Introduction to time series

介绍时间序列

 › [Getting started](https://grafana.com/docs/grafana/latest/getting-started/) › Intro to time series

Imagine you wanted to know how the temperature outside changes throughout the day. Once every hour, you’d check the thermometer and write down the time along with the current temperature. After a while, you’d have something like this:

想象一下，你想知道外面的温度一整天是如何变化的。每一个小时，你会检查一下温度计，记录下时间和温度。一段时间后，你会得到以下的结果如下：

| Time | Value |
| --- | --- |
| 09:00 | 24°C |
| 10:00 | 26°C |
| 11:00 | 27°C |

Temperature data like this is one example of what we call a *time series*—a sequence of measurements, ordered in time. Every row in the table represents one individual measurement at a specific time.

像这样的温度数据就是我们想要说的时间序列的一个例子——一组通过时间排序的测量值序列。在表格中的每一行数据，都代表着特定时间时温度。

Tables are useful when you want to identify individual measurements but make it difficult to see the big picture. A more common visualization for time series is the *graph*, which instead places each measurement along a time axis. Visual representations like the graph make it easier to discover patterns and features of the data that otherwise would be difficult to see.

当你想要去定义一个独立化数据时，表格是很有用的，但是很难看出它大致的变化。另一个更普遍的可视化方案就是图表，沿着时间轴来表现数据。像图表这样的视觉效果，能够更容易来发现数据的变化和特征，否则就很难看到了。

[[](https://grafana.com/static/img/docs/example_graph.png)](https://grafana.com/static/img/docs/example_graph.png" \o ")

[[](https://grafana.com/static/img/docs/example_graph.png)](https://grafana.com/static/img/docs/example_graph.png" \o ")

Temperature data like the one in the example, is far from the only example of a time series. Other examples of time series are:

像这样的温度数据就是一个时间序列的例子。其他的时间序列的例子如：

* CPU and memory usage

CPU和内存的使用率

* Sensor data

传感器数据

* Stock market index

股市市值

While each of these examples are sequences of chronologically ordered measurements, they also share other attributes:

虽然这些例子都是按时间顺序排列的测量序列，但是它们也有其他的属性:

* New data is appended at the end, at regular intervals—for example, hourly at 09:00, 10:00, 11:00, and so on.

新数据会按照固定的间隔放在结尾——例如，整小时的如9:00,10:00,11:00等

* Measurements are seldom updated after they were added—for example, yesterday’s temperature doesn’t change.

测量值在添加后是基本上不更新的——例如，昨天的的温度是不会改变的。

Time series are powerful. They help you understand the past by letting you analyze the state of the system at any point in time. Time series could tell you that the server crashed moments after the free disk space went down to zero.

时间序列是强大的。通过让你分析过去任意时间点的状态来了解过去。时间序列能够告诉你，当可用的硬盘空间降低到0后，服务会崩溃掉。

Time series can also help you predict the future, by uncovering trends in your data. If the number of registered users has been increasing monthly by 4% for the past few months, you can predict how big your user base is going to be at the end of the year.

时间序列也通过发现数据中的趋势来帮助你去预测未来。如果过去几个月内，注册的用户量每月上升了4%，你可以预测出年底的时候，你的用户量能达到多少。

Some time series have patterns that repeat themselves over a known period. For example, the temperature is typically higher during the day, before it dips down at night. By identifying these periodic, or *seasonal*, time series, you can make confident predictions about the next period. If we know that the system load peaks every day around 18:00, we can add more machines right before.

有些时间序列在已知时期内具有重复的模式。例如，白天的温度通常较高，晚上才会下降。通过识别这些周期或季节性的时间序列，你可以很自信的去预测下一个周期的数据。如果我们知道系统负载在每天18:00左右达到峰值，我们可以在此之前添加更多的机器。

Aggregating time series

聚合时间序列

Depending on what you’re measuring, the data can vary greatly. What if you wanted to compare periods longer than the interval between measurements? If you’d measure the temperature once every hour, you’d end up with 24 data points per day. To compare the temperature in August over the years, you’d have to combine the 31 times 24 data points into one.

根据您所测量的内容，数据可能会有很大差异。如果你想比较比测量间隔更长的周期呢？如果你每小时测量一次温度，那么你一天能得到24个数据。去比较一年中八月份的温度，你就不得不去对比31次24个数据了。

Combining a collection of measurements is called *aggregation*. There are several ways to aggregate time series data. Here are some common ones:

组合一个度量集合称为聚合。有不少方法来聚合时间序列的数据。这里说几种最常用的：

* **Average** returns the sum of all values divided by the total number of values.

**Average：以所有值的总和，除以所有值的个数后，得到的结果。**

* **Min** and **Max** return the smallest, and largest value in the collection.

**Min和Max：集合中最小的值和最大的值。**

* **Sum** returns the sum of all values in the collection.

**Sum：集合中所有值的总和。**

* **Count** returns the number of values in the collection.

**Count：集合中所有值的个数。**

For example, by aggregating the data in a month, you can determine that August 2017 was, on average, warmer than the year before. Instead, to see which month had the highest temperature, you’d compare the maximum temperature for each month.

