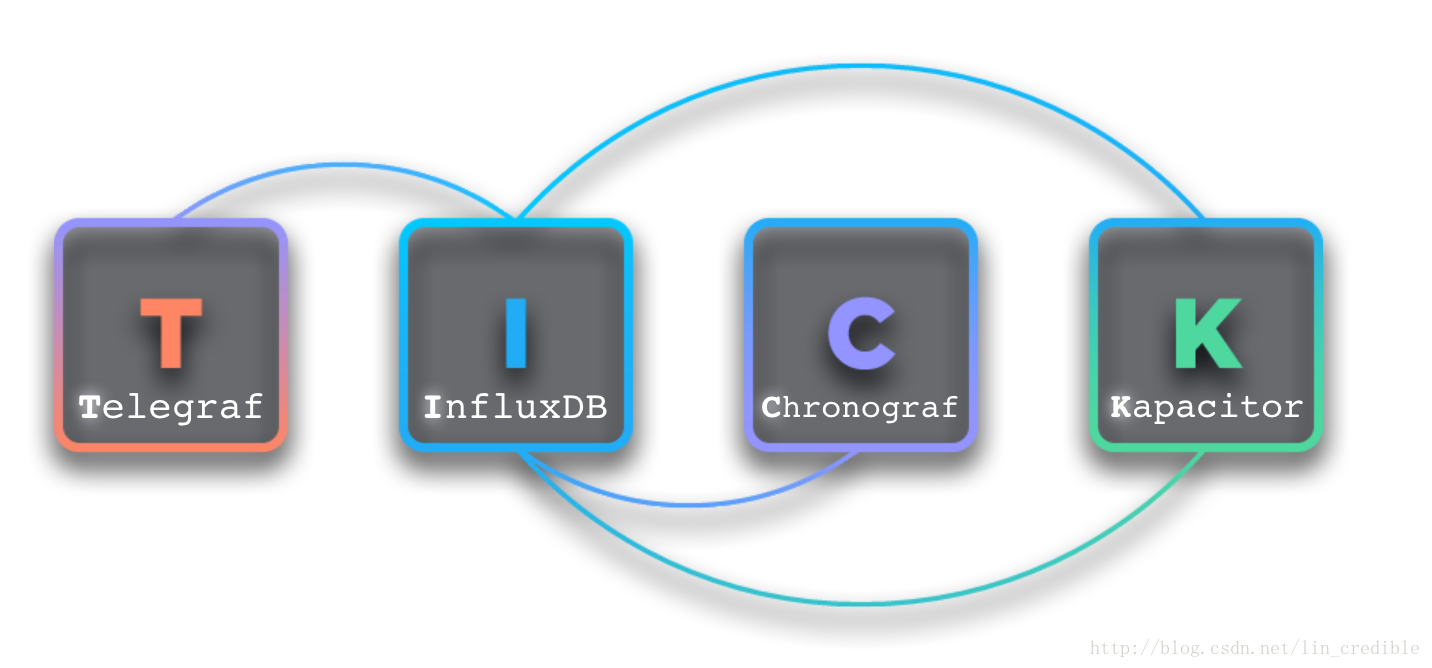
- (new Function()).name - 博客频道 - CSDN.NET

http://blog.csdn.net/lin\_credible/article/details/60579738

了解和学习TICK栈不久，还有很多需要进一步深入。但我个人非常看好这个项目，也希望更进一步研究，同时，在阅读源码和二次开发中，希望技术上能有所提升。另外，telegraf有CPU毛刺，已经转战更稳定的老牌 [collectd](https://github.com/collectd/collectd)。同时，生产环境，[grafana](https://github.com/grafana/grafana) 也是非常不错的选择，不需要二次开发，即可满足大部分需求。前端框架一个用的reactjs，一个angular1.5。感觉chronograf技术更轻量和前卫点儿，不过grafana也有非常多值得学习的地方，想二次开发的话，两个项目都值得深入 : ) 看个人选择了。

[TICK技术栈](https://www.influxdata.com/open-source/)

简介

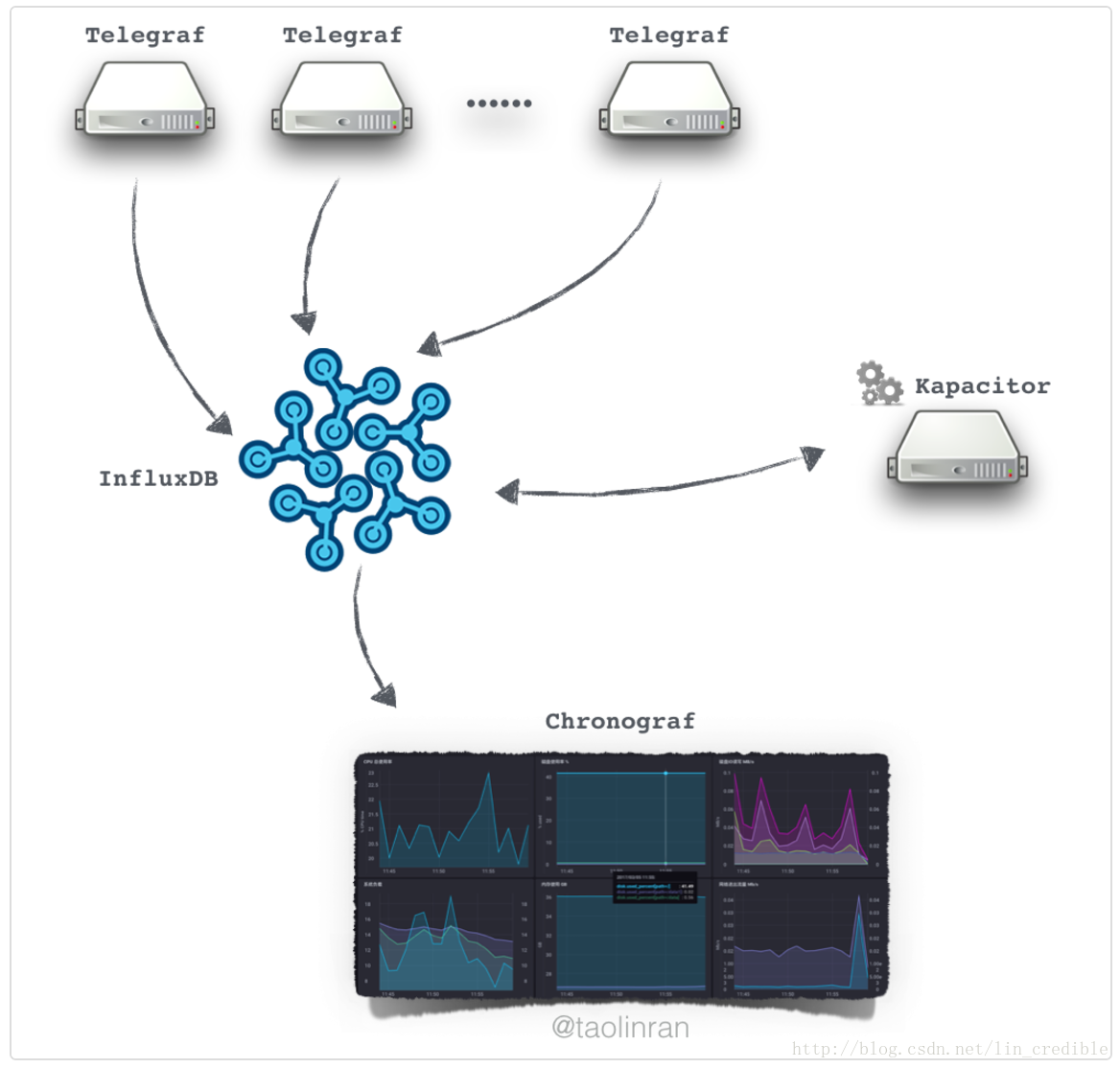


* **T** = [Telegraf](https://github.com/influxdata/telegraf) is a plugin-driven server agent for collecting and reporting metrics.
* **I** = [InfluxDB](https://github.com/influxdata/influxdb) is a time series database built from the ground up to handle high write and query loads.
* **C** = [Chronograf](https://github.com/influxdata/chronograf) is a graphing and visualization application for performing ad hoc exploration of data.
* **K** = [Kapacitor](https://github.com/influxdata/kapacitor) is a data processing framework proving alerting, anomaly detection and action frameworks.

简单表达：

* Telegraf - 数据采集
* InfluxDB - 数据接收和存储
* Chronograf - 数据汇总展示
* Kapacitor - 数据处理，比如监控策略等

**技术架构**



**安装和部署**

根据官方的开源解决方案，[安装和部署](https://portal.influxdata.com/downloads)都比较简单，可以用快捷安装方式即可。Chronograf建议编译安装，可以做一些基础的二次开发，部分[**React**](http://lib.csdn.net/base/react)的组件修改和做一下汉化等。不过由于[**Go**](http://lib.csdn.net/base/go)的依赖比较多，有一些包需要翻墙才能get到，我在这儿也是倒腾了蛮久，有点蛋疼 >.<，具体略，有需要可以沟通。(不过没啥聊的其实，就是折腾)

比如，如下是针对CentOS平台：

#1. influxdb

#1.1 下载并安装

wget https://dl.influxdata.com/influxdb/releases/influxdb-1.2.0.x86\_64.rpm

sudo yum localinstall influxdb-1.2.0.x86\_64.rpm

#1.2 修改配置文件(修改了data目录后，注意修改对应目录的权限，需要将influxdb用户设置为属主和属组)

#1.3 启动服务

service influxd start

#2. telegraf

#2.1 下载并安装

wget https://dl.influxdata.com/telegraf/releases/telegraf-1.2.1.x86\_64.rpm

sudo yum localinstall telegraf-1.2.1.x86\_64.rpm

#2.2 修改配置文件 /etc/telegraf/telegraf

#主要是修改influxdb的配置

#2.3 启动服务

service telegraf start

#3. kapacitor

#3.1 下载并安装

wget https://dl.influxdata.com/kapacitor/releases/kapacitor-1.2.0.x86\_64.rpm

sudo yum localinstall kapacitor-1.2.0.x86\_64.rpm

#3.2 修改配置文件 /etc/kapacitor/kapacitor.conf

#3.3 启动服务

#4. chronograf

# (源码编译安装略)

#4.1 下载并安装(快捷安装)

wget https://dl.influxdata.com/chronograf/releases/chronograf-1.2.0~beta3.x86\_64.rpm

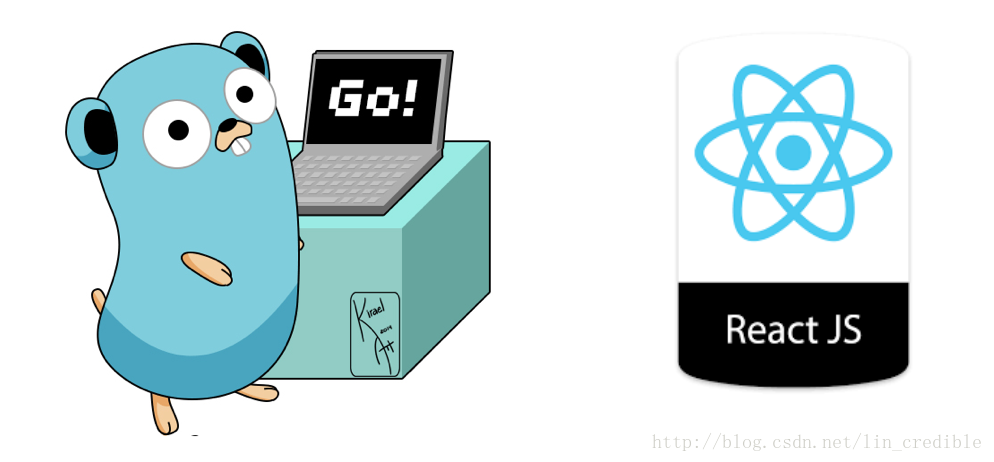
sudo yum localinstall chronograf-1.2.0~beta3.x86\_64.rpm

#4.2 启动服务

service chronograf start

都启动服务后，打开浏览器，访问: [http://chronograf-ip:8888](http://chronograf-ip:8888/) 即可访问

**二次开发**



二次开发，我这边主要针对 [Chronograf](https://github.com/linux-devops/chronograf) 做了一些微调，还需要进一步学习源码。有兴趣的朋友，并且熟悉go或reactjs的最好，可以一起学习该开源方案。

**可视效果**



**其它**



Chronograf 的开源版本有支持和github,google等账号体系打通，通过[JWT](https://tools.ietf.org/html/rfc7519)等，但是没有开放用户管理等功能，而且一般公司内部有账号体系，也不愿意将部分组织开放到github等上面，所以，要更好地在内部的环境使用它，目前来看，可能还需要进一步开发才行。不过，我个人倒是觉得 JWT 这种比较好，因为无状态，减少了维护成本。另外，后期肯定需要关联CMDB系统，关于这些后期的设计，如果有读者感兴趣，可以一起沟通和交流。欢迎指导！

**注意**

发现 telegraf 这个 agent 占 cpu 资源太多了，真是坑爹：

strace -T -r -c -p PID

% time seconds usecs/call calls errors syscall

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

99.12 16.050820 29451 545 12 futex

0.35 0.055932 3 19278 18790 openat

0.18 0.029826 36 838 read

0.09 0.014101 22 627 sched\_yield

0.07 0.012027 2405 5 clone

0.07 0.010825 71 153 select

0.03 0.004468 25 178 getdents64

0.03 0.004195 839 5 waitid

0.02 0.003915 8 496 stat

0.02 0.003291 6 507 close

0.01 0.002091 5 390 fstat

0.00 0.000366 46 8 1 readlinkat

0.00 0.000187 37 5 wait4

0.00 0.000166 21 8 lstat

0.00 0.000147 15 10 pipe2

0.00 0.000105 12 9 2 rt\_sigreturn

0.00 0.000101 8 12 statfs

0.00 0.000072 14 5 getpid

0.00 0.000030 30 1 sysinfo

0.00 0.000005 1 7 epoll\_wait

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

100.00 16.192670 23087 18805 total

# 大数据|监控 - 浅析时间序列数据 - 开源中国社区

http://www.oschina.net/question/2680454\_2191874

时间序列是一个在IT基础设施组件、物联网传感器的每个业务流程中以及在应用程序中功能强大的等待被解锁的强大武器。利用好它可以揭示可操作的趋势，模式，可变性，变化，共变，周期异常，异常和异常值率。在实践中，认识的时间序列数据可帮助您回答这样的问题：

* 基于访问者的行为给用户最好的反馈方式是什么，实时？
* 我该用什么样的模式可以让我在金融市场上执行更快速更智能的交易？
* 我可以预见到访客的停留时间以及为什么他们会离开么？
* 我能跟踪运输车队上面的传感器随着时间的推移，以优化交货的时间和燃油经济性？
* 我能否预测我的弹性基础设施能否承受住类似黑色星期五规模的事件？
* 我可以通过在我网络中PB级别数据传输随着时间的变化来检测我网络中的恶意模式么？
* 我可以根据环境条件实时的调整水，肥降低我的成本，增加我的作物产量么？

时序数据

用例

在InfluxData，最普遍使用的情况下，我们帮助到企业是自定义的监视体系，实时分析解决方案，物联网（IOT）和传感器数据管理系统，再加上的OpenStack，云，容器或虚拟基础架构监控互联网。

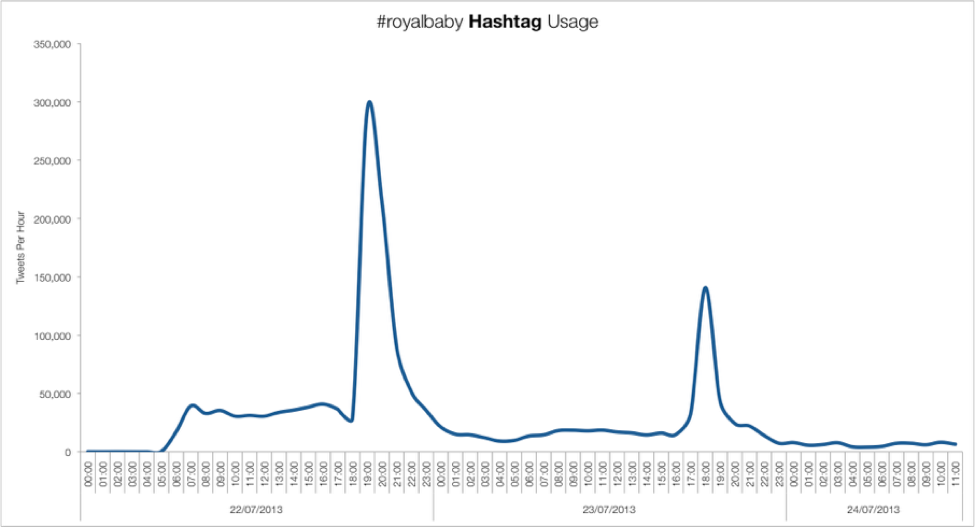
然而，时间序列数据限制为仅少数使用上述情况是不现实的。 InfluxData用户与我们分享种类繁多的通用和特殊的使用案例，例如：

* 异常检测
* 消息
* 个性化
* 股票交易
* 市政基础设施管理
* GPS服务
* 量子物理研究
* 销售点系统
* 制造与家庭自动化
* 运输和物资保障

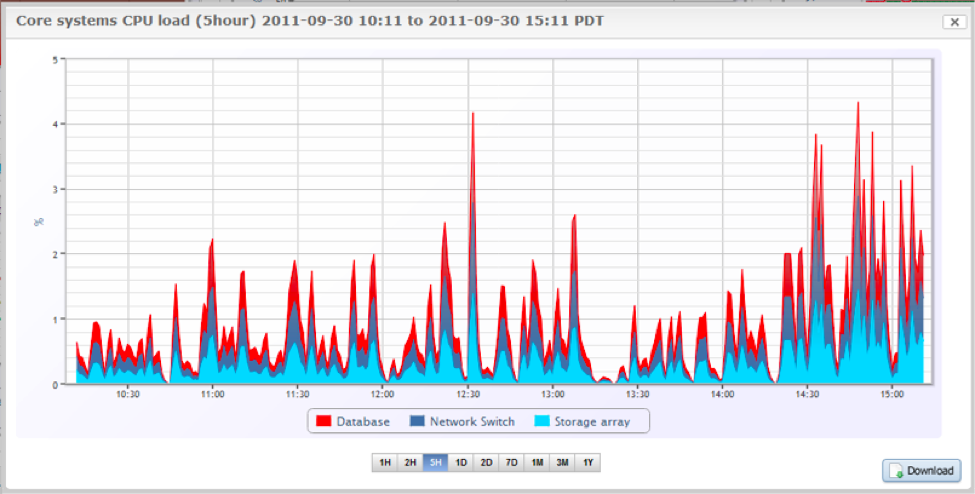
什么是时序数据

时间序列数据无非是基于时间的一序列数据点，更典型地由来自在一时间间隔相同的源制成连续测量的。换句话说，如果你要在图上画出你的观点，你的其中一个轴将永远是时间。例如，时间序列数据可通过像气象站或RFID标识，IT基础设施组件，如应用程序，服务器和网络交换机或通过股票交易系统的传感器来产生的。

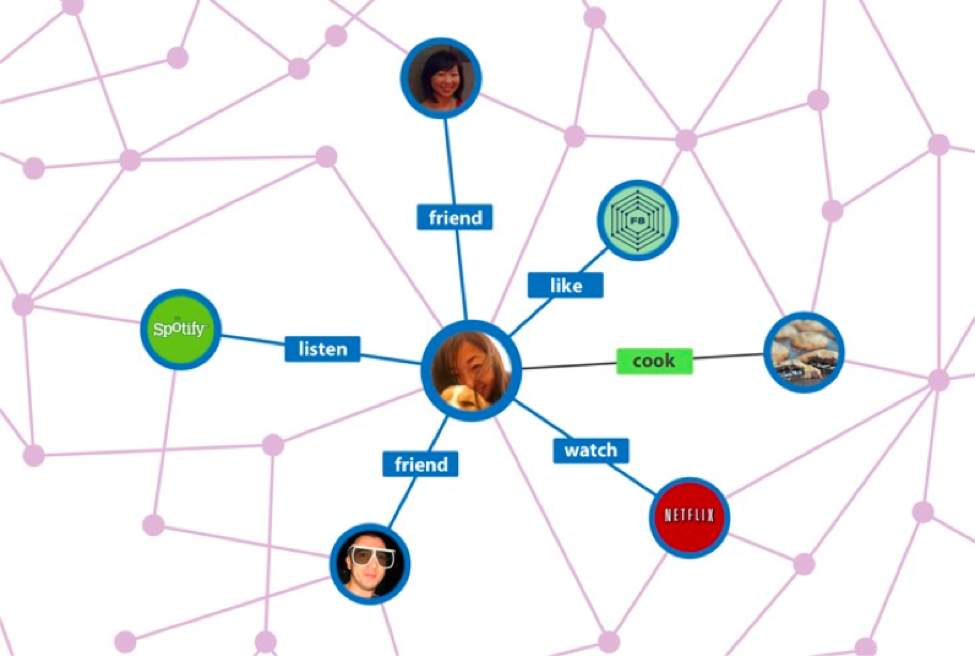
时间序列数据的图形示例

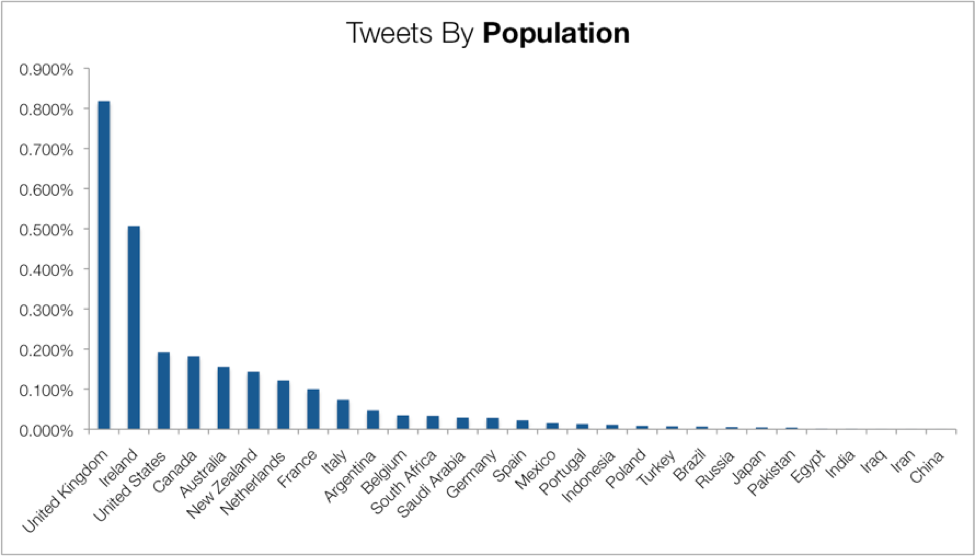






那些不是时序的数据集

这里有几个随时间变化的缺乏时间成分的数据集的几个例子：示出了人们之间的关系的数据，散点图，显示一个变量如何影响另一个，或示出了活动的给定体积比例的捐款的数据。因为他们缺乏时间的组件，我们不能认为他们“时间序列”的数据。 



时序的数据库

时序数据库优化了收集，存储，检索和处理时序数据。相对于文档数据库（优化了对json文档的存储），搜索数据库（优化了全文搜索）或者传统的关系型数据库（优化了关系型数据表格存储）

Baron Schwartz概述了一些专用时间序列数据库的典型特征应包括：

* 数据库90%以上的工作量是高频高容量的写入
* 写操作通常是随着时间追加到现有的表中
* 这些写操作通常是按一定时序的，例如：每秒或者每分钟
* 如果时间序列数据库中获取受限资源，这通常是因为它是I / O绑定
* 更新单个点数据的操作很少
* 删除数据几乎总是跨越大的时间范围（日，月或年）进行，几乎从不到一个特定的点
* 数据库查询操作通常是在某序列中有序的，可能是按时间排序或者按某功能排序
* 执行并行读取或者多组读取是常见的

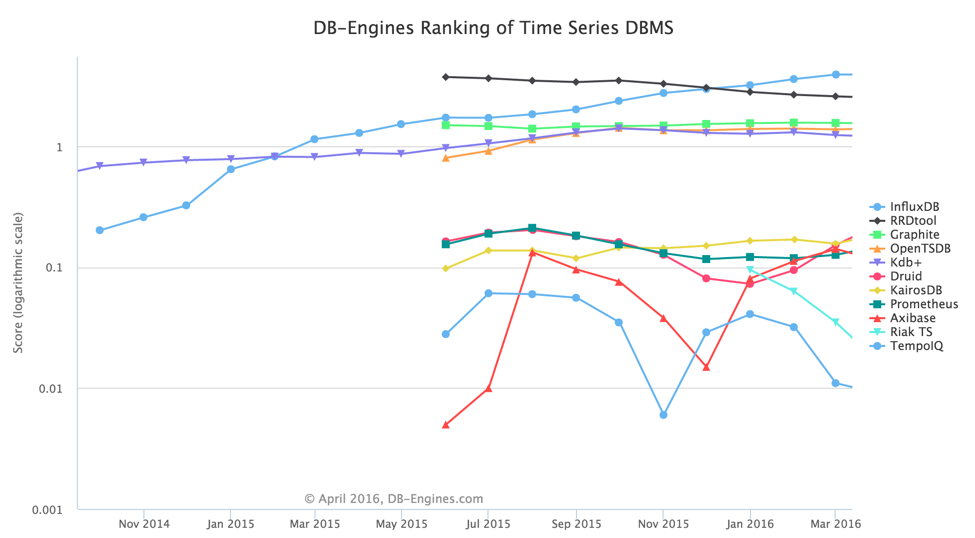
这里有一些开源的，对时序数据存储和处理都比较高效的时序数据库：

* [InfluxDb](https://github.com/influxdata/influxdb)
* [OpenTSDB](http://opentsdb.net/) (Runs on top of Hadoop and HBase)
* [KDB+](https://kx.com/software.php)
* [Graphite](https://github.com/graphite-project/)
* More examples on [Wikipedia](https://en.wikipedia.org/wiki/Time_series_database)

需要注意的是一些时间序列的产品和项目正在积极维护和开发，而另一些人很少能得到更新。我们建议把项目的社区发展/支持，发展速度和商业的支持来作为你选用项目的一些评价标准（如果你正在考虑该数据库是开源的。）

目前最火的时序数据库是什么？

截至2016年4月的，InfluxDB被[DB-engines.com](http://db-engines.com/en/)排名最流行的时间序列数据库。你可以在[这里](http://db-engines.com/en/ranking/time+series+dbms)获得最新排名。



开明的开发人员和架构师明白，最成功的项目都是那些无论公司组织的偏见最终选择了正确的工具的项目。InfluxDB已经帮助许多企业避免了很多额外的努力去整合一个时间序列数据库与其他正在使用的数据管理系统，与试图使一个“圆钉适合到一个方孔。”在接下来的几节中，我们将看看为什么企业通过采用专用数据库来管理自己的时间序列数据服务往往是更好的。

Time-Series vs ElasticSearch 和其他全文检索数据库

全文或“搜索”数据库是由文本文档，如书籍，报纸，学术论文或文字在网页上，或者日志中发现。为了说明时间序列和全文检索数据库之间的差异，我们将使用ElasticSearch作为一个例子。

ElasticSearch是用Java编写的一个开源的，全文搜索引擎，建在了Apache Lucene项目之上。这不是在传统意义上的数据库，这是一个JSON文档的数据存储。你可以把它和InfluxDB（一个开源的用Go实现的时序数据库）在表和标签方面做个比较。随着一些努力，ElasticSearch可以用有限的数学和分析功能，管理时间序列指标，但它是不是这个用例建的目的。对比InfluxDB来说它在单节点上你无法得到很好的性能和压缩能力。且不说像数据聚合，连续查询和保留策略的时间序列的特定功能。但是，InfluxDB和ElasticSearch很好地协同工作的一种方式是在日志由ElasticSearch和性能指标与用于数据结合共同的可视化工具，管理由InfluxDB系统（如Grafana）。

Time-Series vs Cassandra和其他混合/面向列的数据库

正如其名称所暗示的，面向列的数据库中组织数据基于列而不是行。面向列的数据库非常适合构建数据仓库、或者需要支持大量聚合的，需要计算非常大类似数据的即席查询的系统。为了说明时间序列和面向列数据库之间的差异，我们将使用Cassandra，这个在技术上是一个面向列的数据库为例。

[Cassandra](https://en.wikipedia.org/wiki/Apache_Cassandra)是一个用Java实现的开源的、分布式的基于键值对和面向列的混合型数据库。它在做了高容错的同事还保持着高性能的特性。与此相比，InFluxDB是开源的用Go写的用表和标签的形式来管理时序数据的时序数据库。

Cassandra是InfluxDB的界定“易于使用”特点的对立面。它有很多的依赖，配置，部署和管理都比较复杂。Cassandra需要用户编写一个查询计划，执行器和其它代码到应用程序来处理时间序列数据（即已经在InfluxDB中的功能。）最后，先把所有数据从系统中提取出来再执行查询，对比用InfluxDB直接在数据库内直接执行查询。你可以想像，对于非常大的数据集，这可能会对性能产生不良影响。

Time-Series vs MongoDB 和其他面向文档的数据库

[MongoDB](https://en.wikipedia.org/wiki/MongoDB)是用C++实现的开源的面向文档的数据库，它用动态模式优化了对类JSON（BSON）文档的管理。与此相比，InFluxDB是开源的用Go写的用表和标签的形式来管理时序数据的时序数据库。

开发人员经常用MongoDB去管理时序数据因为这是他们所熟悉的一种技术。毫无疑问，如果你想在MongoDB中管理时间序列数据，正确的模式设计至关重要。如果开发人员理解"一个Point(InfluxDB中的一条数据)等于一个文档"，那么他们很快就会遇到如何有效地管理大量的文档内存需求挑战。相反，尝试存储时序数据在一个文档中是艰巨的因为文件由它们的大小限制。此外，对子文档可用的查询功能也是有限的。并且只是用文档存储时序数据，开发人员肯定会遇到更糟糕的查询性能。最后，MongoDB的子文档中不能使用MongoDB的map-reduce功能，这也是MongoDB处理时序数据的缺憾。

Resources

* [Wikipedia:Time-Series Database](https://en.wikipedia.org/wiki/Time_series_database)
* [Time-Series Database Requirements by Baron Schwartz](http://www.xaprb.com/blog/2014/06/08/time-series-database-requirements/)
* [Thoughts on Time-Series Databases by Jason Moiron](http://jmoiron.net/blog/thoughts-on-timeseries-databases)
* [The Case for Tagging in Time-Series Data by Baron Schwartz](http://www.xaprb.com/blog/2015/10/16/time-series-tagging/)
* [InfluxDB Getting Started](https://docs.influxdata.com/influxdb/v0.12/introduction/getting_started/)
* [InfluxDB Key Concepts](https://docs.influxdata.com/influxdb/v0.11/concepts/key_concepts/)
* [Comparing Time-Series to SQL](https://docs.influxdata.com/influxdb/v0.12/concepts/crosswalk/)
* [Time-Series Database Design and Trade-Offs](https://docs.influxdata.com/influxdb/v0.12/concepts/insights_tradeoffs/)
* [Getting Started with Cassandra Time-Series Data Modeling](https://academy.datastax.com/resources/getting-started-time-series-data-modeling)
* [Schema Design for Time-Series Data in MongoDB](http://blog.mongodb.org/post/65517193370/schema-design-for-time-series-data-in-mongodb)
* [ElasticSearch as a Time-Series Data Store](https://www.elastic.co/blog/elasticsearch-as-a-time-series-data-store)

**作者信息**

原文链接：<https://influxdata.com/what-is-time-series-data/>

翻译自力谱宿云LeapCloud旗下MaxLeap团队\_数据分析组 成员：谭杨

**相关文章**

[微服务实战：从架构到发布（一）](https://blog.maxleap.cn/archives/195)

[微服务实战：从架构到发布（二）](https://blog.maxleap.cn/archives/218)

[移动云平台的基础架构之旅（一）：云应用](https://blog.maxleap.cn/archives/734)

[从应用到平台 – 云服务架构的演进过程](https://blog.maxleap.cn/archives/940)

**作者往期佳作**

[浅析Apache Spark Caching和Checkpointing](https://blog.maxleap.cn/archives/617)

# influxdata/influxdb: Scalable datastore for metrics, events, and real-time analytics

https://github.com/influxdata/influxdb

An Open-Source Time Series Database

InfluxDB is an open source **time series database** with **no external dependencies**. It's useful for recording metrics, events, and performing analytics.

Features

* Built-in [HTTP API](https://docs.influxdata.com/influxdb/latest/guides/writing_data/) so you don't have to write any server side code to get up and running.
* Data can be tagged, allowing very flexible querying.
* SQL-like query language.
* Simple to install and manage, and fast to get data in and out.
* It aims to answer queries in real-time. That means every data point is indexed as it comes in and is immediately available in queries that should return in < 100ms.

Installation

We recommend installing InfluxDB using one of the [pre-built packages](https://influxdata.com/downloads/#influxdb). Then start InfluxDB using:

* service influxdb start if you have installed InfluxDB using an official Debian or RPM package.
* systemctl start influxdb if you have installed InfluxDB using an official Debian or RPM package, and are running a distro with systemd. For example, Ubuntu 15 or later.
* $GOPATH/bin/influxd if you have built InfluxDB from source.

Getting Started

Create your first database

curl -XPOST 'http://localhost:8086/query' --data-urlencode "q=CREATE DATABASE mydb"

Insert some data

curl -XPOST 'http://localhost:8086/write?db=mydb' \

-d 'cpu,host=server01,region=uswest load=42 1434055562000000000'

curl -XPOST 'http://localhost:8086/write?db=mydb' \

-d 'cpu,host=server02,region=uswest load=78 1434055562000000000'

curl -XPOST 'http://localhost:8086/write?db=mydb' \

-d 'cpu,host=server03,region=useast load=15.4 1434055562000000000'

Query for the data

curl -G http://localhost:8086/query?pretty=true --data-urlencode "db=mydb" \

--data-urlencode "q=SELECT \* FROM cpu WHERE host='server01' AND time < now() - 1d"

Analyze the data

curl -G http://localhost:8086/query?pretty=true --data-urlencode "db=mydb" \

--data-urlencode "q=SELECT mean(load) FROM cpu WHERE region='uswest'"

Documentation

* Read more about the [design goals and motivations of the project](https://docs.influxdata.com/influxdb/latest/).
* Follow the [getting started guide](https://docs.influxdata.com/influxdb/latest/introduction/getting_started/) to learn the basics in just a few minutes.
* Learn more about [InfluxDB's key concepts](https://docs.influxdata.com/influxdb/latest/guides/writing_data/).

Contributing

If you're feeling adventurous and want to contribute to InfluxDB, see our [contributing doc](https://github.com/influxdata/influxdb/blob/master/CONTRIBUTING.md) for info on how to make feature requests, build from source, and run tests.

Looking for Support?

InfluxDB offers a number of services to help your project succeed. We offer Developer Support for organizations in active development, Managed Hosting to make it easy to move into production, and Enterprise Support for companies requiring the best response times, SLAs, and technical fixes. Visit our [support page](https://influxdata.com/services/) or contact [sales@influxdb.com](mailto:sales@influxdb.com) to learn how we can best help you succeed.

# InfluxData | Documentation | Key Concepts

https://docs.influxdata.com/influxdb/v1.2/concepts/key\_concepts/#field-value

Before diving into InfluxDB it’s good to get acquainted with some of the key concepts of the database. This document provides a gentle introduction to those concepts and common InfluxDB terminology. We’ve provided a list below of all the terms we’ll cover, but we recommend reading this document from start to finish to gain a more general understanding of our favorite time series database.

|  |  |  |
| --- | --- | --- |
| [**database**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#database) | [**field key**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#field-key) | [**field set**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#field-set) |
| [**field value**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#field-value) | [**measurement**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#measurement) | [**point**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#point) |
| [**retention policy**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#retention-policy) | [**series**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#series) | [**tag key**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#tag-key) |
| [**tag set**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#tag-set) | [**tag value**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#tag-value) | [**timestamp**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#timestamp) |

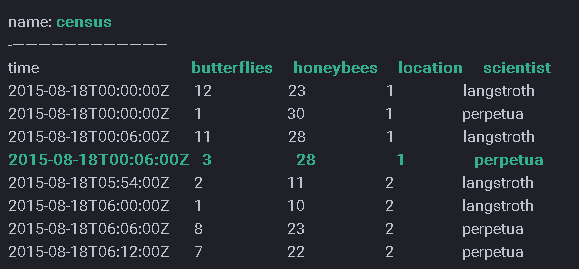
Check out the [**Glossary**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/) if you prefer the cold, hard facts.

[**Sample data**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#sample-data)

The next section references the data printed out below. The data are fictional, but represent a believable setup in InfluxDB. They show the number of butterflies and honeybees counted by two scientists (langstroth andperpetua) in two locations (location 1 and location 2) over the time period from August 18, 2015 at midnight through August 18, 2015 at 6:12 AM. Assume that the data live in a database called my\_database and are subject to the autogen retention policy (more on databases and retention policies to come).

Hint: Hover over the links for tooltips to get acquainted with InfluxDB terminology and the layout.

name:



[**Discussion**](https://docs.influxdata.com/influxdb/v1.2/concepts/key_concepts/#discussion)

Now that you’ve seen some sample data in InfluxDB this section covers what it all means.

InfluxDB is a time series database so it .makes sense to start with what is at the root of everything we do: time. In the data above there’s a column called time - all data in InfluxDB have that column. time stores timestamps, and the timestamp shows the date and time, in [**RFC3339**](https://www.ietf.org/rfc/rfc3339.txt) UTC, associated with particular data.

The next two columns, called butterflies and honeybees, are fields. Fields are made up of field keys and field values. Field keys (butterflies and honeybees) are strings and they store metadata; the field key butterfliestells us that the field values 12-7 refer to butterflies and the field key honeybees tells us that the field values23-22 refer to, well, honeybees.

Field values are your data; they can be strings, floats, integers, or booleans, and, because InfluxDB is a time series database, a field value is always associated with a timestamp. The field values in the sample data are:

12 23

1 30

11 28

3 28

2 11

1 10

8 23

7 22

In the data above, the collection of field-key and field-value pairs make up a field set. Here are all eight field sets in the sample data:

* butterflies = 12 honeybees = 23
* butterflies = 1 honeybees = 30
* butterflies = 11 honeybees = 28
* butterflies = 3 honeybees = 28
* butterflies = 2 honeybees = 11
* butterflies = 1 honeybees = 10
* butterflies = 8 honeybees = 23
* butterflies = 7 honeybees = 22

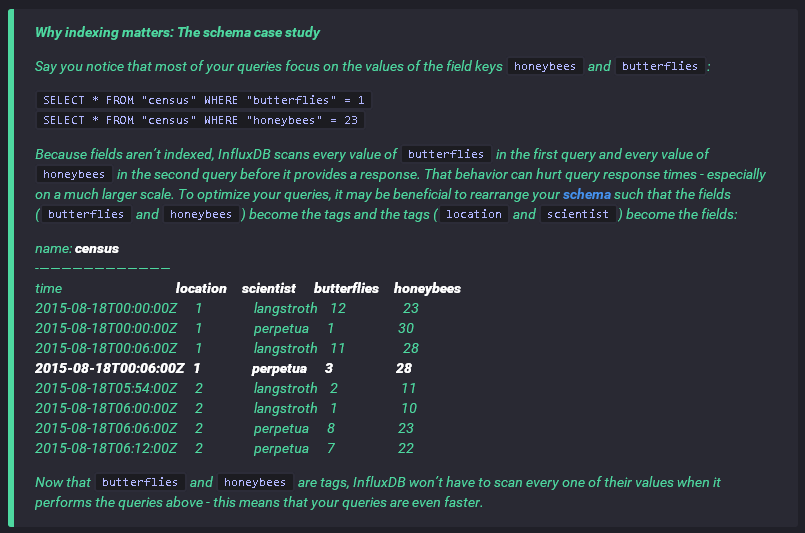
Fields are a required piece of InfluxDB’s data structure - you cannot have data in InfluxDB without fields. It’s also important to note that fields are not indexed. [**Queries**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#query) that use field values as filters must scan all values that match the other conditions in the query. As a result, those queries are not performant relative to queries on tags (more on tags below). In general, fields should not contain commonly-queried metadata.

The last two columns in the sample data, called location and scientist, are tags. Tags are made up of tag keys and tag values. Both tag keys and tag values are stored as strings and record metadata. The tag keys in the sample data are location and scientist. The tag key location has two tag values: 1 and 2. The tag keyscientist also has two tag values: langstroth and perpetua.

In the data above, the tag set is the different combinations of all the tag key-value pairs. The four tag sets in the sample data are:

* location = 1, scientist = langstroth
* location = 2, scientist = langstroth
* location = 1, scientist = perpetua
* location = 2, scientist = perpetua

Tags are optional. You don’t need to have tags in your data structure, but it’s generally a good idea to make use of them because, unlike fields, tags are indexed. This means that queries on tags are faster and that tags are ideal for storing commonly-queried metadata.



The measurement acts as a container for tags, fields, and the time column, and the measurement name is the description of the data that are stored in the associated fields. Measurement names are strings, and, for any SQL users out there, a measurement is conceptually similar to a table. The only measurement in the sample data is census. The name census tells us that the field values record the number of butterflies andhoneybees - not their size, direction, or some sort of happiness index.

A single measurement can belong to different retention policies. A retention policy describes how long InfluxDB keeps data (DURATION) and how many copies of those data are stored in the cluster (REPLICATION). If you’re interested in reading more about retention policies, check out [**Database Management**](https://docs.influxdata.com/influxdb/v1.2/query_language/database_management/#retention-policy-management).

Replication factors do not serve a purpose with single node instances.

In the sample data, everything in the census measurement belongs to the autogen retention policy. InfluxDB automatically creates that retention policy; it has an infinite duration and a replication factor set to one.

Now that you’re familiar with measurements, tag sets, and retention policies it’s time to discuss series. In InfluxDB, a series is the collection of data that share a retention policy, measurement, and tag set. The data above consist of four series:

| **Arbitrary series number** | **Retention policy** | **Measurement** | **Tag set** |
| --- | --- | --- | --- |
| series 1 | autogen | census | location = 1,scientist = langstroth |
| series 2 | autogen | census | location = 2,scientist = langstroth |
| series 3 | autogen | census | location = 1,scientist = perpetua |
| series 4 | autogen | census | location = 2,scientist = perpetua |

Understanding the concept of a series is essential when designing your [**schema**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#schema) and when working with your data in InfluxDB.

Finally, a point is the field set in the same series with the same timestamp. For example, here’s a single point:

name: census

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

time butterflies honeybees location scientist

2015-08-18T00:00:00Z 1 30 1 perpetua

The series in the example is defined by the retention policy (autogen), the measurement (census), and the tag set (location = 1, scientist = perpetua). The timestamp for the point is 2015-08-18T00:00:00Z.

All of the stuff we’ve just covered is stored in a database - the sample data are in the database my\_database. An InfluxDB database is similar to traditional relational databases and serves as a logical container for users, retention policies, continuous queries, and, of course, your time series data. See [**Authentication and Authorization**](https://docs.influxdata.com/influxdb/v1.2/query_language/authentication_and_authorization/) and [**Continuous Queries**](https://docs.influxdata.com/influxdb/v1.2/query_language/continuous_queries/) for more on those topics.

Databases can have several users, continuous queries, retention policies, and measurements. InfluxDB is a schemaless database which means it’s easy to add new measurements, tags, and fields at any time. It’s designed to make working with time series data awesome.

You made it! You’ve covered the fundamental concepts and terminology in InfluxDB. If you’re just starting out, we recommend taking a look at [**Getting Started**](https://docs.influxdata.com/influxdb/v1.2/introduction/getting_started/) and the [**Writing Data**](https://docs.influxdata.com/influxdb/v1.2/guides/writing_data/) and [**Querying Data**](https://docs.influxdata.com/influxdb/v1.2/guides/querying_data/) guides. May our time series database serve you well 🕔.

 This documentation is [**open source**](https://github.com/influxdata/docs.influxdata.com/blob/master/content/influxdb/v1.2/concepts/key_concepts.md). See a typo? Please, open an [**issue**](https://github.com/influxdata/docs.influxdata.com/issues/new).

**Need help getting up and running?** [**Get Support**](https://influxdata.com/pricing/#product-subscriptions)

# InfluxData | Documentation | Comparison to SQL

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

What’s in a database?

This page gives SQL users an overview of how InfluxDB is like an SQL database and how it’s not. It highlights some of the major distinctions between the two and provides a loose crosswalk between the different database terminologies and query languages.

[In general…](https://docs.influxdata.com/influxdb/v1.2/concepts/crosswalk/#in-general)

InfluxDB is designed to work with time-series data. SQL databases can handle time-series but weren’t created strictly for that purpose. In short, InfluxDB is made to store a large volume of time-series data and perform real-time analysis on those data, quickly.

[**Timing is everything**](https://docs.influxdata.com/influxdb/v1.2/concepts/crosswalk/#timing-is-everything)

In InfluxDB, a timestamp identifies a single point in any given data series. This is like an SQL database table where the primary key is pre-set by the system and is always time.

InfluxDB also recognizes that your [**schema**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#schema) preferences may change over time. In InfluxDB you don’t have to define schemas up front. Data points can have one of the fields on a measurement, all of the fields on a measurement, or any number in-between. You can add new fields to a measurement simply by writing a point for that new field. If you need an explanation of the terms measurements, tags, and fields check out the next section for an SQL database to InfluxDB terminology crosswalk.

[Terminology](https://docs.influxdata.com/influxdb/v1.2/concepts/crosswalk/#terminology)

The table below is a (very) simple example of a table called foodships in an SQL database with the unindexed column #\_foodships and the indexed columns park\_id, planet, and time.

+---------+---------+---------------------+--------------+

| park\_id | planet | time | #\_foodships |

+---------+---------+---------------------+--------------+

| 1 | Earth | 1429185600000000000 | 0 |

| 1 | Earth | 1429185601000000000 | 3 |

| 1 | Earth | 1429185602000000000 | 15 |

| 1 | Earth | 1429185603000000000 | 15 |

| 2 | Saturn | 1429185600000000000 | 5 |

| 2 | Saturn | 1429185601000000000 | 9 |

| 2 | Saturn | 1429185602000000000 | 10 |

| 2 | Saturn | 1429185603000000000 | 14 |

| 3 | Jupiter | 1429185600000000000 | 20 |

| 3 | Jupiter | 1429185601000000000 | 21 |

| 3 | Jupiter | 1429185602000000000 | 21 |

| 3 | Jupiter | 1429185603000000000 | 20 |

| 4 | Saturn | 1429185600000000000 | 5 |

| 4 | Saturn | 1429185601000000000 | 5 |

| 4 | Saturn | 1429185602000000000 | 6 |

| 4 | Saturn | 1429185603000000000 | 5 |

+---------+---------+---------------------+--------------+

Those same data look like this in InfluxDB:

name: foodships

tags: park\_id=1, planet=Earth

time #\_foodships

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

2015-04-16T12:00:00Z 0

2015-04-16T12:00:01Z 3

2015-04-16T12:00:02Z 15

2015-04-16T12:00:03Z 15

name: foodships

tags: park\_id=2, planet=Saturn

time #\_foodships

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

2015-04-16T12:00:00Z 5

2015-04-16T12:00:01Z 9

2015-04-16T12:00:02Z 10

2015-04-16T12:00:03Z 14

name: foodships

tags: park\_id=3, planet=Jupiter

time #\_foodships

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

2015-04-16T12:00:00Z 20

2015-04-16T12:00:01Z 21

2015-04-16T12:00:02Z 21

2015-04-16T12:00:03Z 20

name: foodships

tags: park\_id=4, planet=Saturn

time #\_foodships

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

2015-04-16T12:00:00Z 5

2015-04-16T12:00:01Z 5

2015-04-16T12:00:02Z 6

2015-04-16T12:00:03Z 5

Referencing the example above, in general:

* An InfluxDB measurement (foodships) is similar to an SQL database table.
* InfluxDB tags ( park\_id and planet) are like indexed columns in an SQL database.
* InfluxDB fields (#\_foodships) are like unindexed columns in an SQL database.
* InfluxDB points (for example, 2015-04-16T12:00:00Z 5) are similar to SQL rows.

Building on this comparison of database terminology, InfluxDB’s [**continuous queries**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#continuous-query-cq) and [**retention policies**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#retention-policy-rp) are similar to stored procedures in an SQL database. They’re specified once and then performed regularly and automatically.

Of course, there are some major disparities between SQL databases and InfluxDB. SQL JOINs aren’t available for InfluxDB measurements; your schema design should reflect that difference. And, as we mentioned above, a measurement is like an SQL table where the primary index is always pre-set to time. InfluxDB timestamps must be in UNIX epoch (GMT) or formatted as a date-time string valid under RFC3339.

For more detailed descriptions of the InfluxDB terms mentioned in this section see our [**Glossary of Terms**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/).

[InfluxQL and SQL](https://docs.influxdata.com/influxdb/v1.2/concepts/crosswalk/#influxql-and-sql)

InfluxQL is an SQL-like query language for interacting with InfluxDB. It has been lovingly crafted to feel familiar to those coming from other SQL or SQL-like environments while also providing features specific to storing and analyzing time series data.

InfluxQL’s SELECT statement follows the form of an SQL SELECT statement:

SELECT <stuff> FROM <measurement\_name> WHERE <some\_conditions>

where WHERE is optional. To get the InfluxDB output in the section above, you’d enter:

SELECT \* FROM "foodships"

If you only wanted to see data for the planet Saturn, you’d enter:

SELECT \* FROM "foodships" WHERE "planet" = 'Saturn'

If you wanted to see data for the planet Saturn after 12:00:01 UTC on April 16, 2015, you’d enter:

SELECT \* FROM "foodships" WHERE "planet" = 'Saturn' AND time > '2015-04-16 12:00:01'

As shown in the example above, InfluxQL allows you to specify the time range of your query in the WHEREclause. You can use date-time strings wrapped in single quotes that have the format YYYY-MM-DD HH:MM:SS.mmm (mmm is milliseconds and is optional, and you can also specify microseconds or nanoseconds). You can also use relative time with now() which refers to the server’s current timestamp:

SELECT \* FROM "foodships" WHERE time > now() - 1h

That query outputs the data in the foodships measure where the timestamp is newer than the server’s current time minus one hour. The options for specifying time durations with now() are:

| **Letter** | **Meaning** |
| --- | --- |
| u or µ | microseconds |
| ms | milliseconds |
| s | seconds |
| m | minutes |
| h | hours |
| d | days |
| w | weeks |

InfluxQL also supports regular expressions, arithmetic in expressions, SHOW statements, and GROUP BYstatements. See our [**data exploration**](https://docs.influxdata.com/influxdb/v1.2/query_language/data_exploration/) page for an in-depth discussion of those topics. InfluxQL functions includeCOUNT, MIN, MAX, MEDIAN, DERIVATIVE and more. For a full list check out the [**functions**](https://docs.influxdata.com/influxdb/v1.2/query_language/functions/) page.

Now that you have the general idea, check out our [**Getting Started Guide**](https://docs.influxdata.com/influxdb/v1.2/introduction/getting_started/).

[A note on why InfluxDB isn’t CRUD…](https://docs.influxdata.com/influxdb/v1.2/concepts/crosswalk/#a-note-on-why-influxdb-isn-t-crud)

InfluxDB is a database that has been optimized for time series data. This data commonly comes from sources like distributed sensor groups, click data from large websites, or lists of financial transactions.

One thing this data has in common is that it is more useful in the aggregate. One reading saying that your computer’s CPU is at 12% utilization at 12:38:35 UTC on a Tuesday is hard to draw conclusions from. It becomes more useful when combined with the rest of the series and visualized. This is where trends over time begin to show, and actionable insight can be drawn from the data. In addition, time series data is generally written once and rarely updated.

The result is that InfluxDB is not a full CRUD database but more like a CR-ud, prioritizing the performance of creating and reading data over update and destroy, and preventing some update and destroy behaviors to make create and read more performant.

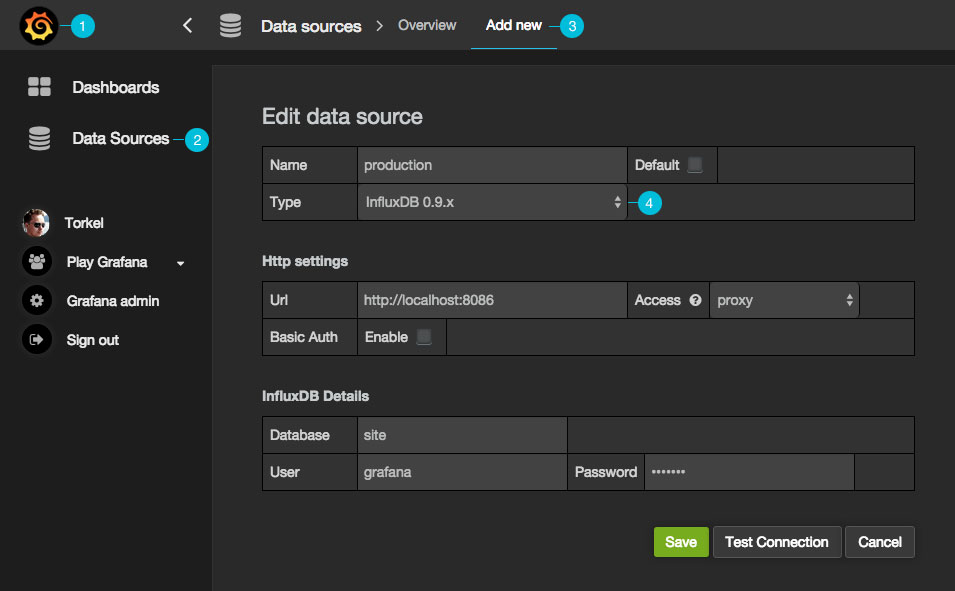
# Using InfluxDB in Grafana | Grafana Documentation

http://docs.grafana.org/features/datasources/influxdb/

Using InfluxDB in Grafana

Grafana ships with very feature rich data source plugin for InfluxDB. Supporting a feature rich query editor, annotation and templating queries.

**Adding the data source**



1. Open the side menu by clicking the the Grafana icon in the top header.
2. In the side menu under the Dashboards link you should find a link named Data Sources.

NOTE: If this link is missing in the side menu it means that your current user does not have theAdmin role for the current organization.

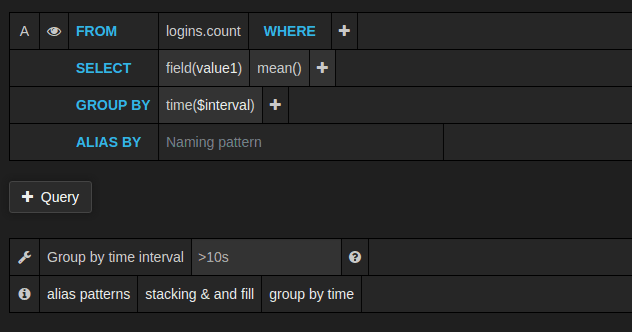
1. Click the Add new link in the top header.

| **Name** | **Description** |
| --- | --- |
| Name | The data source name, important that this is the same as in Grafana v1.x if you plan to import old dashboards. |
| Default | Default data source means that it will be pre-selected for new panels. |
| Url | The http protocol, ip and port of you influxdb api (influxdb api port is by default 8086) |
| Access | Proxy = access via Grafana backend, Direct = access directly from browser. |
| Database | Name of your influxdb database |
| User | Name of your database user |
| Password | Database user’s password |

Proxy access means that the Grafana backend will proxy all requests from the browser, and send them on to the Data Source. This is useful because it can eliminate CORS (Cross Origin Site Resource) issues, as well as eliminate the need to disseminate authentication details to the Data Source to the browser.

Direct access is still supported because in some cases it may be useful to access a Data Source directly depending on the use case and topology of Grafana, the user, and the Data Source.

**Query Editor**



You find the InfluxDB editor in the metrics tab in Graph or Singlestat panel’s edit mode. You enter edit mode by clicking the panel title, then edit. The editor allows you to select metrics and tags.

**Filter data (WHERE)**

To add a tag filter click the plus icon to the right of the WHERE condition. You can remove tag filters by clicking on the tag key and select --remove tag filter--.

**Regex matching**

You can type in regex patterns for metric names or tag filter values, be sure to wrap the regex pattern in forward slashes (/). Grafana will automatically adjust the filter tag condition to use the InfluxDB regex match condition operator (=~).

**Field & Aggregation functions**

In the SELECT row you can specify what fields and functions you want to use. If you have a group by time you need an aggregation function. Some functions like derivative require an aggregation function.

The editor tries simplify and unify this part of the query. For example: http://docs.grafana.org/img/docs/influxdb/select_editor.png

The above will generate the following InfluxDB SELECT clause:

SELECT derivative(mean("value"), 10s) /10 AS "REQ/s" FROM ....

**Select multiple fields**

Use the plus button and select Field > field to add another SELECT clause. You can also specify an asterix \* to select all fields.

**Group By**

To group by a tag click the plus icon at the end of the GROUP BY row. Pick a tag from the dropdown that appears. You can remove the group by by clicking on the tag and then click on the x icon.

**Text Editor Mode (RAW)**

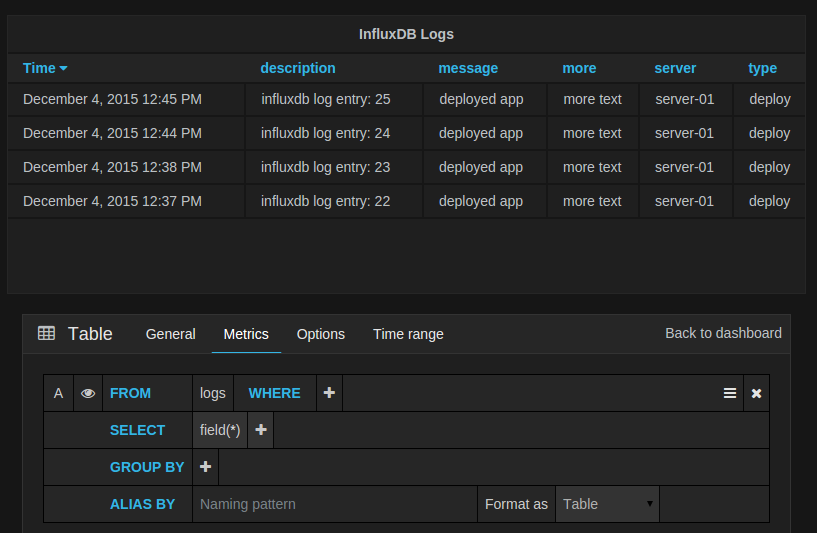
You can switch to raw query mode by clicking hamburger icon and then Switch editor mode.

If you use Raw Query be sure your query at minimum have WHERE $timeFilter Also please always have a group by time and an aggregation function, otherwise InfluxDB can easily return hundreds of thousands of data points that will hang the browser.

**Alias patterns**

* $m = replaced with measurement name
* $measurement = replaced with measurement name
* $col = replaced with column name
* $tag\_hostname = replaced with the value of the hostname tag
* You can also use [[tag\_hostname]] pattern replacement syntax

**Table query / raw data**



You can remove the group by time by clicking on the time part and then the x icon. You can change the optionFormat As to Table if you want to show raw data in the Table panel.

**Templating**

You can create a template variable in Grafana and have that variable filled with values from any InfluxDB metric exploration query. You can then use this variable in your InfluxDB metric queries.

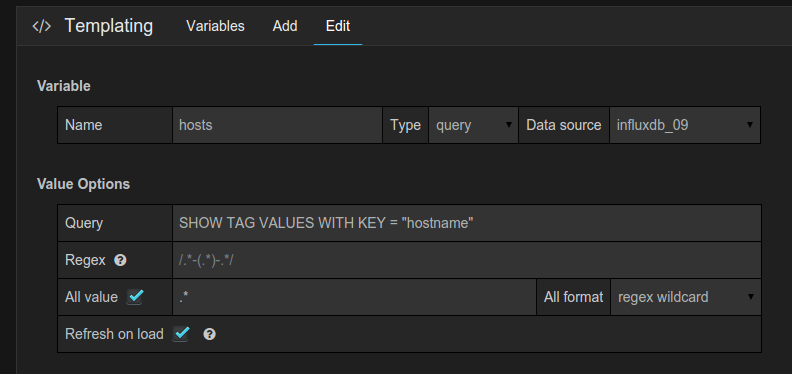
For example you can have a variable that contains all values for tag hostname if you specify a query like this in the templating edit view.

SHOW TAG VALUES WITH KEY = "hostname"

You can also create nested variables. For example if you had another variable, for example region. Then you could have the hosts variable only show hosts from the current selected region with a query like this:

SHOW TAG VALUES WITH KEY = "hostname" WHERE region =~ /$region/

Always use regex values or regex wildcard for All format or multi select format.



**Annotations**

Annotations allows you to overlay rich event information on top of graphs.

An example query:

SELECT title, description from events WHERE $timeFilter order asc

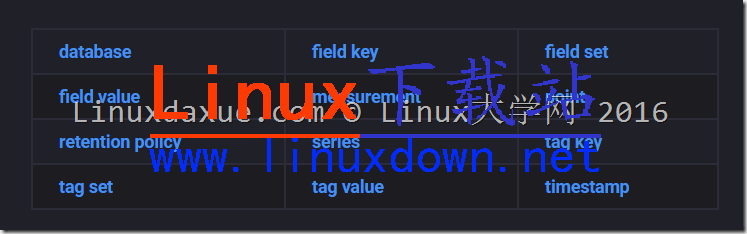
# InfluxDB学习之InfluxDB的关键概念

http://www.linuxdown.net/install/faq/20161025\_how\_linux\_8680.html

一、基本概念

1）database--数据库，这个同传统数据库的数据库概念。

2）measurement--数据表，在InfluxDB中，measurement即为表的作用，同传统数据库中的table作用一致。

[](http://cdn.linuxdaxue.com/wp-content/uploads/2016/07/image-3.png)

二、InfluxDB特有概念

1）tag--标签，在InfluxDB中，tag是一个非常重要的部分，表名+tag一起作为数据库的索引，是“key-value”的形式。

2）field--数据，field主要是用来存放数据的部分，也是“key-value”的形式。

3）timestamp--时间戳，作为时序型数据库，时间戳是InfluxDB中最重要的部分，在插入数据时可以自己指定也可留空让系统指定。

说明：在插入新数据时，tag、field和timestamp之间用空格分隔。

4）series--序列，所有在数据库中的数据，都需要通过图表来展示，而这个series表示这个表里面的数据，可以在图表上画成几条线。

5）Retention policy--数据保留策略，可以定义数据保留的时长，每个数据库可以有多个数据保留策略，但只能有一个默认策略。。

6）Point--点，表示每个表里某个时刻的某个条件下的一个field的数据，因为体现在图表上就是一个点，于是将其称为point。

# influxdb 安装配置详解 - 康建华 - 51CTO技术博客

http://michaelkang.blog.51cto.com/1553154/1759865

1：简介

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

1、它有三大特性：

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

2. Metrics（度量）：你可以实时对大量数据进行计算

3. Eevents（事件）：它支持任意的事件数据

2、特点

schemaless(无结构)，可以是任意数量的列

Scalable

min, max, sum, count, mean, median 一系列函数，方便统计

Native HTTP API, 内置http支持，使用http读写

Powerful Query Language 类似sql

Built-in Explorer 自带管理工具

3、API

InfluxDB 支持两种api方式

HTTP API

Protobuf API

2：安装部署

添加yum 源

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

[influxdb]

name = InfluxDB Repository - RHEL \$releasever

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

enabled = 1

gpgcheck = 1

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

EOF

3：安装

 yum install influxdb -y

4：启动服务、添加开机启动

 service influxdb start

systemctl enable  influxdb

service influxdb status

5：服务默认使用端口：

Networking By default, InfluxDB uses the following network ports:

TCP port 8083 is used for InfluxDB’s Admin panel

TCP port 8086 is used for client-server communication over InfluxDB’s HTTP API

TCP ports 8088 and 8091 are required for clustered InfluxDB instances

6：服务验证

输入

influx

进入数据库

[root@ctn-7-12 ~]# influx

Visit https://enterprise.influxdata.com to register for updates, InfluxDB server management, and monitoring.

Connected to http://localhost:8086 version 0.11.0

InfluxDB shell 0.11.0

1、创建一个查询用户

CREATE USER "ptquery" WITH PASSWORD 'ptquery'

2、查看用户：

> show users;

user    admin

ptquery false

ptdb1   fals

3、也可以在页面创建查询用户

CREATE USER "ptquery" WITH PASSWORD 'ptquery'

4、查看服务端口

[root@ctn-7-12 ~]# netstat -tunlp

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name

tcp6       0      0 :::8083                 :::\*                    LISTEN      22124/influxd

tcp6       0      0 :::8086                 :::\*                    LISTEN      22124/influxd

4、浏览器访问数据库管理平台：

http://172.16.7.11:8083/

参考信息：

https://influxdata.com/downloads/

# influxdb 端口、数据结构、写数据 - 蝈蝈俊 - 博客园

http://www.cnblogs.com/ghj1976/p/4095073.html

InfluxDB 是一个开源，分布式，时间序列，事件，可度量和无外部依赖的数据库。

InfluxDB有三大特性：

1. Time Series （时间序列）：你可以使用与时间有关的相关函数（如最大，最小，求和等）
2. Metrics（度量）：你可以实时对大量数据进行计算
3. Events（事件）：它支持任意的事件数据

端口暴露

influxdb会监听4个端口：

tcp        0      0 0.0.0.0:8099                0.0.0.0:\*                   LISTEN      29458/influxdb  
tcp        0      0 0.0.0.0:8083                0.0.0.0:\*                   LISTEN      29458/influxdb  
tcp        0      0 0.0.0.0:8086                0.0.0.0:\*                   LISTEN      29458/influxdb  
tcp        0      0 0.0.0.0:8090                0.0.0.0:\*                   LISTEN      29458/influxdb

其中单机使用只需要用到两个，另外两个是分布式部署时采用的

* 8083  WEB 管理界面
* 8086 HTTP API 接口服务

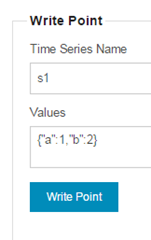
数据结构

在 influxdb 中 database 、 series、point 分别类似于数据库系统中的 数据库、表、列的概念。

所有的数据项在创建时都会自动增加两个字段：

* time 数据创建的时间，时间戳类型
* sequence\_number 字段是 influxdb 数据库维护的，类似于mysql的 主键概念。

比如我们用下面数据创建一个 Series

[](http://images.cnitblog.com/blog/120296/201411/131524393509305.png)

他就会产生下面数据格式的数据存储：

[](http://images.cnitblog.com/blog/120296/201411/131524441631111.png)

这里可以看到，系统自动增加了2个字段： time 和 sequence\_number 。

接口协议

InfluxDB 支持两种api方式：

* HTTP API ，已经提供
* Protobuf API， 计划提供

如何使用 http api 进行操作？

比如对于foo\_production这个数据库，插入一系列数据，可以发现POST 请求到 /db/foo\_production/series?u=some\_user&p=some\_password, 数据放到body里。

数据看起来是这样的:

下面的"name": "events", 其中"events"就是一个series,类似关系型数据库的表table

[

{

"name": "events",

"columns": ["state", "email", "type"],

"points": [

["ny", "paul@influxdb.org", "follow"],

["ny", "todd@influxdb.org", "open"]

]

},

{

"name": "errors",

"columns": ["class", "file", "user", "severity"],

"points": [

["DivideByZero", "example.py", "someguy@influxdb.org", "fatal"]

]

}

]

格式是json，可以在一个POST请求发送多个 series, 每个 series 里的 points 可以是多个，但索引要和columns对应。

上面的数据里没有包含time 列，InfluxDB会自己加上，不过也可以指定time,比如：

[

{

"name": "response\_times",

"columns": ["time", "value"],

"points": [

[1382819388, 234.3],

[1382819389, 120.1],

[1382819380, 340.9]

]

}

]

time 在InfluxDB里是很重要的，毕竟InfluxDB是time series database  
在InfluxDB里还有个sequence\_number字段是数据库维护的，类似于mysql的 主键概念

当然 sequence\_number 也是可以指定的，类似如下：

[

{

"name": "log\_lines",

"columns": ["time", "sequence\_number", "line"],

"points": [

[1400425947368, 1, "this line is first"],

[1400425947368, 2, "and this is second"]

]

}

]

<http://influxdb.com/docs/v0.8/api/reading_and_writing_data.html>

InfluxDB 增删更查都是用http api来完成，甚至支持使用正则表达式删除数据，还有计划任务。

比如:

发送POST请求到 /db/:name/scheduled\_deletes， body如下，

{

"regex": "stats\..\*",

"olderThan": "14d",

"runAt": 3

}

这个查询会删除大于14天的数据，并且任何以stats开头的数据，并且每天3:00 AM运行。

**参考资料：**

InfluxDB 开源分布式时序、事件和指标数据库  
<http://segmentfault.com/blog/lds/1190000000444617>

开源监控利器grafana  
<http://www.cnblogs.com/txwsqk/p/3974915.html>

# InfluxDB系列教程 | Linux大学

http://www.linuxdaxue.com/series/influxdb-series/

## InfluxDB学习之InfluxDB的HTTP API写入操作 | Linux大学

http://www.linuxdaxue.com/influxdb-write-data-by-http-api.html

一、说明

为了方便，本文主要使用curl来发起http请求，示例当中也是使用curl这个工具来模拟HTTP 请求。

在实际使用中，可以将请求写入代码中，通过其他编程语言来模拟HTTP请求。

二、InfluxDB通过HTTP API操作数据库

1）建立数据库

curl -POST http://localhost:**8086**/query --data-urlencode "q=CREATE DATABASE mydb"

执行这个语句后，会在本地建立一个名为mydb的数据库。

2）[删除](http://www.linuxdaxue.com/tag/%e5%88%a0%e9%99%a4/)数据库

curl -POST http://localhost:**8086**/query --data-urlencode "q=DROP DATABASE mydb"

其实使用HTTP API就是向 InfluxDB 接口发送相应的POST请求。

将语句通过POST方式发送到[服务](http://www.linuxdaxue.com/tag/%e6%9c%8d%e5%8a%a1/)器。

三、InfluxDB通过HTTP API添加数据

InfluxDB通过HTTP API添加数据主要使用如下格式：

curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary 'cpu\_load\_short,host=server01,region=us-west value=0.64 1434055562000000000'

说明：db=mydb是指使用mydb这个数据库。

--data-binary后面是需插入数据。

cpu\_load\_short是表名（measurement），tag字段是host和region，值分别为：server01和us-west。

field key字段是value，值为0.64。

时间戳（timestamp）指定为1434055562000000000。

这样，就向mydb数据库的cpu\_load\_short表中插入了一条数据。

其中，db参数必须指定一个数据库中已经存在的数据库名，数据体的格式遵从InfluxDB规定格式，首先是表名，后面是tags，然后是field，最后是时间戳。tags、field和时间戳三者之间以空格相分隔。

四、InfluxDB通过HTTP API添加多条数据

InfluxDB通过HTTP API添加多条数据与添加单条数据相似，示例如下：

curl -i -XPOST 'http://localhost:8086/write?db=mydb' --data-binary 'cpu\_load\_short,host=server02 value=0.67

cpu\_load\_short,host=server02,region=us-west value=**0.55** **1422568543702900257**

cpu\_load\_short,direction=in,host=server01,region=us-west value=**2.0** **1422568543702900257**'

这条语句向数据库mydb的表cpu\_load\_short中插入了三条数据。

第一条指定tag为host，值为server02，第二条指定tag为host和region，值分别为server02和us-west，第三条指定tag为direction，host，region，值分别为：in，server01，us-west。

五、InfluxDB 的HTTP API响应

在使用HTTP API时，InfluxDB的响应主要有以下几个：

1）2xx：204代表no content，200代表InfluxDB可以接收请求但是没有完成请求。一般会在body体中带有出错信息。

2）4xx：InfluxDB不能解析请求。

3）5xx：系统出现错误。

好了，有关InfluxDB的HTTP API写入操作就先为大家介绍到这里，下一篇将会介绍下如何使用 InfluxDB的HTTP API执行查询操作。

更多InfluxDB详细教程请看：[InfluxDB系列学习教程目录](http://www.linuxdaxue.com/influxdb-study-series-manual.html)

InfluxDB技术交流群：[580487672（点击加入）](http://www.linuxdaxue.com/wp-content/themes/template/inc/go.php?url=http://shang.qq.com/wpa/qunwpa?idkey=5b0509cf57a31e00504e549ba2c0bce1ff4e21d4b5631e55e626d1ad84a55989)

# influxdata/influxdb: Scalable datastore for metrics, events, and real-time analytics

https://github.com/influxdata/influxdb

An Open-Source Time Series Database

InfluxDB is an open source **time series database** with **no external dependencies**. It's useful for recording metrics, events, and performing analytics.

Features

* Built-in [HTTP API](https://docs.influxdata.com/influxdb/latest/guides/writing_data/) so you don't have to write any server side code to get up and running.
* Data can be tagged, allowing very flexible querying.
* SQL-like query language.
* Simple to install and manage, and fast to get data in and out.
* It aims to answer queries in real-time. That means every data point is indexed as it comes in and is immediately available in queries that should return in < 100ms.

Installation

We recommend installing InfluxDB using one of the [pre-built packages](https://influxdata.com/downloads/#influxdb). Then start InfluxDB using:

* service influxdb start if you have installed InfluxDB using an official Debian or RPM package.
* systemctl start influxdb if you have installed InfluxDB using an official Debian or RPM package, and are running a distro with systemd. For example, Ubuntu 15 or later.
* $GOPATH/bin/influxd if you have built InfluxDB from source.

Getting Started

Create your first database

curl -XPOST 'http://localhost:8086/query' --data-urlencode "q=CREATE DATABASE mydb"

Insert some data

curl -XPOST 'http://localhost:8086/write?db=mydb' \

-d 'cpu,host=server01,region=uswest load=42 1434055562000000000'

curl -XPOST 'http://localhost:8086/write?db=mydb' \

-d 'cpu,host=server02,region=uswest load=78 1434055562000000000'

curl -XPOST 'http://localhost:8086/write?db=mydb' \

-d 'cpu,host=server03,region=useast load=15.4 1434055562000000000'

Query for the data

curl -G http://localhost:8086/query?pretty=true --data-urlencode "db=mydb" \

--data-urlencode "q=SELECT \* FROM cpu WHERE host='server01' AND time < now() - 1d"

Analyze the data

curl -G http://localhost:8086/query?pretty=true --data-urlencode "db=mydb" \

--data-urlencode "q=SELECT mean(load) FROM cpu WHERE region='uswest'"

Documentation

* Read more about the [design goals and motivations of the project](https://docs.influxdata.com/influxdb/latest/).
* Follow the [getting started guide](https://docs.influxdata.com/influxdb/latest/introduction/getting_started/) to learn the basics in just a few minutes.
* Learn more about [InfluxDB's key concepts](https://docs.influxdata.com/influxdb/latest/guides/writing_data/).

Contributing

If you're feeling adventurous and want to contribute to InfluxDB, see our [contributing doc](https://github.com/influxdata/influxdb/blob/master/CONTRIBUTING.md) for info on how to make feature requests, build from source, and run tests.

Looking for Support?

InfluxDB offers a number of services to help your project succeed. We offer Developer Support for organizations in active development, Managed Hosting to make it easy to move into production, and Enterprise Support for companies requiring the best response times, SLAs, and technical fixes. Visit our [support page](https://influxdata.com/services/) or contact [sales@influxdb.com](mailto:sales@influxdb.com) to learn how we can best help you succeed.

# InfluxData | Documentation | Storage Engine

https://docs.influxdata.com/influxdb/v1.2/concepts/storage\_engine/

The InfluxDB Storage Engine and the Time-Structured Merge Tree (TSM)

The new InfluxDB storage engine looks very similar to a LSM Tree. It has a write ahead log and a collection of read-only data files which are similar in concept to SSTables in an LSM Tree. TSM files contain sorted, compressed series data.

InfluxDB will create a [**shard**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#shard) for each block of time. For example, if you have a [**retention policy**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#retention-policy) with an unlimited duration, shards will be created for each 7 day block of time. Each of these shards maps to an underlying storage engine database. Each of these databases has its own [**WAL**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#wal-write-ahead-log) and TSM files.

We’ll dig into each of these parts of the storage engine.

[Storage Engine](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#storage-engine)

The storage engine ties a number components together and provides the external interface for storing and querying series data. It is composed of a number of components that each serve a particular role:

* In-Memory Index - The in-memory index is a shared index across shards that provides the quick access to[**measurements**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#measurement), [**tags**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#tags), and [**series**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#series). The index is used by the engine, but is not specific to the storage engine itself.
* WAL - The WAL is a write-optimized storage format that allows for writes to be durable, but not easily queryable. Writes to the WAL are appended to segments of a fixed size.
* Cache - The Cache is an in-memory representation of the data stored in the WAL. It is queried at runtime and merged with the data stored in TSM files.
* TSM Files - TSM files store compressed series data in a columnar format.
* FileStore - The FileStore mediates access to all TSM files on disk. It ensures that TSM files are installed atomically when existing ones are replaced as well as removing TSM files that are no longer used.
* Compactor - The Compactor is responsible for converting less optimized Cache and TSM data into more read-optimized formats. It does this by compressing series, removing deleted data, optimizing indices and combining smaller files into larger ones.
* Compaction Planner - The Compaction Planner determines which TSM files are ready for a compaction and ensures that multiple concurrent compactions do not interfere with each other.
* Compression - Compression is handled by various Encoders and Decoders for specific data types. Some encoders are fairly static and always encode the same type the same way; others switch their compression strategy based on the shape of the data.
* Writers/Readers - Each file type (WAL segment, TSM files, tombstones, etc..) has Writers and Readers for working with the formats.

[**Write Ahead Log (WAL)**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#write-ahead-log-wal)

The WAL is organized as a bunch of files that look like \_000001.wal. The file numbers are monotonically increasing and referred to as WAL segments. When a segment reaches 10MB in size, it is closed and a new one is opened. Each WAL segment stores multiple compressed blocks of writes and deletes.

When a write comes in the new points are serialized, compressed using Snappy, and written to a WAL file. The file is fsync’d and the data is added to an in-memory index before a success is returned. This means that batching points together is required to achieve high throughput performance. (Optimal batch size seems to be 5,000-10,000 points per batch for many use cases.)

Each entry in the WAL follows a [**TLV standard**](https://en.wikipedia.org/wiki/Type-length-value) with a single byte representing the type of entry (write or delete), a 4 byte uint32 for the length of the compressed block, and then the compressed block.

[**Cache**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#cache)

The Cache is an in-memory copy of all data points current stored in the WAL. The points are organized by the key, which is the measurement, [**tag set**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#tag-set), and unique [**field**](https://docs.influxdata.com/influxdb/v1.2/concepts/glossary/#field). Each field is kept as its own time-ordered range. The Cache data is not compressed while in memory.

Queries to the storage engine will merge data from the Cache with data from the TSM files. Queries execute on a copy of the data is made from the cache at query processing time. This way writes that come in while a query is running won’t affect the result.

Deletes sent to the Cache will clear out the given key or the specific time range for the given key.

The Cache exposes a few controls for snapshotting behavior. The two most important controls are the memory limits. There is a lower bound, [**cache-snapshot-memory-size**](https://docs.influxdata.com/influxdb/v1.2/administration/config/#cache-snapshot-memory-size-26214400), which when exceeded will trigger a snapshot to TSM files and remove the corresponding WAL segments. There is also an upper bound, [**cache-max-memory-size**](https://docs.influxdata.com/influxdb/v1.2/administration/config/#cache-max-memory-size-524288000), which when exceeded will cause the Cache to reject new writes. These configurations are useful to prevent out of memory situations and to apply back pressure to clients writing data faster than the instance can persist it.  
The checks for memory thresholds occur on every write.

The other snapshot controls are time based. The idle threshold, [**cache-snapshot-write-cold-duration**](https://docs.influxdata.com/influxdb/v1.2/administration/config/#cache-snapshot-write-cold-duration-1h0m0s), forces the Cache to snapshot to TSM files if it hasn’t received a write within the specified interval.

The in-memory Cache is recreated on restart by re-reading the WAL files on disk.

[**TSM Files**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#tsm-files)

TSM files are a collection of read-only files that are memory mapped. The structure of these files looks very similar to an SSTable in LevelDB or other LSM Tree variants.

A TSM file is composed of four sections: header, blocks, index, and footer.

┌────────┬────────────────────────────────────┬─────────────┬──────────────┐

│ Header │ Blocks │ Index │ Footer │

│5 bytes │ N bytes │ N bytes │ 4 bytes │

└────────┴────────────────────────────────────┴─────────────┴──────────────┘

The Header is a magic number to identify the file type and a version number.

┌───────────────────┐

│ Header │

├─────────┬─────────┤

│ Magic │ Version │

│ 4 bytes │ 1 byte │

└─────────┴─────────┘

Blocks are sequences of pairs of CRC32 checksums and data.  
The block data is opaque to the file.  
The CRC32 is used for block level error detection.  
The length of the blocks is stored in the index.

┌───────────────────────────────────────────────────────────┐

│ Blocks │

├───────────────────┬───────────────────┬───────────────────┤

│ Block 1 │ Block 2 │ Block N │

├─────────┬─────────┼─────────┬─────────┼─────────┬─────────┤

│ CRC │ Data │ CRC │ Data │ CRC │ Data │

│ 4 bytes │ N bytes │ 4 bytes │ N bytes │ 4 bytes │ N bytes │

└─────────┴─────────┴─────────┴─────────┴─────────┴─────────┘

Following the blocks is the index for the blocks in the file. The index is composed of a sequence of index entries ordered lexicographically by key and then by time. The key includes the measurement name, tag set, and one field. Multiple fields per point creates multiple indices in the TSM file. Each index entry starts with a key length and the key, followed by the block type (float, int, bool, string) and a count of the number of blocks in the file.  
Each index block entry is composed of the min and max time for the block, the offset into the file where the block is located and the the size of the block.

The index structure can provide efficient access to all blocks as well as the ability to determine the cost associated with accessing a given key. Given a key and timestamp, we can determine whether a file contains the block for that timestamp. We can also determine where that block resides and how much data must be read to retrieve the block. Knowing the size of the block, we can efficiently provision our IO statements.

┌────────────────────────────────────────────────────────────────────────────┐

│ Index │

├─────────┬─────────┬──────┬───────┬─────────┬─────────┬────────┬────────┬───┤

│ Key Len │ Key │ Type │ Count │Min Time │Max Time │ Offset │ Size │...│

│ 2 bytes │ N bytes │1 byte│2 bytes│ 8 bytes │ 8 bytes │8 bytes │4 bytes │ │

└─────────┴─────────┴──────┴───────┴─────────┴─────────┴────────┴────────┴───┘

The last section is the footer that stores the offset of the start of the index.

┌─────────┐

│ Footer │

├─────────┤

│Index Ofs│

│ 8 bytes │

└─────────┘

[**Compression**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#compression)

Each block is compressed to reduce storage space and disk IO when querying. A block contains the timestamps and values for a given series and field. Each block has one byte header, followed by the compressed timestamps and then the compressed values.

┌───────┬─────┬─────────────────┬──────────────────┐

│ Type │ Len │ Timestamps │ Values │

│1 Byte │VByte│ N Bytes │ N Bytes │

└───────┴─────┴─────────────────┴──────────────────┘

The timestamps and values are compressed and stored separately using encodings dependent on the data type and its shape. Storing them independently allows timestamp encoding to be used for all timestamps, while allowing different encodings for different field types. For example, some points may be able to use run-length encoding whereas other may not.

Each value type also contains a 1 byte header indicating the type of compression for the remaining bytes. The four high bits store the compression type and the four low bits are used by the encoder if needed.

**Timestamps**

Timestamp encoding is adaptive and based on the structure of the timestamps that are encoded. It uses a combination of delta encoding, scaling, and compression using simple8b run-length encoding, as well as falling back to no compression if needed.

Timestamp resolution is variable but can be as granular as a nanosecond, requiring up to 8 bytes to store uncompressed. During encoding, the values are first delta-encoded. The first value is the starting timestamp and subsequent values are the differences from the prior value. This usually converts the values into much smaller integers that are easier to compress. Many timestamps are also monotonically increasing and fall on even boundaries of time such as every 10s. When timestamps have this structure, they are scaled by the largest common divisor that is also a factor of 10. This has the effect of converting very large integer deltas into smaller ones that compress even better.

Using these adjusted values, if all the deltas are the same, the time range is stored using run-length encoding. If run-length encoding is not possible and all values are less than (1 << 60) - 1 ([**~36.5 years**](https://www.wolframalpha.com/input/?i=(1+%3C%3C+60)+-+1+nanoseconds+to+years) at nanosecond resolution), then the timestamps are encoded using [**simple8b encoding**](https://github.com/jwilder/encoding/tree/master/simple8b). Simple8b encoding is a 64bit word-aligned integer encoding that packs multiple integers into a single 64bit word. If any value exceeds the maximum the deltas are stored uncompressed using 8 bytes each for the block. Future encodings may use a patched scheme such as Patched Frame-Of-Reference (PFOR) to handle outliers more effectively.

**Floats**

Floats are encoded using an implementation of the [**Facebook Gorilla paper**](http://www.vldb.org/pvldb/vol8/p1816-teller.pdf). The encoding XORs consecutive values together to produce a small result when the values are close together. The delta is then stored using control bits to indicate how many leading and trailing zeroes are in the XOR value. Our implementation removes the timestamp encoding described in paper and only encodes the float values.

**Integers**

Integer encoding uses two different strategies depending on the range of values in the uncompressed data. Encoded values are first encoded using [**ZigZag encoding**](https://developers.google.com/protocol-buffers/docs/encoding?hl=en#signed-integers). This interleaves positive and negative integers across a range of positive integers.

For example, [-2,-1,0,1] becomes [3,1,0,2]. See Google’s [**Protocol Buffers documentation**](https://developers.google.com/protocol-buffers/docs/encoding?hl=en#signed-integers) for more information.

If all ZigZag encoded values are less than (1 << 60) - 1, they are compressed using simple8b encoding. If any values are larger than the maximum then all values are stored uncompressed in the block. If all values are identical, run-length encoding is used.  
This works very well for values that are frequently constant.

**Booleans**

Booleans are encoded using a simple bit packing strategy where each boolean uses 1 bit. The number of booleans encoded is stored using variable-byte encoding at the beginning of the block.

**Strings**

Strings are encoding using [**Snappy**](http://google.github.io/snappy/) compression. Each string is packed consecutively and they are compressed as one larger block.

[**Compactions**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#compactions)

Compactions are recurring processes that migrate data stored in a write-optimized format into a more read-optimized format. There are a number of stages of compaction that take place while a shard is hot for writes:

* Snapshots - Values in the Cache and WAL must be converted to TSM files to free memory and disk space used by the WAL segments. These compactions occur based on the cache memory and time thresholds.
* Level Compactions - Level compactions (levels 1-4) occur as the TSM files grow. TSM files are compacted from snapshots to level 1 files. Multiple level 1 files are compacted to produce level 2 files. The process continues until files reach level 4 and the max size for a TSM file.  
  They will not be compacted further unless deletes, index optimization compactions, or full compactions need to run. Lower level compactions use strategies that avoid CPU-intensive activities like decompressing and combining blocks. Higher level (and thus less frequent) compactions will re-combine blocks to fully compact them and increase the compression ratio.
* Index Optimization - When many level 4 TSM files accumulate, the internal indexes become larger and more costly to access. An index optimization compaction splits the series and indices across a new set of TSM files, sorting all points for a given series into one TSM file. Before an index optimization, each TSM file contained points for most or all series, and thus each contains the same series index. After an index optimzation, each TSM file contains points from a minimum of series and there is little series overlap between files. Each TSM file thus has a smaller unique series index, instead of a duplicate of the full series list. In addition, all points from a particular series are contiguous in a TSM file rather than spread across multiple TSM files.
* Full Compactions - Full compactions run when a shard has become cold for writes for long time, or when deletes have occurred on the shard. Full compactions produce an optimal set of TSM files and include all optimizations from Level and Index Optimization compactions. Once a shard is fully compacted, no other compactions will run on it unless new writes or deletes are stored.

[**Writes**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#writes)

Writes are appended to the current WAL segment and are also added to the Cache. Each WAL segment has a maximum size. Writes roll over to a new file once the current file fills up. The cache is also size bounded; snapshots are taken and WAL compactions are initiated when the cache becomes too full. If the inbound write rate exceeds the WAL compaction rate for a sustained period, the cache may become too full, in which case new writes will fail until the snapshot process catches up.

When WAL segments fill up and are closed, the Compactor snapshots the Cache and writes the data to a new TSM file. When the TSM file is successfully written and fsync’d, it is loaded and referenced by the FileStore.

[**Updates**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#updates)

Updates (writing a newer value for a point that already exists) occur as normal writes. Since cached values overwrite existing values, newer writes take precedence. If a write would overwrite a point in a prior TSM file, the points are merged at runtime and the newer write takes precedence.

[**Deletes**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#deletes)

Deletes occur by writing a delete entry to the WAL for the measurement or series and then updating the Cache and FileStore. The Cache evicts all relevant entries. The FileStore writes a tombstone file for each TSM file that contains relevant data. These tombstone files are used at startup time to ignore blocks as well as during compactions to remove deleted entries.

Queries against partially deleted series are handled at query time until a compaction removes the data fully from the TSM files.

[**Queries**](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#queries)

When a query is executed by the storage engine, it is essentially a seek to a given time associated with a specific series key and field. First, we do a search on the data files to find the files that contain a time range matching the query as well containing matching series.

Once we have the data files selected, we next need to find the position in the file of the series key index entries. We run a binary search against each TSM index to find the location of its index blocks.

In common cases the blocks will not overlap across multiple TSM files and we can search the index entries linearly to find the start block from which to read. If there are overlapping blocks of time, the index entries are sorted to ensure newer writes will take precedence and that blocks can be processed in order during query execution.

When iterating over the index entries the blocks are read sequentially from the blocks section. The block is decompressed and we seek to the specific point.

**The new InfluxDB storage engine: from LSM Tree to B+Tree and back again to create the Time Structured Merge Tree**

Writing a new storage format should be a last resort. So how did InfluxData end up writing our own engine? InfluxData has experimented with many storage formats and found each lacking in some fundamental way. The performance requirements for InfluxDB are significant, and eventually overwhelm other storage systems. The 0.8 line of InfluxDB allowed multiple storage engines, including LevelDB, RocksDB, HyperLevelDB, and LMDB. The 0.9 line of InfluxDB used BoltDB as the underlying storage engine. This writeup is about the Time Structured Merge Tree storage engine that was released in 0.9.5 and is the only storage engine supported in InfluxDB 0.11+, including the entire 1.x family.

The properties of the time series data use case make it challenging for many existing storage engines. Over the course of InfluxDB’s development we’ve tried a few of the more popular options. We started with LevelDB, an engine based on LSM Trees, which are optimized for write throughput. After that we tried BoltDB, an engine based on a memory mapped B+Tree, which is optimized for reads. Finally, we ended up building our own storage engine that is similar in many ways to LSM Trees.

With our new storage engine we were able to achieve up to a 45x reduction in disk space usage from our B+Tree setup with even greater write throughput and compression than what we saw with LevelDB and its variants. This post will cover the details of that evolution and end with an in-depth look at our new storage engine and its inner workings.

[Properties of Time Series Data](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#properties-of-time-series-data)

The workload of time series data is quite different from normal database workloads. There are a number of factors that conspire to make it very difficult to scale and remain performant:

* Billions of individual data points
* High write throughput
* High read throughput
* Large deletes (data expiration)
* Mostly an insert/append workload, very few updates

The first and most obvious problem is one of scale. In DevOps, IoT, or APM it is easy to collect hundreds of millions or billions of unique data points every day.

For example, let’s say we have 200 VMs or servers running, with each server collecting an average of 100 measurements every 10 seconds. Given there are 86,400 seconds in a day, a single measurement will generate 8,640 points in a day, per server. That gives us a total of 200 \* 100 \* 8,640 = 172,800,000 individual data points per day. We find similar or larger numbers in sensor data use cases.

The volume of data means that the write throughput can be very high. We regularly get requests for setups than can handle hundreds of thousands of writes per second. Some larger companies will only consider systems that can handle millions of writes per second.

At the same time, time series data can be a high read throughput use case. It’s true that if you’re tracking 700,000 unique metrics or time series you can’t hope to visualize all of them. That leads many people to think that you don’t actually read most of the data that goes into the database. However, other than dashboards that people have up on their screens, there are automated systems for monitoring or combining the large volume of time series data with other types of data.

Inside InfluxDB, aggregate functions calculated on the fly may combine tens of thousands of distinct time series into a single view. Each one of those queries must read each aggregated data point, so for InfluxDB the read throughput is often many times higher than the write throughput.

Given that time series is mostly an append-only workload, you might think that it’s possible to get great performance on a B+Tree. Appends in the keyspace are efficient and you can achieve greater than 100,000 per second. However, we have those appends happening in individual time series. So the inserts end up looking more like random inserts than append only inserts.

One of the biggest problems we found with time series data is that it’s very common to delete all data after it gets past a certain age. The common pattern here is that users have high precision data that is kept for a short period of time like a few days or months. Users then downsample and aggregate that data into lower precision rollups that are kept around much longer.

The naive implementation would be to simply delete each record once it passes its expiration time. However, that means that once the first points written reach their expiration date, the system is processing just as many deletes as writes, which is something most storage engines aren’t designed for.

Let’s dig into the details of the two types of storage engines we tried and how these properties had a significant impact on our performance.

[LevelDB and Log Structured Merge Trees](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#leveldb-and-log-structured-merge-trees)

When the InfluxDB project began, we picked LevelDB as the storage engine because we had used it for time series data storage in the product that was the precursor to InfluxDB. We knew that it had great properties for write throughput and everything seemed to “just work”.

LevelDB is an implementation of a Log Structured Merge Tree (or LSM Tree) that was built as an open source project at Google. It exposes an API for a key/value store where the key space is sorted. This last part is important for time series data as it allowed us to quickly scan ranges of time as long as the timestamp was in the key.

LSM Trees are based on a log that takes writes and two structures known as Mem Tables and SSTables. These tables represent the sorted keyspace. SSTables are read only files that are continuously replaced by other SSTables that merge inserts and updates into the keyspace.

The two biggest advantages that LevelDB had for us were high write throughput and built in compression. However, as we learned more about what people needed with time series data, we encountered a few insurmountable challenges.

The first problem we had was that LevelDB doesn’t support hot backups. If you want to do a safe backup of the database, you have to close it and then copy it. The LevelDB variants RocksDB and HyperLevelDB fix this problem, but there was another more pressing problem that we didn’t think they could solve.

Our users needed a way to automatically manage data retention. That meant we needed deletes on a very large scale. In LSM Trees, a delete is as expensive, if not more so, than a write. A delete writes a new record known as a tombstone. After that queries merge the result set with any tombstones to purge the deleted data from the query return. Later, a compaction runs that removes the tombstone record and the underlying deleted record in the SSTable file.

To get around doing deletes, we split data across what we call shards, which are contiguous blocks of time. Shards would typically hold either one day or seven days worth of data. Each shard mapped to an underlying LevelDB. This meant that we could drop an entire day of data by just closing out the database and removing the underlying files.

Users of RocksDB may at this point bring up a feature called ColumnFamilies. When putting time series data into Rocks, it’s common to split blocks of time into column families and then drop those when their time is up. It’s the same general idea: create a separate area where you can just drop files instead of updating indexes when you delete a large block of data. Dropping a column family is a very efficient operation. However, column families are a fairly new feature and we had another use case for shards.

Organizing data into shards meant that it could be moved within a cluster without having to examine billions of keys. At the time of this writing, it was not possible to move a column family in one RocksDB to another. Old shards are typically cold for writes so moving them around would be cheap and easy. We would have the added benefit of having a spot in the keyspace that is cold for writes so it would be easier to do consistency checks later.

The organization of data into shards worked great for a while, until a large amount of data went into InfluxDB. LevelDB splits the data out over many small files. Having dozens or hundreds of these databases open in a single process ended up creating a big problem. Users that had six months or a year of data would run out of file handles. It’s not something we found with the majority of users, but anyone pushing the database to its limits would hit this problem and we had no fix for it. There were simply too many file handles open.

[BoltDB and mmap B+Trees](https://docs.influxdata.com/influxdb/v1.2/concepts/storage_engine/#boltdb-and-mmap-b-trees)

After struggling with LevelDB and its variants for a year we decided to move over to BoltDB, a pure Golang database heavily inspired by LMDB, a mmap B+Tree database written in C. It has the same API semantics as LevelDB: a key value store where the keyspace is ordered. Many of our users were surprised. Our own posted tests of the LevelDB variants vs. LMDB (a mmap B+Tree) showed RocksDB as the best performer.

However, there were other considerations that went into this decision outside of the pure write performance. At this point our most important goal was to get to something stable that could be run in production and backed up. BoltDB also had the advantage of being written in pure Go, which simplified our build chain immensely and made it easy to build for other OSes and platforms.

The biggest win for us was that BoltDB used a single file as the database. At this point our most common source of bug reports were from people running out of file handles. Bolt solved the hot backup problem and the file limit problems all at the same time.

We were willing to take a hit on write throughput if it meant that we’d have a system that was more reliable and stable that we could build on. Our reasoning was that for anyone pushing really big write loads, they’d be running a cluster anyway.

We released versions 0.9.0 to 0.9.2 based on BoltDB. From a development perspective it was delightful. Clean API, fast and easy to build in our Go project, and reliable. However, after running for a while we found a big problem with write throughput. After the database got over a few GB, writes would start spiking IOPS.

Some users were able to get past this by putting InfluxDB on big hardware with near unlimited IOPS. However, most users are on VMs with limited resources in the cloud. We had to figure out a way to reduce the impact of writing a bunch of points into hundreds of thousands of series at a time.

With the 0.9.3 and 0.9.4 releases our plan was to put a write ahead log (WAL) in front of Bolt. That way we could reduce the number of random insertions into the keyspace. Instead, we’d buffer up multiple writes that were next to each other and then flush them at once. However, that only served to delay the problem. High IOPS still became an issue and it showed up very quickly for anyone operating at even moderate work loads.

However, our experience building the first WAL implementation in front of Bolt gave us the confidence we needed that the write problem could be solved. The performance of the WAL itself was fantastic, the index simply could not keep up. At this point we started thinking again about how we could create something similar to an LSM Tree that could keep up with our write load.

Thus was born the Time Structured Merge Tree.

# 2@influxdb基本操作

- u010185262的博客 - 博客频道 - CSDN.NET

http://blog.csdn.net/u010185262/article/details/53158786

名词解释

在具体的讲解influxdb的相关操作之前先说说influxdb的一些专有名词，这些名词代表什么。

influxDB名词

* **database**：数据库；
* **measurement**：数据库中的表；
* **points**：表里面的一行数据。

influxDB中独有的一些概念

Point由**时间戳**（time）、**数据**（field）和**标签**（tags）组成。

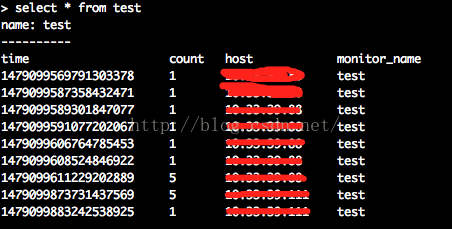
* **time**：每条数据记录的时间，也是数据库自动生成的主索引；
* **fields**：各种记录的值；
* **tags**：各种有索引的属性。

还有一个重要的名词：**series**

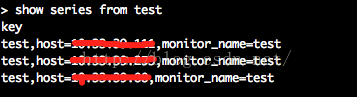
所有在数据库中的数据，都需要通过图表来表示，series表示这个表里面的所有的数据可以在图标上画成几条线（注：线条的个数由tags排列组合计算出来）

举个简单的小例子：

有以下数据：



它的series为：



influxDB基本操作

数据库与表的操作

可以直接在web管理页面做操作，当然也可以命令行。

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. #创建数据库
2. **create** **database** "db\_name"
4. #显示所有的数据库
5. show databases
7. #删除数据库
8. **drop** **database** "db\_name"
10. #使用数据库
11. use db\_name
13. #显示该数据库中所有的表
14. show measurements
16. #创建表，直接在插入数据的时候指定表名
17. **insert** test,host=127.0.0.1,monitor\_name=test count=1
19. #删除表
20. **drop** measurement "measurement\_name"

增

向数据库中插入数据。

* 通过命令行

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. > use metrics
2. Using **database** metrics
3. > **insert** test,host=127.0.0.1,monitor\_name=test count=1

这样，数据库插入数据成功。

* 通过http接口

**[plain]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. curl -i -XPOST 'http://127.0.0.1:8086/write?db=metrics' --data-binary 'test,host=127.0.0.1,monitor\_name=test count=1'

读者看到这里可能会观察到插入的数据的格式貌似比较奇怪，这是因为influxDB存储数据采用的是Line Protocol格式。那么何谓Line Protoco格式？

**Line Protocol格式：**写入数据库的Point的固定格式。

在上面的两种插入数据的方法中都有这样的一部分：

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. test,host=127.0.0.1,monitor\_name=test count=1

其中：

1. test：表名；
2. host=127.0.0.1,monitor\_name=test：tag；
3. count=1：field

相对此格式有详细的了解参见[官方文档](https://docs.influxdata.com/influxdb/v0.10/write_protocols/line/)

查

查询数据库中的数据。

* 通过命令行

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. > use metrics
2. Using **database** metrics
3. > **select** \* **from** test **order** **by** **time** **desc**

* 通过http接口

**[plain]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. curl -G 'http://localhost:8086/query?pretty=true' --data-urlencode "db=metrics" --data-urlencode "q=select \* from test order by time desc"

influxDB是支持类sql语句的，具体的查询语法都差不多，这里就不再做详细的赘述了。

数据保存策略（Retention Policies）

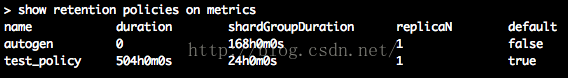
influxDB是没有提供直接删除数据记录的方法，但是提供数据保存策略，主要用于指定数据保留时间，超过指定时间，就删除这部分数据。

* 查看当前数据库Retention Policies

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. show retention policies **on** "db\_name"



* 创建新的Retention Policies

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. **create** retention policy "rp\_name" **on** "db\_name" duration 3w replication 1 **default**
   * rp\_name：策略名
   * db\_name：具体的数据库名
   * 3w：保存3周，3周之前的数据将被删除，influxdb具有各种事件参数，比如：h（小时），d（天），w（星期）
   * replication 1：副本个数，一般为1就可以了
   * default：设置为默认策略

* 修改Retention Policies

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. **alter** retention policy "rp\_name" **on** "db\_name" duration 30d **default**

* 删除Retention Policies

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. **drop** retention policy "rp\_name"

连续查询（Continous Queries）

当数据超过保存策略里指定的时间之后就会被删除，但是这时候可能并不想数据被完全删掉，怎么办？

influxdb提供了联系查询，可以做数据统计采样。

* 查看数据库的Continous Queries

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. show continuous queries



* 创建新的Continous Queries

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. **create** continous query cq\_name **on** db\_name **begin** **select** sum(count) **into** new\_table\_name **from** table\_name **group** **by** **time**(30m) **end**
   * cq\_name：连续查询名字
   * db\_name：数据库名字
   * sum(count)：计算总和
   * table\_name：当前表名
   * new\_table\_name：存新的数据的表名
   * 30m：时间间隔为30分钟

* 删除Continous Queries

**[plain]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. drop continous query cp\_name on db\_name

用户管理

可以直接在web管理页面做操作，也可以命令行。

**[sql]** [view plain](http://blog.csdn.net/u010185262/article/details/53158786) [copy](http://blog.csdn.net/u010185262/article/details/53158786)

[在CODE上查看代码片](https://code.csdn.net/snippets/1984071)

1. #显示用户
2. show users
4. #创建用户
5. **create** user "username" **with** **password** 'password'
7. #创建管理员权限用户
8. **create** user "username" **with** **password** 'password' **with** all **privileges**
10. #删除用户
11. **drop** user "username"

# 2@LSM-Tree （BigTable 的理论模型） - RaymondSQ - 博客园

http://www.cnblogs.com/raymondshiquan/archive/2011/06/04/2072630.html

Google的BigTable架构在分布式结构化存储方面大名鼎鼎，其中的MergeDump模型在读写之间找到了一个较好的平衡点，很好的解决了web scale数据的读写问题。

MergeDump的理论基础是LSM-Tree (Log-Structured Merge-Tree), 原文见：[LSM Tree](http://www.google.com.hk/url?sa=t&source=web&cd=1&ved=0CB4QFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.44.2782%26rep%3Drep1%26type%3Dpdf&ei=Uw-wTZbOJoemugO30u2TBw&usg=AFQjCNGGoN9IFTLShcv2HbL0RVQdElfxow&sig2=6gwuvzSHDyIra0eiwWomXA)

下面先说一下LSM-Tree的基本思想，再记录下读文章的几点感受。

LSM思想非常朴素，就是将对数据的更改hold在内存中，达到指定的threadhold后将该批更改批量写入到磁盘，在批量写入的过程中跟已经存在的数据做rolling merge。

拿update举个例子：

比如有1000万行数据，现在希望update table.a set addr='new addr' where pk = '833'，

如果使用B-Tree类似的结构操作，就需要：

1. 找到该条记录所在的page，

2. load page到内存（如果恰好该page已经在内存中，则省略该步）

3. 如果该page之前被修改过，则先flush page to disk

4. 修改数据

上面的动作平均来说有两次disk I/O，

如果采用LSM-Tree类似结构，则：

1. 将需要修改的数据直接写入内存

可见这里是没有disk I/O的。

当然，我们要说，这样的话读的时候就费劲了，需要merge disk上的数据和memory中的修改数据，这显然降低了读的性能。

确实如此，所以作者其中有个假设，就是写入远大于读取的时候，LSM是个很好的选择。我觉得更准确的描述应该是”优化了写，没有显著降低读“，因为大部分时候我们都是要求读最新的数据，而最新的数据很可能还在内存里面，即使不在内存里面，只要不是那些更新特别频繁的数据，其I/O次数也是有限的。

所以LSM-Tree比较适合的应用场景是：insert数据量大，读数据量和update数据量不高且读一般针对最新数据。

文章读下来有以下几点感受：

1. 基本思想早就有了，作者给出了较好的表现形式。

2. Merge是page/block级别的，而不是BigTable中的文件级别的。这一点主要原因可能是BigTable在分布式场景下做block级别很困那，而且GFS也不支持修改。

3. 其提出的比较标准比较有趣，将磁盘容量，转速等结合起来给出一个以美元为单位的cost标准，然后跟B-Tree结构的实现做了比较，结果当然是大大胜出。但是这里我觉得作者有些比较是不合理的，比如LSM使用log而B-Tree没有使用，这显然对B-Tree不公，其实B-Tree如果使用log，写入性能应该不比LSM差，顺序读取可能差一些。

4. 在Multi components 中，提出Ci/Ci+1的比例达到20的时候是最优的，这个数字意义不大，但是其中的分析方法对于Merge策略的选择是个启发。

分类: [分布式存储](http://www.cnblogs.com/raymondshiquan/category/295920.html)

标签: [分布式存储](http://www.cnblogs.com/raymondshiquan/tag/%E5%88%86%E5%B8%83%E5%BC%8F%E5%AD%98%E5%82%A8/)

# 2@Influxdb原理详解 - waitig - 博客园

http://www.cnblogs.com/waitig/p/influxdb-principle.html

[**Influxdb原理详解**](http://www.cnblogs.com/waitig/p/influxdb-principle.html)

本文属于《[InfluxDB系列教程](http://www.linuxdaxue.com/series/influxdb-series/)》文章系列，该系列共包括以下 15 部分：

1. [InfluxDB学习之InfluxDB的安装和简介](http://www.linuxdaxue.com/how-to-install-influxdb.html)
2. [InfluxDB学习之InfluxDB的基本概念](http://www.linuxdaxue.com/noun-interpretation-of-influxdb.html)
3. [InfluxDB学习之InfluxDB的基本操作](http://www.linuxdaxue.com/influxdb-basic-operation.html)
4. [InfluxDB学习之InfluxDB的HTTP API写入操作](http://www.linuxdaxue.com/influxdb-write-data-by-http-api.html)
5. [InfluxDB学习之InfluxDB数据保留策略（Retention Policies）](http://www.linuxdaxue.com/retention-policies-in-influxdb.html)
6. [InfluxDB学习之InfluxDB连续查询（Continuous Queries）](http://www.linuxdaxue.com/influxdb-continuous-queries.html)
7. [InfluxDB学习之InfluxDB的HTTP API查询操作](http://www.linuxdaxue.com/influxdb-query-data-by-http-api.html)
8. [InfluxDB学习之InfluxDB的关键概念](http://www.linuxdaxue.com/influxdb-study-key-concepts.html)
9. [InfluxDB学习之InfluxDB常用函数（一）聚合类函数](http://www.linuxdaxue.com/influxdb-study-influxdb-aggregations-funcitons.html)
10. [InfluxDB学习之InfluxDB常用函数（二）选择类函数](http://www.linuxdaxue.com/influxdb-study-influxdb-selectors-funcitons.html)
11. [InfluxDB学习之InfluxDB常用函数（三）变换类函数](http://www.linuxdaxue.com/influxdb-study-influxdb-transformations-funcitons.html)
12. [InfluxDB学习之再说连续查询](http://www.linuxdaxue.com/influxdb-continuous-queries-senior-knowlage.html)
13. Influxdb原理详解
14. [InfluxDB安装后web页面无法访问的解决方案](http://www.linuxdaxue.com/influxdb-can-not-use-admin-web.html)
15. [InfluxDB数据备份和恢复方法，支持本地和远程备份](http://www.linuxdaxue.com/influxdb-backup-and-restore.html)

系列详情请看：《[InfluxDB系列教程](http://www.linuxdaxue.com/series/influxdb-series/)》

**文章目录**

* [一、InfluxDB特点](http://www.linuxdaxue.com/influxdb-principle.html#title-0)
* [二、InfluxDB概念](http://www.linuxdaxue.com/influxdb-principle.html#title-1)
* [三、存储引擎 - TSM Tree](http://www.linuxdaxue.com/influxdb-principle.html#title-2)
* [四、目录与文件结构](http://www.linuxdaxue.com/influxdb-principle.html#title-3)
* [五、数据查询与索引结构](http://www.linuxdaxue.com/influxdb-principle.html#title-4)

InfluxDB是一款Go语言写的[时序数据库](http://www.linuxdaxue.com/tag/%E6%97%B6%E5%BA%8F%E6%95%B0%E6%8D%AE%E5%BA%93/)，本文主要介绍下Influxdb的架构和基本原理。

更多InfluxDB详细教程请看：[InfluxDB系列学习教程目录](http://www.linuxdaxue.com/influxdb-study-series-manual.html)

InfluxDB技术交流群：[580487672（点击加入）](http://www.linuxdaxue.com/wp-content/themes/template/inc/go.php?url=http://shang.qq.com/wpa/qunwpa?idkey=5b0509cf57a31e00504e549ba2c0bce1ff4e21d4b5631e55e626d1ad84a55989)

一、InfluxDB特点

* 可以设置metric的保存时间。
* 支持通过条件过滤以及正则表达式[删除](http://www.linuxdaxue.com/tag/%E5%88%A0%E9%99%A4/)数据。
* 支持类似 sql 的语法。
* 可以设置数据在集群中的副本数。
* 支持定期采样数据，写入另外的measurement，方便分粒度存储数据。

二、InfluxDB概念

1）数据格式 Line Protocol

在 InfluxDB 中，我们可以粗略的将要存入的一条数据看作一个虚拟的 key 和其对应的 value(field value)，格式如下：

cpu\_usage,host=server01,region=us-west value=0.64 1434055562000000000

虚拟的 key 包括以下几个部分： database, retention policy, measurement, tag sets, field name, timestamp。 database 和 retention policy 在上面的数据中并没有体现，通常在插入数据时在 http 请求的相应字段中指定。

* **database**: 数据库名，在 InfluxDB 中可以创建多个数据库，不同数据库中的数据文件是隔离存放的，存放在磁盘上的不同目录。
* **retention policy**: 存储策略，用于设置数据保留的时间，每个数据库刚开始会自动创建一个默认的存储策略 autogen，数据保留时间为永久，之后用户可以自己设置，例如保留最近2小时的数据。插入和查询数据时如果不指定存储策略，则使用默认存储策略，且默认存储策略可以修改。InfluxDB 会定期清除过期的数据。
* **measurement**: 测量指标名，例如 cpu\_usage 表示 cpu 的使用率。
* **tag sets**: tags 在 InfluxDB 中会按照字典序排序，不管是 tagk 还是 tagv，只要不一致就分别属于两个 key，例如 host=server01,region=us-west 和 host=server02,region=us-west 就是两个不同的 tag set。
* **field name**: 例如上面数据中的 value 就是 fieldName，InfluxDB 中支持一条数据中插入多个 fieldName，这其实是一个语法上的优化，在实际的底层存储中，是当作多条数据来存储。
* **timestamp**: 每一条数据都需要指定一个时间戳，在 TSM 存储引擎中会特殊对待，以为了优化后续的查询操作。

2）Point

InfluxDB 中单条插入语句的数据结构，series + timestamp 可以用于区别一个 point，也就是说一个 point 可以有多个 field name 和 field value。

3）Series

series 相当于是 InfluxDB 中一些数据的集合，在同一个 database 中，retention policy、measurement、tag sets 完全相同的数据同属于一个 series，同一个 series 的数据在物理上会按照时间顺序排列存储在一起。

series 的 key 为 measurement + 所有 tags 的序列化字符串，这个 key 在之后会经常用到。

代码中的结构如下：

type Series struct {

mu sync.RWMutex

Key string // series key

Tags map[string]string // tags

id uint64 // id

measurement \*Measurement // measurement

}

4）Shard

shard 在 InfluxDB 中是一个[比较](http://www.linuxdaxue.com/tag/%E6%AF%94%E8%BE%83/)重要的概念，它和 retention policy 相关联。每一个存储策略下会存在许多 shard，每一个 shard 存储一个指定时间段内的数据，并且不重复，例如 7点-8点 的数据落入 shard0 中，8点-9点的数据则落入 shard1 中。每一个 shard 都对应一个底层的 tsm 存储引擎，有独立的 cache、wal、tsm file。

[创建数据库](http://www.linuxdaxue.com/tag/%E5%88%9B%E5%BB%BA%E6%95%B0%E6%8D%AE%E5%BA%93/)时会自动创建一个默认存储策略，永久保存数据，对应的在此存储策略下的 shard 所保存的数据的时间段为 7 天，计算的函数如下：

func shardGroupDuration(d time.Duration) time.Duration {

if d >= 180\*24\*time.Hour || d == 0 { // 6 months or 0

return 7 \* 24 \* time.Hour

} else if d >= 2\*24\*time.Hour { // 2 days

return 1 \* 24 \* time.Hour

}

return 1 \* time.Hour

}

如果创建一个新的 retention policy 设置数据的保留时间为 1 天，则单个 shard 所存储数据的时间间隔为 1 小时，超过1个小时的数据会被存放到下一个 shard 中。

三、存储引擎 - TSM Tree

从 LevelDB（LSM Tree），到 BoltDB（mmap B+树），现在InfluxDB使用的是自己实现的 TSM Tree 的算法，类似 LSM Tree，针对 InfluxDB 的使用做了特殊优化。

TSM Tree 是 InfluxDB 根据实际需求在 LSM Tree 的基础上稍作修改优化而来。

TSM 存储引擎主要由几个部分组成： **cache、wal、tsm file、compactor**。

1）Shard

shard 并不能算是其中的一个组件，因为这是在 tsm 存储引擎之上的一个概念。在 InfluxDB 中按照数据的时间戳所在的范围，会去创建不同的 shard，每一个 shard 都有自己的 cache、wal、tsm file 以及 compactor，这样做的目的就是为了可以通过时间来快速定位到要查询数据的相关资源，加速查询的过程，并且也让之后的批量[删除](http://www.linuxdaxue.com/tag/%E5%88%A0%E9%99%A4/)数据的操作变得非常简单且高效。

在 LSM Tree 中删除数据是通过给指定 key 插入一个删除标记的方式，数据并不立即删除，需要等之后对文件进行压缩合并时才会真正地将数据删除，所以删除大量数据在 LSM Tree 中是一个非常低效的操作。

而在 InfluxDB 中，通过 retention policy 设置数据的保留时间，当检测到一个 shard 中的数据过期后，只需要将这个 shard 的资源释放，相关文件删除即可，这样的做法使得删除过期数据变得非常高效。

2）Cache

cache 相当于是 LSM Tree 中的 memtable，在内存中是一个简单的 map 结构，这里的 key 为 seriesKey + 分隔符 + filedName，目前代码中的分隔符为 #!~#，entry 相当于是一个按照时间排序的存放实际值的数组，具体结构如下：

type Cache struct {

commit sync.Mutex

mu sync.RWMutex

store map[string]\*entry

size uint64 // 当前使用内存的大小

maxSize uint64 // 缓存最大值

// snapshots are the cache objects that are currently being written to tsm files

// they're kept in memory while flushing so they can be queried along with the cache.

// they are read only and should never be modified

// memtable 快照，用于写入 tsm 文件，只读

snapshot \*Cache

snapshotSize uint64

snapshotting bool

// This number is the number of pending or failed WriteSnaphot attempts since the last successful one.

snapshotAttempts int

stats \*CacheStatistics

lastSnapshot time.Time

}

插入数据时，实际上是同时往 cache 与 wal 中写入数据，可以认为 cache 是 wal 文件中的数据在内存中的缓存。当 InfluxDB [启动](http://www.linuxdaxue.com/tag/%E5%90%AF%E5%8A%A8/)时，会遍历所有的 wal 文件，重新构造 cache，这样即使系统出现故障，也不会导致数据的丢失。

cache 中的数据并不是无限增长的，有一个 maxSize 参数用于控制当 cache 中的数据占用多少内存后就会将数据写入 tsm 文件。如果不配置的话，默认上限为 25MB，每当 cache 中的数据达到阀值后，会将当前的 cache 进行一次快照，之后清空当前 cache 中的内容，再创建一个新的 wal 文件用于写入，剩下的 wal 文件最后会被删除，快照中的数据会经过排序写入一个新的 tsm 文件中。

目前的 cache 的设计有一个问题，当一个快照正在被写入一个新的 tsm 文件时，当前的 cache 由于大量数据写入，又达到了阀值，此时前一次快照还没有完全写入磁盘，InfluxDB 的做法是让后续的写入操作失败，用户需要自己处理，等待恢复后继续写入数据。

3）WAL

wal 文件的内容与内存中的 cache 相同，其作用就是为了持久化数据，当系统崩溃后可以通过 wal 文件恢复还没有写入到 tsm 文件中的数据。

由于数据是被顺序插入到 wal 文件中，所以写入效率非常高。但是如果写入的数据没有按照时间顺序排列，而是以杂乱无章的方式写入，数据将会根据时间路由到不同的 shard 中，每一个 shard 都有自己的 wal 文件，这样就不再是完全的顺序写入，对性能会有一定影响。看到官方社区有说后续会进行优化，只使用一个 wal 文件，而不是为每一个 shard 创建 wal 文件。

wal 单个文件达到一定大小后会进行分片，创建一个新的 wal 分片文件用于写入数据。

4）TSM file

单个 tsm file 大小最大为 2GB，用于存放数据。

TSM file 使用了自己设计的格式，对查询性能以及压缩方面进行了很多优化，在后面的章节会具体说明其文件结构。

5）Compactor

compactor 组件在后台持续运行，每隔 1 秒会[检查](http://www.linuxdaxue.com/tag/%E6%A3%80%E6%9F%A5/)一次是否有需要压缩合并的数据。

主要进行两种操作，一种是 cache 中的数据大小达到阀值后，进行快照，之后转存到一个新的 tsm 文件中。

另外一种就是合并当前的 tsm 文件，将多个小的 tsm 文件合并成一个，使每一个文件尽量达到单个文件的最大大小，减少文件的数量，并且一些数据的删除操作也是在这个时候完成。

四、目录与文件结构

InfluxDB 的数据存储主要有三个目录。

默认情况下是 meta, wal 以及 data 三个目录。

meta 用于存储数据库的一些元数据，meta 目录下有一个 meta.db 文件。

wal 目录存放预写日志文件，以 .wal 结尾。data 目录存放实际存储的数据文件，以 .tsm 结尾。这两个目录下的结构是相似的，其基本结构如下：

# wal 目录结构

-- wal

-- mydb

-- autogen

-- 1

-- \_00001.wal

-- 2

-- \_00035.wal

-- 2hours

-- 1

-- \_00001.wal

# data 目录结构

-- data

-- mydb

-- autogen

-- 1

-- 000000001-000000003.tsm

-- 2

-- 000000001-000000001.tsm

-- 2hours

-- 1

-- 000000002-000000002.tsm

其中 mydb 是数据库名称，autogen 和 2hours 是存储策略名称，再下一层目录中的以数字命名的目录是 shard 的 ID 值，比如 autogen 存储策略下有两个 shard，ID 分别为 1 和 2，shard 存储了某一个时间段范围内的数据。再下一级的目录则为具体的文件，分别是 .wal 和 .tsm 结尾的文件。

1）WAL 文件

wal 文件中的一条数据，对应的是一个 key(measument + tags + fieldName) 下的所有 value 数据，按照时间排序。

* **Type (1 byte)**: 表示这个条目中 value 的类型。
* **Key Len (2 bytes)**: 指定下面一个字段 key 的长度。
* **Key (N bytes)**: 这里的 key 为 measument + tags + fieldName。
* **Count (4 bytes)**: 后面紧跟着的是同一个 key 下数据的个数。
* **Time (8 bytes)**: 单个 value 的时间戳。
* **Value (N bytes)**: value 的具体内容，其中 float64, int64, boolean 都是固定的字节数存储[比较](http://www.linuxdaxue.com/tag/%E6%AF%94%E8%BE%83/)简单，通过 Type 字段知道这里 value 的字节数。string 类型比较特殊，对于 string 来说，N bytes 的 Value 部分，前面 4 字节用于存储 string 的长度，剩下的部分才是 string 的实际内容。

2）TSM 文件

单个 tsm 文件的主要格式如下：

主要分为四个部分： **Header, Blocks, Index, Footer**。

其中 **Index** 部分的内容会被缓存在内存中，下面详细说明一下每一个部分的数据结构。

Header

* **MagicNumber (4 bytes)**: 用于区分是哪一个存储引擎，目前使用的 tsm1 引擎，MagicNumber 为 0x16D116D1。
* **Version (1 byte)**: 目前是 tsm1 引擎，此值固定为 1。

Blocks

Blocks 内部是一些连续的 Block，block 是 InfluxDB 中的最小读取对象，每次读取操作都会读取一个 block。每一个 Block 分为 CRC32 值和 Data 两部分，CRC32 值用于校验 Data 的内容是否有问题。Data 的长度记录在之后的 Index 部分中。

Data 中的内容根据[数据类型](http://www.linuxdaxue.com/tag/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B/)的不同，在 InfluxDB 中会采用不同的压缩方式，float 值采用了 Gorilla float compression，而 timestamp 因为是一个递增的序列，所以实际上压缩时只需要记录时间的偏移量信息。string 类型的 value 采用了 snappy 算法进行压缩。

Data 的数据解压后的格式为 8 字节的时间戳以及紧跟着的 value，value 根据类型的不同，会占用不同大小的空间，其中 string 为不定长，会在数据开始处存放长度，这一点和 WAL 文件中的格式相同。

Index

Index 存放的是前面 Blocks 里内容的索引。索引条目的顺序是先按照 key 的字典序排序，再按照 time 排序。InfluxDB 在做查询操作时，可以根据 Index 的信息快速定位到 tsm file 中要查询的 block 的位置。

这张图只展示了其中一部分，用结构体来表示的话类似下面的代码：

type BlockIndex struct {

MinTime int64

MaxTime int64

Offset int64

Size uint32

}

type KeyIndex struct {

KeyLen uint16

Key string

Type byte

Count uint32

Blocks []\*BlockIndex

}

type Index []\*KeyIndex

Key Len (2 bytes): 下面一个字段 key 的长度。

Key (N bytes): 这里的 key 指的是 seriesKey + 分隔符 + fieldName。

Type (1 bytes): fieldName 所对应的 fieldValue 的类型，也就是 Block 中 Data 内的数据的类型。

Count (2 bytes): 后面紧跟着的 Blocks 索引的个数。

后面四个部分是 block 的索引信息，根据 Count 中的个数会重复出现，每个 block 索引固定为 28 字节，按照时间排序。

Min Time (8 bytes): block 中 value 的最小时间戳。

Max Time (8 bytes): block 中 value 的最大时间戳。

Offset (8 bytes): block 在整个 tsm file 中的偏移量。

Size (4 bytes): block 的大小。根据 Offset + Size 字段就可以快速读取出一个 block 中的内容。

间接索引

间接索引只存在于内存中，是为了可以快速定位到一个 key 在详细索引信息中的位置而创建的，可以被用于二分查找来实现快速检索。

offsets 是一个数组，其中存储的值为每一个 key 在 Index 表中的位置，由于 key 的长度固定为 2字节，所以根据这个位置就可以找到该位置上对应的 key 的内容。

当指定一个要查询的 key 时，就可以通过二分查找，定位到其在 Index 表中的位置，再根据要查询的数据的时间进行定位，由于 KeyIndex  中的 BlockIndex 结构是定长的，所以也可以进行一次二分查找，找到要查询的数据所在的 BlockIndex 的内容，之后根据偏移量以及 block 长度就可以从 tsm 文件中快速读取出一个 block 的内容。

Footer

tsm file 的最后8字节的内容存放了 Index 部分的起始位置在 tsm file 中的偏移量，方便将索引信息加载到内存中。

五、数据查询与索引结构

由于 LSM Tree 的原理就是通过将大量的随机写转换为顺序写，从而极大地提升了数据写入的性能，与此同时牺牲了部分读的性能。TSM 存储引擎是基于 LSM Tree 开发的，所以情况类似。通常设计数据库时会采用索引文件的方式（例如 LevelDB 中的 Mainfest 文件） 或者 Bloom filter 来对 LSM Tree 这样的数据结构的读取操作进行优化。

**InfluxDB 中采用索引的方式进行优化，主要存在两种类型的索引。**

1）元数据索引

一个数据库的元数据索引通过 DatabaseIndex 这个结构体来存储，在数据库启动时，会进行初始化，从所有 shard 下的 tsm file 中加载 index 数据，获取其中所有 Measurement 以及 Series 的信息并缓存到内存中。

type DatabaseIndex struct {

measurements map[string]\*Measurement // 该数据库下所有 Measurement 对象

series map[string]\*Series // 所有 Series 对象，SeriesKey = measurement + tags

name string // 数据库名

}

这个结构体中最主要存放的就是该数据下所有 Measurement 和 Series 的内容，其数据结构如下：

[复制代码](javascript:void(0);)

type Measurement struct {

Name string `json:"name,omitempty"`

fieldNames map[string]struct{} // 此 measurement 中的所有 filedNames

// 内存中的索引信息

// id 以及其对应的 series 信息，主要是为了在 seriesByTagKeyValue 中存储Id节约内存

seriesByID map[uint64]\*Series // lookup table for series by their id

// 根据 tagk 和 tagv 的双重索引，保存排好序的 SeriesID 数组

// 这个 map 用于在查询操作时，可以根据 tags 来快速过滤出要查询的所有 SeriesID，之后根据 SeriesKey 以及时间范围从文件中读取相应内容

seriesByTagKeyValue map[string]map[string]SeriesIDs // map from tag key to value to sorted set of series ids

// 此 measurement 中所有 series 的 id，按照 id 排序

seriesIDs SeriesIDs // sorted list of series IDs in this measurement

}

type Series struct {

Key string // series key

Tags map[string]string // tags

id uint64 // id

measurement \*Measurement // 所属 measurement

// 在哪些 shard 中存在

shardIDs map[uint64]bool // shards that have this series defined

}

[复制代码](javascript:void(0);)

元数据查询

InfluxDB 支持一些特殊的查询语句（支持正则表达式匹配），可以查询一些 measurement 以及 tags 相关的数据，例如

SHOW MEASUREMENTS

SHOW TAG KEYS FROM "measurement\_name"

SHOW TAG VALUES FROM "measurement\_name" WITH KEY = "tag\_key"

例如我们需要查询 cpu\_usage 这个 measurement 上传数据的机器有哪些，一个可能的查询语句为：

SHOW TAG VALUES FROM "cpu\_usage" WITH KEY = "host"

首先根据 measurement 可以在 DatabaseIndex.measurements 中拿到 cpu\_usage 所对应的 Measurement 对象。

通过 Measurement.seriesByTagKeyValue 获取 tagk=host 所对应的以 tagv 为键的 map 对象。

遍历这个 map 对象，所有的 key 则为我们需要获取的数据。

普通数据查询的定位

对于普通的数据查询语句，则可以通过上述的元数据索引快速定位到要查询的数据所包含的所有 seriesKey，fieldName 和时间范围。

举个例子，假设查询语句为获取 server01 这台机器上 cpu\_usage 指标最近一小时的数据：

`SELECT value FROM "cpu\_usage" WHERE host='server01' AND time > now() - 1h`

先根据 measurement=cpu\_usage 从 DatabaseIndex.measurements 中获取到 cpu\_usage 对应的 Measurement 对象。

之后通过 DatabaseIndex.measurements["cpu\_usage"].seriesByTagKeyValue["host"]["server01"] 获取到所有匹配的 series 的 ID值，再通过 Measurement.seriesByID 这个 map 对象根据 series ID 获取它们的实际对象。

注意这里虽然只指定了 host=server01，但不代表 cpu\_usage 下只有这一个 series，可能还有其他的 tags 例如 user=1 以及 user=2，这样获取到的 series ID 实际上有两个，获取数据时需要获取所有 series 下的数据。

在 Series 结构体中的 shardIDs 这个 map 变量存放了哪些 shard 中存在这个 series 的数据。而 Measurement.fieldNames 这个 map 可以帮助过滤掉 fieldName 不存在的情况。

至此，我们在 o(1) 的时间复杂度内，获取到了所有符合要求的 series key、这些 series key 所存在的 shardID，要查询数据的时间范围，之后我们就可以创建数据迭代器从不同的 shard 中获取每一个 series key 在指定时间范围内的数据。后续的查询则和 tsm file 中的 Index 的在内存中的缓存相关。

2）TSM File 索引

上文中对于 tsm file 中的 Index 部分会在内存中做间接索引，从而可以实现快速检索的目的。这里看一下具体的数据结构：

type indirectIndex struct {

b []byte // 下层详细索引的字节流

offsets []int32 // 偏移量数组，记录了一个 key 在 b 中的偏移量

minKey, maxKey string

minTime, maxTime int64 // 此文件中的最小时间和最大时间，根据这个可以快速判断要查询的数据在此文件中是否存在，是否有必要读取这个文件

tombstones map[string][]TimeRange // 用于记录哪些 key 在指定范围内的数据是已经被删除的

}

b 直接对应着 tsm file 中的 Index 部分，通过对 offsets 进行二分查找，可以获取到指定 key 的所有 block 的索引信息，之后根据 offset 和 size 信息可以取出一个指定的 block 中的所有数据。

type indexEntries struct {

Type byte

entries []IndexEntry

}

type IndexEntry struct {

// 一个 block 中的 point 都在这个最小和最大的时间范围内

MinTime, MaxTime int64

// block 在 tsm 文件中偏移量

Offset int64

// block 的具体大小

Size uint32

}

在上一节中说明了通过元数据索引可以获取到所有 符合要求的 series key，它们对应的 shardID，时间范围。通过 tsm file 索引，我们就可以根据 series key 和 时间范围快速定位到数据在 tsm file 中的位置。

从 tsm file 中读取数据

InfluxDB 中的所有数据读取操作都通过 Iterator 来完成。

Iterator 是一个抽象概念，并且支持嵌套，一个 Iterator 可以从底层的其他 Iterator 中获取数据并进行处理，之后再将结果传递给上层的 Iterator。

这部分的代码逻辑比较复杂，这里不展开说明。实际上 Iterator 底层最主要的就是通过 cursor 来获取数据。

type cursor interface {

next() (t int64, v interface{})

}

type floatCursor interface {

cursor

nextFloat() (t int64, v float64)

}

// 底层主要是 KeyCursor，每次读取一个 block 的数据

type floatAscendingCursor struct {

// 内存中的 value 对象

cache struct {

values Values

pos int

}

tsm struct {

tdec TimeDecoder // 时间序列化对象

vdec FloatDecoder // value 序列化对象

buf []FloatValue

values []FloatValue // 从 tsm 文件中读取到的 FloatValue 的缓存

pos int

keyCursor \*KeyCursor

}

}

cursor 提供了一个 next() 方法用于获取一个 value 值。每一种[数据类型](http://www.linuxdaxue.com/tag/%E6%95%B0%E6%8D%AE%E7%B1%BB%E5%9E%8B/)都有一个自己的 cursor 实现。

底层实现都是 KeyCursor，KeyCursor 会缓存每个 Block 的数据，通过 Next() 函数依次返回，当一个 Block 中的内容读完后再通过 ReadBlock() 函数读取下一个 Block 中的内容。

更多InfluxDB详细教程请看：[InfluxDB系列学习教程目录](http://www.linuxdaxue.com/influxdb-study-series-manual.html)

InfluxDB技术交流群：[580487672（点击加入）](http://www.linuxdaxue.com/wp-content/themes/template/inc/go.php?url=http://shang.qq.com/wpa/qunwpa?idkey=5b0509cf57a31e00504e549ba2c0bce1ff4e21d4b5631e55e626d1ad84a55989)

* 参考：http://blog.fatedier.com/2016/08/05/detailed-in-influxdb-tsm-storage-engine-one/

http://blog.fatedier.com/2016/08/15/detailed-in-influxdb-tsm-storage-engine-two/

http://blog.fatedier.com/2016/07/05/research-of-time-series-database-influxdb/

学习Linux技术，就来[[Linux大学网](http://www.linuxdaxue.com/)]! 关注Linux大学网微信公众号，每日推送最专业的Linux技术知识！

标签: [InfluxDB](http://www.cnblogs.com/waitig/tag/InfluxDB/), [时序数据库](http://www.cnblogs.com/waitig/tag/%E6%97%B6%E5%BA%8F%E6%95%B0%E6%8D%AE%E5%BA%93/), [HTTP](http://www.cnblogs.com/waitig/tag/HTTP/), [API](http://www.cnblogs.com/waitig/tag/API/), [原理详解](http://www.cnblogs.com/waitig/tag/%E5%8E%9F%E7%90%86%E8%AF%A6%E8%A7%A3/)

# InfluxDB学习之InfluxDB连续查询（Continuous Queries） | Linux大学

http://www.linuxdaxue.com/influxdb-continuous-queries.html

本文属于《[InfluxDB系列教程](http://www.linuxdaxue.com/series/influxdb-series/)》文章系列，该系列共包括以下 16 部分：

1. [InfluxDB学习之InfluxDB的安装和简介](http://www.linuxdaxue.com/how-to-install-influxdb.html)
2. [InfluxDB学习之InfluxDB的基本概念](http://www.linuxdaxue.com/noun-interpretation-of-influxdb.html)
3. [InfluxDB学习之InfluxDB的基本操作](http://www.linuxdaxue.com/influxdb-basic-operation.html)
4. [InfluxDB学习之InfluxDB的HTTP API写入操作](http://www.linuxdaxue.com/influxdb-write-data-by-http-api.html)
5. [InfluxDB学习之InfluxDB数据保留策略（Retention Policies）](http://www.linuxdaxue.com/retention-policies-in-influxdb.html)
6. InfluxDB学习之InfluxDB连续查询（Continuous Queries）
7. [InfluxDB学习之InfluxDB的HTTP API查询操作](http://www.linuxdaxue.com/influxdb-query-data-by-http-api.html)
8. [InfluxDB学习之InfluxDB的关键概念](http://www.linuxdaxue.com/influxdb-study-key-concepts.html)
9. [InfluxDB学习之InfluxDB常用函数（一）聚合类函数](http://www.linuxdaxue.com/influxdb-study-influxdb-aggregations-funcitons.html)
10. [InfluxDB学习之InfluxDB常用函数（二）选择类函数](http://www.linuxdaxue.com/influxdb-study-influxdb-selectors-funcitons.html)
11. [InfluxDB学习之InfluxDB常用函数（三）变换类函数](http://www.linuxdaxue.com/influxdb-study-influxdb-transformations-funcitons.html)
12. [InfluxDB学习之再说连续查询](http://www.linuxdaxue.com/influxdb-continuous-queries-senior-knowlage.html)
13. [Influxdb原理详解](http://www.linuxdaxue.com/influxdb-principle.html)
14. [InfluxDB安装后web页面无法访问的解决方案](http://www.linuxdaxue.com/influxdb-can-not-use-admin-web.html)
15. [InfluxDB数据备份和恢复方法，支持本地和远程备份](http://www.linuxdaxue.com/influxdb-backup-and-restore.html)
16. [InfluxDB执行语句管理(query management)](http://www.linuxdaxue.com/influxdb-query-management.html)

系列详情请看：《[InfluxDB系列教程](http://www.linuxdaxue.com/series/influxdb-series/)》

在上一篇：[InfluxDB学习之InfluxDB数据保留策略（Retention Policies）](http://www.linuxdaxue.com/retention-policies-in-influxdb.html) 中，我们介绍了 InfluxDB的数据保留策略，数据超过保存策略里指定的时间之后，就会被删除。

但是如果我们不想完全将这些数据删除掉，就需要连续查询（Continuous Queries）的帮助了。

连续查询主要用在将数据归档，以降低系统空间的占用率，主要是以降低精度为代价。

更多InfluxDB详细教程请看：[InfluxDB系列学习教程目录](http://www.linuxdaxue.com/influxdb-study-series-manual.html)

InfluxDB技术交流群：[580487672（点击加入）](http://www.linuxdaxue.com/wp-content/themes/template/inc/go.php?url=http://shang.qq.com/wpa/qunwpa?idkey=5b0509cf57a31e00504e549ba2c0bce1ff4e21d4b5631e55e626d1ad84a55989)

一、InfluxDB连续查询 定义

InfluxDB的连续查询是在数据库中自动定时启动的一组语句，语句中必须包含 SELECT 关键词和 GROUP BY time() 关键词。

InfluxDB会将查询结果放在指定的数据表中。

二、InfluxDB连续查询 目的

使用连续查询是最优的降低采样率的方式，连续查询和存储策略搭配使用将会大大降低InfluxDB的系统占用量。

而且使用连续查询后，数据会存放到指定的数据表中，这样就为以后统计不同精度的数据提供了方便。

三、InfluxDB连续查询 操作

只有[管理](http://www.linuxdaxue.com/tag/%e7%ae%a1%e7%90%86/)员用户可以操作 连续查询。

1）新建连续查询

新建连续查询的语法如下所示：

CREATE CONTINUOUS QUERY <cq\_name> ON <database\_name>

[RESAMPLE [EVERY <interval>] [FOR <interval>]]

BEGIN SELECT <function>(<stuff>)[,<function>(<stuff>)] INTO <[diff](http://www.linuxdaxue.com/tag/diff/)erent\_measurement>

FROM <current\_measurement> [WHERE <stuff>] GROUP BY time(<interval>)[,<stuff>]

END

查询部分被 CREATE CONTINUOUS QUERY [...] BEGIN 和 END 所包含，主要的逻辑代码也是在这一部分。

使用示例：

> CREATE CONTINUOUS QUERY cq\_30m ON telegraf BEGIN SELECT mean(used) INTO mem\_used\_30m FROM mem GROUP BY time(30m) END

> SHOW CONTINUOUS QUERIES

name: telegraf

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

name query

cq\_30m CREATE CONTINUOUS QUERY cq\_30m ON telegraf BEGIN

SELECT mean(used) INTO telegraf."default".mem\_used\_30m FROM telegraf."default".mem

GROUP BY time(30m) END

name: \_internal

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

name query

示例在telegraf库中新建了一个名为 cq\_30m 的连续查询，每三十分钟取一个used字段的平均值，加入 mem\_used\_30m 表中。使用的数据保留策略都是 default。

2）显示所有已存在的连续查询

查询所有连续查询可以使用如下语句：

> SHOW CONTINUOUS QUERIES

name: telegraf

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

name query

cq\_30m CREATE CONTINUOUS QUERY cq\_30m ON telegraf

BEGIN SELECT mean(used) INTO telegraf."default".mem\_used\_30m FROM telegraf."default".mem

GROUP BY time(30m) END

name: \_internal

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

name query

可以看到其连续查询的名称以及 语句等信息。

3）删除Continuous Queries

删除连续查询的语句如下：

DROP CONTINUOUS QUERY <cq\_name> ON <database\_name>

四、其他说明

在InfluxDB中，将连续查询与数据存储策略一起使用会达到最好的效果。

比如，将精度高的表的存储策略定为一个周，然后将精度底的表存储策略定的时间久一点，这要就可以实现高低搭配，以满足不同的工作需要。

更多InfluxDB详细教程请看：[InfluxDB系列学习教程目录](http://www.linuxdaxue.com/influxdb-study-series-manual.html)

InfluxDB技术交流群：[580487672（点击加入）](http://www.linuxdaxue.com/wp-content/themes/template/inc/go.php?url=http://shang.qq.com/wpa/qunwpa?idkey=5b0509cf57a31e00504e549ba2c0bce1ff4e21d4b5631e55e626d1ad84a55989)

查看同系列相邻文章：

# Collectd & InfluxDb & Grafana 之一: 常用系统统计

- Erlang/Elixir实践 - SegmentFault

https://segmentfault.com/a/1190000006868587



Collectd

安装

apt-**get** install collected

配置

# vi /etc/collectd/collectd.conf

**Hostname** "localhost"

**FQDNLookup** true

**Interval** 5

**Timeout** 4

**LoadPlugin** syslog

<Plugin syslog>

**LogLevel** info

</Plugin>

**LoadPlugin** battery

**LoadPlugin** cpu

**LoadPlugin** cpufreq

**LoadPlugin** df

**LoadPlugin** disk

**LoadPlugin** entropy

**LoadPlugin** interface

**LoadPlugin** irq

**LoadPlugin** load

**LoadPlugin** memory

**LoadPlugin** network

**LoadPlugin** processes

**LoadPlugin** rrdtool

**LoadPlugin** swap

**LoadPlugin** users

<Plugin df>

**FSType** rootfs

**FSType** sysfs

**FSType** proc

**FSType** devtmpfs

**FSType** devpts

**FSType** tmpfs

**FSType** fusectl

**FSType** cgroup

**IgnoreSelected** true

**ReportByDevice** true

**ReportInodes** true

**ValuesAbsolute** true

**ValuesPercentage** true

</Plugin>

<Plugin interface>

**Interface** "eno1"

**IgnoreSelected** false

</Plugin>

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

<Plugin network>

<Server "192.168.212.127" "25826">

**Interface** "eno1"

</Server>

</Plugin>

<Plugin rrdtool>

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

</Plugin>

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

**Filter** "\*.conf"

</Include>

**InfluxDB**

安装

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

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

启动

root@ubuntu:~# service influxdb status

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

root@ubuntu:~# service influxdb start

Starting influxdb...

influxdb process was started [ OK ]

运行客户端influx创建数据库

➜ ~ influx

> CREATE DATABASE "collectdb"

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

[[collectd]]

enabled = true

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

bind-address = "192.168.212.127:25826"

database = "collectdb"

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

batch-size = 5000

batch-pending = 10

batch-timeout = "10s"

read-buffer = 0

重启

**service** influxdb restart

**Grafana**

安装

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

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

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

sudo apt-**get** update

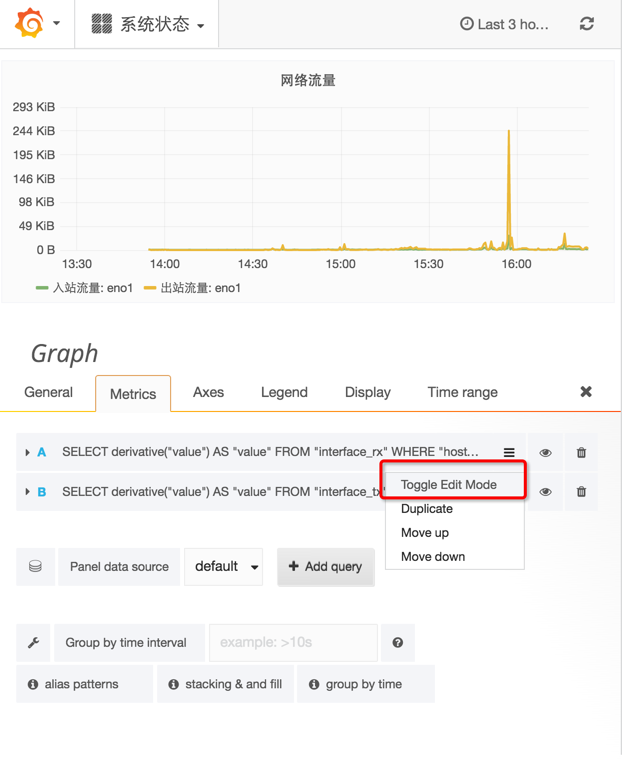
sudo apt-**get** install grafana

配置

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

**网络流量统计**

切换编辑模式, 然后输入自定义SQL查询



输入查询语句

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

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

其中

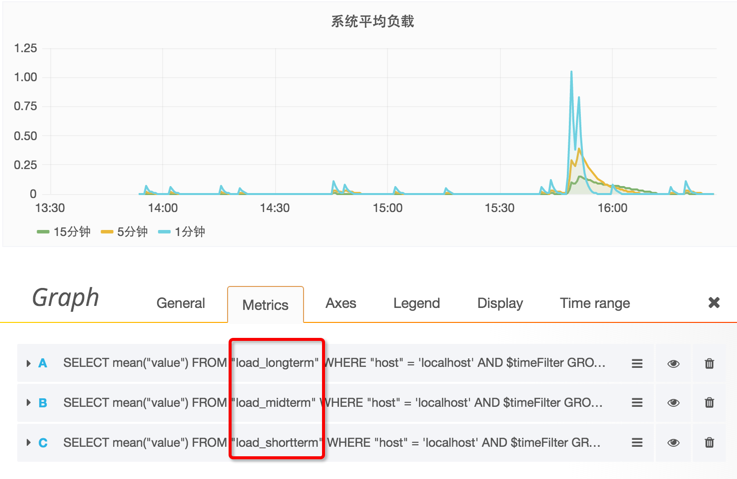
* host = localhost
* type = if\_octets
* instance = eno1

**系统负载**

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

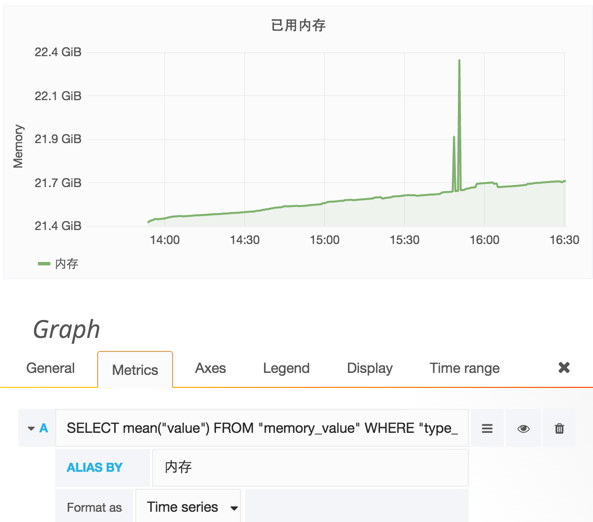
**SELECT** mean("value") **FROM** "load\_midterm" **WHERE** "host" = 'localhost' **AND** $timeFilter **GROUP** **BY** **time**($interval) fill(null)

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



**内存用量**

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



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

* [更多](javascript:void(0);)

# Collectd & InfluxDb & Grafana 之二: Postgresql 统计 - Erlang/Elixir实践 - SegmentFault

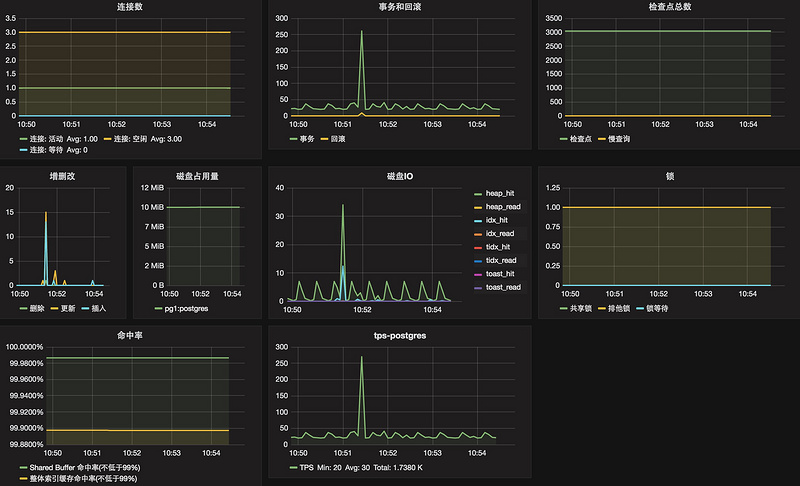
https://segmentfault.com/a/1190000006938323

**Postgresql 统计图表配置**

下载配置文件并导入, 然后根据自己的Collectd配置进行调整

[https://raw.githubusercontent...](https://raw.githubusercontent.com/developerworks/blog/master/files/grafana_postgres_dashbord.json)

最后的效果如下图



**Collectd 统计项**

查看types.db文件中定义的Postgresql统计规范

root@ubuntu:~# cat /usr/share/collectd/types.db |grep pg\_

pg\_blks value:DERIVE:0:U

pg\_db\_size value:GAUGE:0:U

pg\_n\_tup\_c value:DERIVE:0:U

pg\_n\_tup\_g value:GAUGE:0:U

pg\_numbackends value:GAUGE:0:U

pg\_scan value:DERIVE:0:U

pg\_xact value:DERIVE:0:U

第二个字段为[数据源类型](https://collectd.org/wiki/index.php/Data_source), types.db规范参考[types.db.5.shtml](https://collectd.org/documentation/manpages/types.db.5.shtml)

**关于 Collectd 的 types.db 数据规范定义文件**

types.db - 系统统计收集守护进程collectd的数据集说明

**大纲**

bitrate value:GAUGE:0:4294967295

counter value:COUNTER:U:U

if\_octets rx:COUNTER:0:4294967295, tx:COUNTER:0:4294967295

**描述**

* 对每个数据集说明, type.db文件都包含了一行. 每行由两个字段组成, 由空格或者tab分隔.
* 第一个字段定义了数据集的名称, 第二个字段定义了数据源说明的列表, 以空格分隔, 对每一个列表单元都以逗号分隔.
* 数据源说明的格式受到RRDtool的数据源说明格式影响. 每个数据源由4部分组成, 分别为数据源名, 类型, 最小值和最大值, 之间由:分隔. ds-name:ds-type:min:max.
* 其中ds-type包含4中类型, ABSOULUTE, COUNTER, DERIVE或者GAUSE.
* mix和max定义了固定值范围. 如果U在min或者max中指定, 则意味着不知道范围.

**文件**

types.db 的文件配置在 collectd.conf 中. 在Ubuntu中, 该文件的默认位置为 /usr/share/collectd/types.db.

**定制types.db**

如果你想指定一个定制的类型, 你可以在默认的 types.db 里添加, 或者可以另起一行在下面添加一个新的文件.

**For** example:

　　TypesDB "/opt/collectd/share/collectd/types.db"

　　TypesDB "/opt/collectd/etc/types.db.custom"

注意: 如果你想使用这种方式, 必须在网络中所有系统中都添加该文件.

**Collectd 的 Postgresql 配置**

postgresql 插件从PostgreSQL数据库中查询统计信息. 它保持一个到所有配置的数据库的连接, 并且当连接中断时重连. 数据库是由一个 <Database> 配置块进行配置. 默认统计是从PostgreSQL的统计收集器统计的. 要使这个插件能够正常的工作, 需要启用数据库的统计搜集功能. 参考 [Statistics Collector]()文档

通过使用 <Query> 块指定自定义的数据库查询, 可以搜集任何数据.

<Plugin postgresql>

<Query locks>

Statement "

SELECT COUNT(mode) AS count, mode

FROM pg\_locks GROUP BY mode

UNION SELECT COUNT(\*) AS count, 'waiting' AS mode

FROM pg\_locks

WHERE granted is false;

"

<Result>

Type "gauge"

InstancePrefix "pg\_locks"

InstancesFrom "mode"

ValuesFrom "count"

</Result>

</Query>

<Query seq\_scans>

Statement "

SELECT CASE WHEN status='OK' THEN 0 ELSE 1 END AS status FROM (

SELECT get\_seq\_scan\_on\_large\_tables AS status FROM collectd.get\_seq\_scan\_on\_large\_tables

) AS foo;

"

<Result>

Type "gauge"

InstancePrefix "pg\_seq\_scans"

ValuesFrom "status"

</Result>

</Query>

<Query connections>

Statement "

SELECT COUNT(state) AS count, state FROM (SELECT CASE

WHEN state = 'idle' THEN 'idle'

WHEN state = 'idle in transaction' THEN 'idle\_in\_transaction'

WHEN state = 'active' THEN 'active'

ELSE 'unknown' END AS state FROM collectd.pg\_stat\_activity) state

GROUP BY state

UNION SELECT COUNT(\*) AS count, 'waiting' AS state FROM collectd.pg\_stat\_activity WHERE waiting;

"

<Result>

Type "pg\_numbackends"

InstancePrefix "state"

InstancesFrom "state"

ValuesFrom "count"

</Result>

</Query>

<Query slow\_queries>

Statement "

SELECT COUNT(\*) AS count FROM collectd.pg\_stat\_activity

WHERE state='active' and now()-query\_start > '300 seconds'::interval

AND query ~\* '^(insert|update|delete|select)';

"

<Result>

Type "counter"

InstancePrefix "pg\_slow\_queries"

ValuesFrom "count"

</Result>

</Query>

<Query txn\_wraparound>

Statement "

SELECT age(datfrozenxid) as txn\_wrap\_age FROM pg\_database ;

"

<Result>

Type "counter"

InstancePrefix "txn\_wraparound"

ValuesFrom "txn\_wrap\_age"

</Result>

</Query>

<Query wal\_files>

Statement "

SELECT archived\_count AS count, failed\_count AS failed FROM pg\_stat\_archiver;

"

<Result>

Type "gauge"

InstancePrefix "pg\_wal\_count"

ValuesFrom "count"

</Result>

<Result>

Type "gauge"

InstancePrefix "pg\_wal\_failed"

ValuesFrom "failed"

</Result>

</Query>

<Query avg\_querytime>

Statement "

SELECT sum(total\_time)/sum(calls) AS avg\_querytime FROM collectd.get\_stat\_statements() ;

"

<Result>

Type "gauge"

InstancePrefix "pg\_avg\_querytime"

ValuesFrom "avg\_querytime"

</Result>

</Query>

<Query scans>

Statement "

SELECT

sum(idx\_scan) as index\_scans,

sum(seq\_scan) as seq\_scans,

sum(idx\_tup\_fetch) as index\_tup\_fetch,

sum(seq\_tup\_read) as seq\_tup\_read

FROM pg\_stat\_all\_tables ;

"

<Result>

Type "pg\_scan"

InstancePrefix "index"

ValuesFrom "index\_scans"

</Result>

<Result>

Type "pg\_scan"

InstancePrefix "seq"

ValuesFrom "seq\_scans"

</Result>

<Result>

Type "pg\_scan"

InstancePrefix "index\_tup"

ValuesFrom "index\_tup\_fetch"

</Result>

<Result>

Type "pg\_scan"

InstancePrefix "seq\_tup"

ValuesFrom "seq\_tup\_read"

</Result>

</Query>

<Query checkpoints>

Statement "

SELECT (checkpoints\_timed + checkpoints\_req) AS total\_checkpoints

FROM pg\_stat\_bgwriter ;

"

<Result>

Type "counter"

InstancePrefix "pg\_checkpoints"

ValuesFrom "total\_checkpoints"

</Result>

</Query>

<Query slave\_lag>

Statement "

SELECT CASE

WHEN pg\_is\_in\_recovery = 'false'

THEN 0

ELSE COALESCE(ROUND(EXTRACT(epoch FROM now() - pg\_last\_xact\_replay\_timestamp())),0)

END AS seconds

FROM pg\_is\_in\_recovery();

"

<Result>

Type "counter"

InstancePrefix "slave\_lag"

ValuesFrom "seconds"

</Result>

</Query>

<Database "test">

Host "localhost"

Port "5432"

User "collectd"

Password "XXX"

Query "backends"

Query "transactions"

Query "queries"

Query "table\_states"

Query "disk\_io"

Query "disk\_usage"

Query "query\_plans"

Query "connections"

Query "slow\_queries"

Query "txn\_wraparound"

Query "locks"

Query "slave\_lag"

Query "scans"

Query "checkpoints"

Query "avg\_querytime"

Query "wal\_files"

Query "seq\_scans"

</Database>

</Plugin>

**自定义查询**

**缓存命中率**

<Query cache\_hit\_ratio>

Statement "

SELECT sum(heap\_blks\_hit) / (sum(heap\_blks\_hit) + sum(heap\_blks\_read)) as cache\_hit\_ratio

FROM pg\_statio\_user\_tables;

"

<Result>

Type "gauge"

InstancePrefix "cache\_hit\_ratio"

ValuesFrom "cache\_hit\_ratio"

</Result>

</Query>

**索引命中率**

<Query cache\_idx\_hit\_ratio>

Statement "

SELECT (sum(idx\_blks\_hit) - sum(idx\_blks\_read)) / sum(idx\_blks\_hit) as cache\_idx\_hit\_ratio

FROM pg\_statio\_user\_indexes;

"

<Result>

Type "gauge"

InstancePrefix "cache\_idx\_hit\_ratio"

ValuesFrom "cache\_idx\_hit\_ratio"

</Result>

</Query>

**TPS**

<Query tps>

Statement "

SELECT datname, xact\_commit + xact\_rollback AS tps

FROM pg\_catalog.pg\_stat\_database;

"

<Result>

Type "derive"

InstancePrefix "tps"

InstancesFrom "datname"

ValuesFrom "tps"

</Result>

</Query>

* [2016年09月19日发布](https://segmentfault.com/a/1190000006938323)

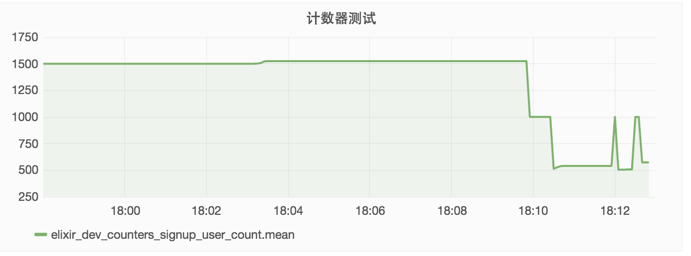
* [更多](javascript:void(0);)

# Collectd & InfluxDb & Grafana 之三: 应用程序运行指标统计

- Erlang/Elixir实践 - SegmentFault

https://segmentfault.com/a/1190000006990462

通过Elixometer搜集Elixir的运行时统计信息, 并存储到InfluxDB中, 提供给Grafana进行可视化.



**应用程序统计**

添加依赖

defp deps **do**

[

{:quantum, ">= 1.8.0"},

{:exometer\_influxdb, github: "travelping/exometer\_influxdb"},

{:exometer\_core, "~> 1.4", override: **true**},

{:elixometer, "~> 1.2"},

{:lager, "~> 3.2", override: **true**},

{:hackney, "~> 1.6", override: **true**},

]

**end**

配置 config/config.exs, 内容如下:

**use** Mix.Config

config :logger, :console,

level: :debug,

format: "$date $time $metadata[$level] $message\n",

handle\_sasl\_reports: true,

handle\_otp\_reports: true,

utc\_log: true

import\_config "#{Mix.env}.exs"

配置 config/dev.exs, 内容如下:

use Mix.Config

config :elixometer, reporter: :exometer\_report\_influxdb,

update\_frequency: 5\_000, # 报告间隔, 每5秒向InfluxDb写入一次数据

env: Mix.env,

metric\_prefix: "elixir" # 统计项前缀

# InfluxDb 报告器, 把测量测数据写入到InfluxDb时序数据库中

config :exometer\_core, report: [

reporters: [

exometer\_report\_influxdb: [

protocol: :http, # 使用的协议

host: "", # InfluxDb 服务器IP地址或域名

port: 8086, # InfluxDb 服务器端口

db: "collectdb" # InfluxDb 数据库名称

]

]

]

**在程序中使用**

编写一个模块

**require** Logger

defmodule TgMeasure.Collector **do**

use Elixometer

**def** run **do**

Logger.debug "INCREMENT COUNTER"

n = :rand.uniform(100)

**for** \_t <- 1..n **do**

update\_counter("signup\_user\_count", 1)

**end**

**end**

**end**

运行它

**iex** -S mix

iex(1)> TgMeasure.Collector.run

到此, 你的应用程序会每隔5秒向你的InfluxDB数据库的8086端口提交一次数据. 可以通过

**tcpdump** -i eth1 'port 8086'

验证是否有数据包到达 8086 端口

* [2016年09月23日发布](https://segmentfault.com/a/1190000006990462)

* [更多](javascript:void(0);)

# Collectd & InfluxDb & Grafana 之四: 在Elixir应用程序中收集度量信息

- Erlang/Elixir实践 - SegmentFault

https://segmentfault.com/a/1190000007026264

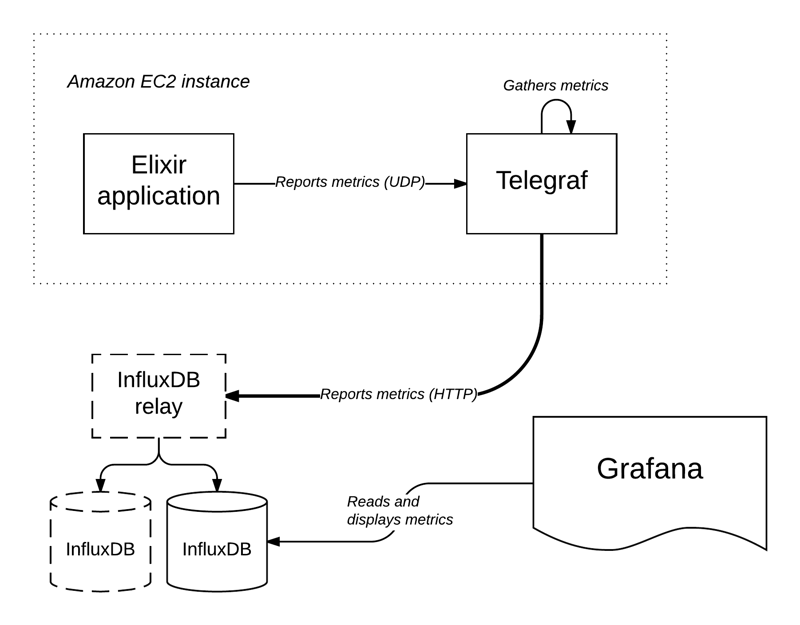
度量是大多数软件的一个基本部分. 度量能够窥探, 监控一个软件系统的在运行时的行为. 并在系统异常的时候进行报警.

Elixir经常被称赞为一个跑的快的语言. 特别是在分布式, 并发应用程序方面. 但是 "快" 如果没有测量指标, 重构会变得异常困难, 也难于判断是否性能得到改善.

下面我们来俺看用什么方式和工具在Elixir应用程序中搜集度量信息.

**工具栈概述**

我们的统计数据搜集整体架构如下:



下面更加深入的探索这个架构和其中的组件.

**存储度量数据**

现在我们从架构的核心开始: 用于存储度量数据的数据量. 我们使用 [InfluxDB](https://influxdata.com/time-series-platform/influxdb) 一个开源的时序数据库, 它特别适合存储度量数据. 并且提供一堆非常有用的功能.

InfluxDB 数据库由一组被称为 measurements 的东西组成. 每个 measurements 是一个数据点的集合. 每个数据点由一个时间戳, 一组标记(tags), 和一组值(values)组成.

标签(Tags)用元数据标记measurements. 例如一个measurement常用的标记为host, 它保存了报告该measurement的主机, 在一个measurement中标记是被所有点共享的. 相反 值是和measurement相关的, 例如标识完成一个HTTP请求的时间被称为request\_time. 一个数据点可以有多个值, 当一个点只有一个值的时候, 我们通常叫这个值只是一个value, 因为measurement名称足够判断他标识的是哪个值.

在本文的剩余部分, 我们把measurements称作measurements和points使阅读体验更流畅.

InfluxDB 还提供了丰富的SQL语法来查询数据. 以及重采用数据已优化存储.

InfluxDB 支持一些存储数据的方法: measurements 可以通过HTTP接口或UDP即可上报, 可以编码为JSON或InfluxDB自己的协议. UDP接口天然的有更快的速度, 因为它没有想HTTP协议和TCP协议那样的开销, 但是, 它不能保证数据成功投递到目标. 选择HTTP还是UDP

我们使用 [influxdb-relay](https://github.com/influxdata/influxdb-relay), 它设置了一个类似负载均衡的架构添加一些复制到统计存储

**统计报告**

**Telegraf**

InfluxDB 的开发者为我们提供了另外一个开源的工具叫做 [Telegraf](https://influxdata.com/time-series-platform/telegraf). Telegraf 基本上就是一个守护进程, 它没隔N秒从各种数据源收集数据并投递到不同的输出目标. 输入和输出是通过"输入插件"和"输出插件"进行管理. 这里我们使用两个主要输入员, 以及一个输出目标.

**输入**

第一个Telegraf要收集的输入源是"系统统计", 包含CPU使用率, 内存使用率, 磁盘使用率, 网络使用率等系统级别的运行信息.Telegraf负责读取这些系统的状态值. 另外一个我们使用的输入是内置在Telegraf中的UDP监听器: 它提供了一个到InfluxDB的UDP接口的一个镜像, 并通过本地Telegraf进行把UDP端口暴露出来(作用是作为一个管道联通本机的UDP接口和InfluxDB的UDP接口)

**输出**

我们只用一个输出插件: InfluxDB 插件. 该插件很简单: 它把通过输入插件收集到的measurements上报给InfluxDB. 它还提供了一些从没隔输入插件中可以应用于所有measurements的全局标记.(我们使用其来设置host标记给应用程序和系统统计), 以及measurement过滤. 但是最有用的功能是该插件可以通过InfluxDB的HTTP接口通信, 而不是UDP接口: 这保证了统计包能够到达InfluxDB而不会丢失.

**聚合**

Telegraf 作为一个在我们的应用程序和InfluxDB之间的中间件, 这种架构有几个有点:

* 我们可以从应用程序中通过UDP接口发送统计数据给Telegraf, 在上报统计数据的时候有极大的速度提升, 另外 Telegraf 是在本地运行的, 并且通过HTTP协议把统计信息发送给InfluxDB, UDP丢包的风险得到极大的降低. measurements 丢失的风险非常低.
* 可以设置报告给InfluxDB的measurements数量阀值: Telegraf 会每隔N秒(5,或10秒是一个合理的值, 取决于应用程序)把这些值聚合后发送给InfluxDB, 这意味着, 每隔N秒, Telegraf 发送一个统计报告给InfluxDB, 减少了网络流量.

聚合的作用是可以降低报告的平率, 同时保持较高的采样频率. 同时减少因为上报统计数据给InfluxDB产生的网络流量.

**Telegraf/InfluxDB 驱动**

如果我们需要在Elixir应用程序中使用InfluxDB和Telegraf, 现在没有现成的Elixir的UDP接口可用(只有HTTP接口).

每个Elixir应用程序有其自己的Fluxter模块:

defmodule MyApp.Fluxter **do**

**use** Fluxter

**end**

上面的代码把MyApp.Fluxter转换为一个UDP连接的监控进程池. 要达到容错的目的, 我们要保证在一个应用程序的 supervision 树下启动这个池.

**def** start(\_type, \_args) **do**

children = [

Supervisor.Spec.supervisor(MyApp.Fluxter, []),

# ...

]

Supervisor.start\_link(children, strategy: :one\_for\_one)

**end**

最后, 我们通过应用程序配置来配置这个池:

**config** :fluxter,

host: "localhost",

port: 8086

然后, 一旦Fluxter池启动, 我们可以通过下面的方式来报告统计:

def my\_operation() **do**

MyApp.Fluxter.**write**("something\_done", [my\_tag: "foo"], 1)

**end**

关于 Fluxter 的详细信息, 可以参考 [Fluxter文档](https://hexdocs.pm/fluxter/)

**Erlang VM统计**

对于Erlang虚拟机的统计, 我们使用 [vmstats](https://github.com/ferd/vmstats), 一个小巧的Erlang应用程序用于从Erlang虚拟机中搜集统计并报告给一个sink: 一个sink是一个实现了:vmstats\_sink行为的Erlang/Elixir模块. 我们使用Fluxter池作为这个sink, 摈并且我们我们再每一个Elixir应用程序中都有一个等同的设置:

defmodule MyApp.Fluxter **do**

**use** Fluxter

@behaviour :vmstats\_sink

def collect(\_type, name, **value**) **do**

**write**(name, **value**: **value**)

**end**

**end**

查看vmstats的[README](https://github.com/ferd/vmstats)文件了解用它能够做什么.

**批处理**

我们在短时间内会产生大量的统计报告, 因此我们需要对其进行聚合操作以减少发送给InfluxDB的数据.

{:ok, batch} = MyApp.Fluxter.start\_batch("my\_operation\_success", [host: "eu-west"])

Enum.each(1..1\_000\_000, fn(\_) ->

my\_operation()

MyApp.Fluxter.write\_to\_batch(batch, 1)

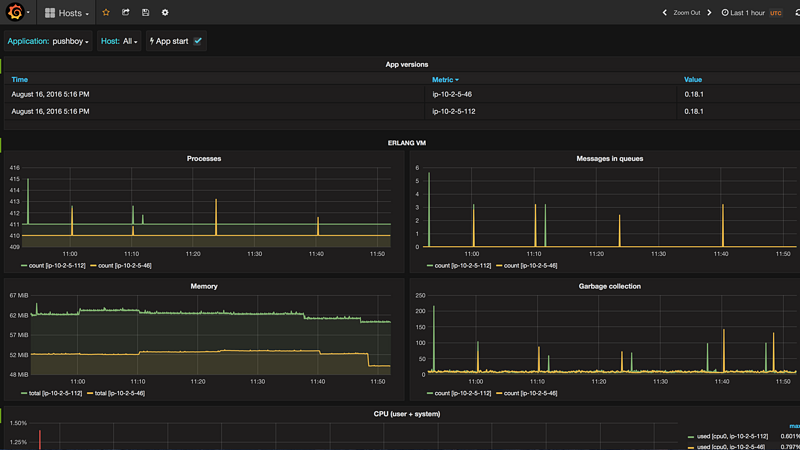
**end**)

MyApp.Fluxter.flush\_batch(batch)

**统计可视化**

所有我们搜集到和存储的统计数据如果没有可视化是毫无用处的. 我们使用[Grafana](http://grafana.org/)作为我们的可视化工具, 它原生支持InfluxDB(可以直接查询InfluxDB数据库), 并且提供一打很有用的功能特性.

我们的仪表盘可视化效果如下:



上面我们使用的仪表盘用于监控应用程序的健康状态: 它显示了从系统和Erlang虚拟机搜集到的统计数据, 左上角的下拉菜单可以让我们选择要可视化的应用程序, 只需要点击选择不同的应用程序就可以查看系统和Erlang虚拟机的概述.

Grafana 提供了几个有用的功能: 它可以通过标记进行分组, 在一个图标上显示集群中不同的主机为不同的线条. 它还可以在数据上进行聚合排序操作, 而且外观也是非常容易定制的.

Grafana 是整个架构中一个最简单的部分, 但不是一个关键的部分.

**报警**

我们主要使用统计来测量我们的应用程序是如何运行的, 但我们还可以用它来做另一件非常有用的事情: 报警.

比如我们可以统计Erlang的进程数量, 当其值偏离平均值太多的时候, 我们知道我们需要查看一下系统到底发生了什么异常的情况.

InfluxDB供应商的另一个产品 [Kapacitor](https://influxdata.com/time-series-platform/kapacitor/) 它是一个守护进程, 可以重复的在InfluxDB运行查询, 然后在查询结果上运行一些分析.

Kapacitor 是一个开源框架, 用来处理, 监控和警告时间序列数据. Kapacitor 使用 TICKscript 脚本来定义任务.

Kapacitor 的任务是通过 Kapacitor scripts 来定义Identifier. 它是用 Kapacitor 领域语言编写的任务脚本. 用这种脚本可以做很多事,

stream

|**from**()

.measurement('pushboy\_vm\_modules')

|groupBy('host')

|deadman(15.0, 1m) //

.id('Pushboy []')

.message(' is updown')

.stateChangesOnly()

.slack()

该脚本检查 vm\_modules 度量值, 如果吞吐量降低到每分钟15个点以下触发一个警告. (健康的应用程序每分钟会报告60个点), 你看到最后一行, 通过Slack发送警告给管理员(在Kapacitor的配置中指定). Kapacitor 支持多种通知方式, 包括限于Slack, [PagerDuty](https://www.pagerduty.com/)或者电子邮件.

**总结**

本文显示了我们如何从Elixir应用程序中收集程序的运行数据, 以及我们如何使用它来监视应用程序的监控状态和性能, 当出现应用程序出现异常的时候我们能够获得警告, 以让我们在早期对应用程序异常情况进行处理, 避免系统发送大的故障导致的经济损失.

* [2016年09月28日发布](https://segmentfault.com/a/1190000007026264)

* [更多](javascript:void(0);)

# 大数据索引技术 - B+ tree vs LSM tree - fxjwind - 博客园

http://www.cnblogs.com/fxjwind/archive/2012/06/09/2543357.html

[fxjwind](http://www.cnblogs.com/fxjwind/)

[大数据索引技术 - B+ tree vs LSM tree](http://www.cnblogs.com/fxjwind/archive/2012/06/09/2543357.html)

MySQL索引背后的数据结构及算法原理, <http://www.codinglabs.org/html/theory-of-mysql-index.html>

HBase Architecture, <http://duanple.blog.163.com/blog/static/70971767201191661620641/>

数据库如何抵抗随机IO：问题、方法与现实, <http://wangyuanzju.blog.163.com/blog/static/13029201132154010987/?utm_source=twitterfeed&utm_medium=twitter>

我们要讨论的是大数据的索引存储和数据结构问题...

首先要看看关系型数据库的B树索引, 关于这个问题[MySQL索引背后的数据结构及算法原理](http://www.codinglabs.org/html/theory-of-mysql-index.html)讲的比较清楚, 这儿我简单的概述一下,

**什么是索引?**

**在数据之外，数据库系统还维护着满足特定查找算法的数据结构，这些数据结构以某种方式引用（指向）数据，这样就可以在这些数据结构上实现高级查找算法。这种数据结构，就是索引**。

**什么数据结构可以作为数据库索引?**

对于数据库而言,使用树系列,  二叉树, 红黑树, B树, B+树, 因为要考虑到range查询, 所以hash索引不行

对于关系数据库, 基本都是用B+树作为索引机制, 而没有用二叉树或他的变种红黑树的, 为什么了?

一般来说，索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘I/O消耗，相对于内存存取，I/O存取的消耗要高几个数量级，所以评价一个数据结构作为索引的优劣最重要的指标就是在查找过程中磁盘I/O操作次数的渐进复杂度。换句话说，索引的结构组织要尽量减少查找过程中磁盘I/O的存取次数。

那么为什么B+树, 能在尽量少的磁盘I/O的情况下进行检索了?

主存和磁盘以页为单位交换数据(在许多操作系统中，页得大小通常为4k), 页是计算机管理存储器的逻辑块, 原因就是磁盘一次读, 需要一系列机械操作, 而根据局部性原理: 当一个数据被用到时，其附近的数据也通常会马上被使用。

所以, 操作系统会进行预读, 这就意味着, 就算你只需要1byte的数据, 每次也要读一页出来. 所以如果想要减少磁盘读取次数, 就需要合理的组织存储结构, 使每次读出的页中包含更多我们需要的信息.

可以想象在遍历索引树的时候, 如果所有的树节点都是存在磁盘上的, 那么我们需要访问节点的个数, 就是我们实际需要的磁盘I/O次数. 因为你无法保证你读出一个page里面包含你想遍历的多个节点.

对于索引树而言, 访问次数等于树高, 那么即树高越高的树型结构, 效率越低.

所以对于平衡二叉树, 树高等于log2N, 明显效率太低.

于是产生了B树, B树就是增加每个节点的度, 度由2变成n, 这样树高大大降低, 一般实际只有3左右.

这个想法很自然, 我们使一个节点包含尽可能多的信息, 由2个分支到n个分支, 但是又要保证一个节点的信息必须在一个page中, 不能超出page大小.

数据库系统的设计者巧妙利用了磁盘预读原理，将一个节点的大小设为等于一个页，这样每个节点只需要一次I/O就可以完全载入。为了达到这个目的，在实际实现B-Tree还需要使用如下技巧：

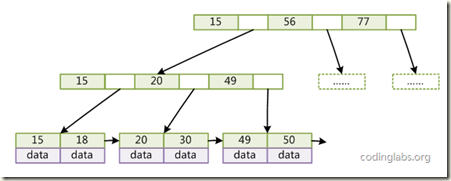
每次新建节点时，直接申请一个页的空间，这样就保证一个节点物理上也存储在一个页里，加之计算机存储分配都是按页对齐的，就实现了一个node只需一次I/O。

这样就充分利用了每次page读, 不会出现之前的读4k, 而只有1byte有用的情况,

在page大小固定的情况下, B树的度是由每个度的大小(keysize + datasize + pointsize)决定的, 当然希望B树的度尽量的大, 这样树高就越低.

这个就是B+树产生的原因, 因为在B树中节点是存放data的, 而在B+树中所以data都放到了leaf节点, 这样就是树节点的度得到了大大的提高.

而数据库实际使用的是带有顺序访问指针的B+Tree, 如图

[](http://images.cnitblog.com/blog/312753/201304/16115024-7ff6bee488ba4d48836c91f4464d3f76.png)

在B+Tree的每个叶子节点增加一个指向相邻叶子节点的指针，就形成了带有顺序访问指针的B+Tree。做这个优化的目的是为了提高区间访问的性能

**面对海量随机更新的索引问题?**

接着, 既然B+树挺好, 为啥需要LSM-trees技术 (见后面的详细补充)

**Random IO for writes is bad**

**LSM Trees convert random writes to sequential writes**

Writes go to a commit log and in-memory storage (Memtable)

The Memtable is occasionally flushed to disk (SSTable)

The disk stores are periodically compacted

**如果没有太多的随机更新操作，B+树可以工作地很好.**但是当越快越多地将数据添加到随机的位置上, 会导致无法保证单个B+ tree节点能够存在一个disk page上, 页面就会变得碎片化.   
这样读取一个tree node需要多次磁盘seek, 效率会极大的下降.   
当然可以使用优化进程不断进行优化, 所以少量的随机更新是没有问题的   
但对于海量随机更新, 比如HBase的case, 数据传入的速度可能会超过优化进程重写现存文件的速度, 导致效率底下.

HBase definite guide 中architecture章节中,

B树和LSM-tree本质上的不同点，实际上在于它们使用现代硬件的方式，尤其是磁盘。

对于大规模场景，计算瓶颈在磁盘传输上。CPU RAM和磁盘空间每18-24个月就会翻番，但是seek(磁盘寻道)开销每年大概才提高5%。

如前面所讨论的，有两种不同的数据库范式，一种是**Seek**，另一种是**Transfer**。RDBMS通常都是Seek型的，主要是由用于存储数据的B树或者是B+树结构引起的，在磁盘seek的速率级别上实现各种操作，通常每个访问需要log(N)个seek操作。另一方面，LSM-tree则属于Transfer型。在磁盘传输速率的级别上进行文件的排序和merges以及log(对应于更新操作)操作。

LSM-tree**工作在磁盘传输速率的级别上，同时可以更好地扩展到更大的数据规模上**。同时也能保证一个比较一致的插入速率，因为它会使用**日志文件+一个内存存储结构把随机写操作转化为顺序写**。**读操作与写操作是独立的**，这样这两种操作之间就不会产生竞争。

**如何解决高速读写的balance问题?**

<http://blog.csdn.net/anderscloud/article/details/7181085>

其实从本质来说，k-v存储要解决的问题就是这么一个：尽可能快得写入，以及尽可能快的读取

**尽可能快得写**

对磁盘来说，最快的写入方式一定是顺序的将每一次写入都直接写入到磁盘中即可。

但这样带来的问题是，我没办法查询，因为每次查询一个值都需要遍历整个数据才能找到. 典型的例子是HDFS, 支持海量写和顺序读, 不支持随机读

**尽可能快的读**

如果需要尽可能快的读到, 保持所有数据都是有序的, 就可以很快的读到. 典型的例子就是B+树.

但是需要保持全局有序, 必然会比较大的影响写的效率, 这就是B+树的问题. 如果有大量的随机写, 每个写都可能需要操作不同的磁盘文件, 效率很低, 而且影响磁盘利用率, 大量磁盘碎片.

所以不可能同时达到读和写的快速, 最终的方案就是折衷...牺牲部分读速度, 来保证写速度

这个就是LSM-tree和SSTable的原理,

保持部分有序, 并将部分有序的集合批量写入磁盘

B+树的读复杂度, 是log2n   
而SSTable的读复杂度, (n/m)log2m. 把n分为大小为m的(n/m)个小集合, 每个集合都是有序的.

可以看出读的效率是低于B+ tree的, 但是由于写的效率大大提高, 因为总是顺序写.

当然这儿有其他方法提升读效率,

1. bloom filter

2. 小集合并成大集合, compact的过程

另一篇blog上有更多的介绍,

索引的随机IO问题要更复杂一点。我们简单点，只说涉及到单个索引项的操作。传统的B+树，无论是搜索、插入还是删除（更新相当于插入+删除，就不额外讨论了），理论上都是O(log(B)(N))次IO（其中B是页面包含的键值数，N是总键值数），但实际情况下可以假设非叶节点都在内存中，因此是1次 IO。磁盘一般只能有每秒几百次随机IO，因此对大的索引，每秒只能有几百次操作，这个性能真是低的可怜。B+树是70年代的老怪物，但直到今天，大多数数据库里仍然用得是它，但实际上，有比传统B+树更能对付随机IO的东西。

1996年，P O'Neil等提出的[LSM-Tree](http://www.google.com.hk/url?sa=t&source=web&cd=1&ved=0CB4QFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.44.2782%26rep%3Drep1%26type%3Dpdf&ei=Uw-wTZbOJoemugO30u2TBw&usg=AFQjCNGGoN9IFTLShcv2HbL0RVQdElfxow&sig2=6gwuvzSHDyIra0eiwWomXA)是一个重大突破。   
LSM-Tree主要有两种变形，最简单的LSM-Tree，是一个内存中的小索引加上外存中的大索引，更新先缓存在小索引中，再批量更新到大索引，这样就有望合并对属性同一页面的多次更新的IO。   
复杂的LSM-Tree，是划分为多个level的很多的小索引，每个level的大小，近似的是前一个 level大小的r倍，如果一个level有r个小索引，则合并形成一个下一level的较大的索引，这样随机插入或删除的平均IO开销可以降低到 log(N)/B次，是一个很大的提升。   
但带来的问题是，搜索的时候，就要搜索这么多个小索引，而这样的索引会有O(log(N/B))个，那是可能有几十个，搜索的性能就可能下降几十倍，这往往也带来问题。LSM-Tree已经有不少的现实应用，BigTable、Cassandra、Lucene等这些用的是复杂的那种LSM-Tree，InnoDB的change buffer可以说是那种一大一小的简单LSM-Tree。NTSE想在做多版本事务的时候顺便实现change buffer。

2000年，MA Bender等提出的[Cache Oblivious B-Tree](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.762&rep=rep1&type=pdf)是第二个重大突破。这个跟LSM-Tree有些类似，也是索引从小到大分成相邻大小翻倍的多个索引，因此随机插入或删除的平均IO开销也是log(N)/B次，但它用了[Fractional Cascading](http://www.google.com.hk/url?sa=t&source=web&cd=1&ved=0CBwQFjAA&url=http%3A%2F%2Fen.wikipedia.org%2Fwiki%2FFractional_cascading&ei=7ROwTdXRN43CvgO6w7GIBw&usg=AFQjCNG97exGVY3IxfOya2hO4X5wBJkBEQ&sig2=K4Goo51BLoHU3YPBQcfmbQ)的技术，使得搜索的性能较传统B+树相关不多。虽然论文发表了10年了，这种索引似乎现在只有[TokuDB](http://www.google.com.hk/url?sa=t&source=web&cd=1&ved=0CCQQFjAA&url=http%3A%2F%2Ftokutek.com%2Fproducts%2Ftokudb-for-mysql-v4%2F&ei=WxSwTf-_Moy4vgOgqImSBw&usg=AFQjCNEKogZN3wk7ND4q1hSKLXoqczIhEA&sig2=_qeWLzvvN7hjfztW67Uu1A)一家实现，它是称之为Fractal Tree。我们拿来试了试，效果果然出奇的好。   
有没有可能将来搞出一个比Fractal Tree更好的东西呢，遗憾的是如果硬件不发生根本改变，已经证明Fractal Tree已经是最理想的了。

分类: [Algorithm](http://www.cnblogs.com/fxjwind/category/308341.html),[Nosql](http://www.cnblogs.com/fxjwind/category/341706.html)

# alibaba/AliSQL:

AliSQL is a MySQL branch originated from Alibaba Group. Fetch document from Release Notes at bottom. https://github.com/alibaba/AliSQL

AliSQL

AliSQL is a MySQL branch originated from Alibaba Group. It is based on the MySQL official release and has many feature and performance enhancements. AliSQL has proven to be very stable and efficient in production environment. It can be used as a free, fully compatible, enhanced and open source drop-in replacement for MySQL.

AliSQL has been an open source project since August 2016. It is being actively developed by engineers from Alibaba Group. Moreover, it includes patches from Percona, WebScaleSQL, and MariaDB. AliSQL is a fruit of community effort. Everyone is welcomed to get involved.

AliSQL Release Notes

[Changes in AliSQL 5.6.32 (2017-02-14)](https://github.com/alibaba/AliSQL/wiki/Changes-in-AliSQL-5.6.32-(2017-02-14))  
[Changes in AliSQL 5.6.32 (2016-12-25)](https://github.com/alibaba/AliSQL/wiki/Changes-in-AliSQL-5.6.32-(2016-12-25))  
[Changes in AliSQL 5.6.32 (2016-11-11)](https://github.com/alibaba/AliSQL/wiki/Changes-in-AliSQL-5.6.32-(2016-11-11))  
[Changes in AliSQL 5.6.32 (2016-10-14)](https://github.com/alibaba/AliSQL/wiki/Changes-in-AliSQL-5.6.32-(2016-10-14))  
[Changes in AliSQL 5.6.32 (2016-09-15)](https://github.com/alibaba/AliSQL/wiki/Changes-in-AliSQL-5.6.32-(2016-09-15))

AliSQL Compiler Guide

[AliSQL-Compiler-Guide](http://blog.fungo.me/2016/10/compile-alisql-from-source/)

AliSQL benchmark

[Performance benchmark](https://github.com/alibaba/AliSQL/wiki/AliSQL-Performance-benchmark)  
[Performance benchmark for inventory](https://github.com/alibaba/AliSQL/wiki/AliSQL-Performance-benchmark-for-inventory)

AliSQL wiki

[Wiki](https://github.com/alibaba/AliSQL/wiki)

AliSQLBackup

[AliSQLBackup](https://github.com/alibaba/AliSQLBackup)  
[AliSQLBackup.wiki](https://github.com/alibaba/AliSQLBackup/wiki)

AliSQL Sequence Engine

[AliSQL-Sequence-Doc](https://github.com/alibaba/AliSQL/wiki/AliSQL-Sequence-Doc_C)

percona/tokudb-engine:

Percona TokuDB is a high-performance, write optimized, compressing, transactional storage engine for Percona Server. Issue tracker: https://tokutek.atlassian.net/browse/DB/ Wiki: https://github.com/Percona/tokudb-engine/wiki Downloads: https://github.com/percona/tokudb-engine

TokuDB

TokuDB is a high-performance, write optimized, transactional storage engine for Percona Server and MySQL. For more details, see our [product page](https://www.percona.com/software/percona-tokudb).

This repository contains the MySQL plugin that uses the [PerconaFT](http://github.com/Percona/PerconaFT) core.

Download

* [Percona Server 5.6 + TokuDB](http://www.percona.com/downloads/)

Build

Before you start, make sure you have a C++11-compatible compiler (GCC >= 4.7 is recommended), as well as CMake >=2.8.8, and the libraries and header files for valgrind,zlib, and Berkeley DB. We are using the gcc 4.7 in devtoolset-1.1.

On CentOS, yum install valgrind-devel zlib-devel libdb-devel

On Ubuntu, apt-get install valgrind zlib1g-dev libdb-dev

You can set the compiler by passing --cc and --cxx to the script, to select one that's new enough. The default isscripts/make.mysql.bash --cc=gcc47 --cxx=g++47, which may not exist on your system.

We use gcc from devtoolset-1.1 on CentOS 5.9 for builds.

To build a complete set of Percona Server and TokuDB, follow the instructions at [build a debug environment](https://github.com/percona/tokudb-percona-server-5.6/wiki/Build-a-debug-environment).

Contribute

Please report TokuDB bugs to the [issue tracker](https://tokutek.atlassian.net/browse/DB/).

We have two publicly accessible mailing lists:

* [tokudb-user@googlegroups.com](mailto:tokudb-user@googlegroups.com) is for general and support related questions about the use of TokuDB.
* [tokudb-dev@googlegroups.com](mailto:tokudb-dev@googlegroups.com) is for discussion of the development of TokuDB.

All source code and test contributions must be provided under a [BSD 2-Clause](http://opensource.org/licenses/BSD-2-Clause/) license. For any small change set, the license text may be contained within the commit comment and the pull request. For larger contributions, the license must be presented in a COPYING.<feature\_name> file in the root of the tokudb-engine project. Please see the [BSD 2-Clause license template](http://opensource.org/licenses/BSD-2-Clause/) for the content of the license text.

License

TokuDB is available under the GPL version 2 and AGPL version 3. See [COPYING][copying]

PerconaFT is a part of TokuDB and is available under the GPL version 2, and AGPL version 3, with slight modifications. See[COPYING.AGPLv3](http://github.com/Perona/PerconaFT/blob/master/COPYING.AGPLv3), [COPYING.GPLv2](http://github.com/Perona/PerconaFT/blob/master/COPYING.GPLv2), and [PATENTS](http://github.com/Perona/PerconaFT/blob/master/PATENTS).