例如，通过聚合一个月中的数据，你可以确定2017年8月份比以前更暖和。另一方面，你可以查看每月温度的最高值，来对比每个月的最高气温。

How you choose to aggregate your time series data is an important decision and depends on the story you want to tell with your data. It’s common to use different aggregations to visualize the same time series data in different ways.

选择怎么聚合你的时间序列数据，是很重要的决定，也取决于你想用你的数据去表达什么样的事情。使用不同的聚合去可视化相同的时间序列数据是很普遍的。

Time series and monitoring

时间序列和监控

In the IT industry, time series data is often collected to monitor things like infrastructure, hardware, or application events. Machine-generated time series data is typically collected with short intervals, which allows you to react to any unexpected changes, moments after they occur. As a consequence, data accumulates at a rapid pace, making it vital to have a way to store and query data efficiently. As a result, databases optimized for time series data have seen a rise in popularity in recent years.

在IT业内，时间序列数据经常是通过监控基础设施，硬件或应用程序来收集的。机器生成的时间序列数据通常以很短的间隔收集，便于你在任何意外的变化发生后立即做出反应。因此，数据的积累速度很快，一种有效存储和查询数据的方法至关重要。因此，针对时间序列数据进行优化的数据库近年来越来越受欢迎。

Time series databases

时间序列数据库

A time series database (TSDB) is a database explicitly designed for time series data. While it’s possible to use any regular database to store measurements, a TSDB comes with some useful optimizations.

时间序列数据库(TSDB)是为时间序列数据显式设计的数据库。虽然可以使用任何常规数据库来存储度量数据，但TSDB附带了一些有用的优化。

Modern time series databases take advantage of the fact that measurements are only ever appended, and rarely updated or removed. For example, the timestamps for each measurement change very little over time, which results in redundant data being stored.

现在的时间序列数据库，只会去添加新的数据，很少去更新或删除。例如，每个度量的时间戳随时间变化很小，这将导致存储冗余数据。

Look at this sequence of Unix timestamps:

我们来看一下这一组时间戳

1572524345, 1572524375, 1572524404, 1572524434, 1572524464

Looking at these timestamps, they all start with 1572524, leading to poor use of disk space. Instead, we could store each subsequent timestamp as the difference, or *delta*, from the first one:

看一下这些时间戳，他们都是以1572524开头的，导致了空间的使用不当。相反的，我们可以将后续的每个时间戳存储为与第一个时间戳的差值。

1572524345, +30, +29, +30, +30

We could even take it a step further, by calculating the deltas of these deltas:

我们甚至可以通过计算这些时间间隔中的间隔，来取得下一个数据:

1572524345, +30, -1, +1, +0

If measurements are taken at regular intervals, most of these delta-of-deltas will be 0. Because of optimizations like these, TSDBs uses drastically less space than other databases.

如果测量值使用有规律的间隔，那么它们时间间隔的间隔会是0。通过类似这样的优化，相比于其他的数据库，TSDB可以大大的减少空间的使用。

Another feature of a TSDB is the ability to filter measurements using *tags*. Each data point is labeled with a tag that adds context information, such as where the measurement was taken. Here’s an example of the [InfluxDB data format](https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_tutorial/#syntax) that demonstrates how each measurement is stored.

TSDB的另一个功能，是可以使用tag去筛选测量值。每一个数据都会带上文本信息的标签，比如测量值是从哪里取来的。这里有一个关于InfluxDB数据格式的例子，演示了每个数据都是如何被存储的。

Here are some of the TSDBs supported by Grafana:

这里有一些Grafana支持的TSDB：

* [Graphite](https://graphiteapp.org/)
* [InfluxDB](https://www.influxdata.com/products/influxdb-overview/)
* [Prometheus](https://prometheus.io/)

weather,location=us-midwest temperature=82 1465839830100400200

| -------------------- -------------- |

| | | |

| | | |

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

|measurement|,tag\_set| |field\_set| |timestamp|

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

Collecting time series data

收集时间序列数据

Now that we have a place to store our time series, how do we actually gather the measurements? To collect time series data, you’d typically install a *collector* on the device, machine, or instance you want to monitor. Some collectors are made with a specific database in mind, and some support different output destinations.

现在我们有了一个地方来存储我们的时间序列，那么实际上我们怎么来收集这些测量值呢？我们需要安装一个专业的采集端，安装在你的设备，机器或者任何你想监控的实例上，去收集时间序列数据。一些采集端需要考虑到是需要使用特定的数据库，而另一些可能支持不同的输出端。

Here are some examples of collectors:

这里有一些采集端：

* [collectd](https://collectd.org/)
* [statsd](https://github.com/statsd/statsd)
* [Prometheus exporters](https://prometheus.io/docs/instrumenting/exporters/)
* [Telegraf](https://github.com/influxdata/telegraf)

A collector either *pushes* data to a database or lets the database *pull* the data from it. Both methods come with their own set of pros and cons:

采集端要么可以把数据推送到数据库，要么可以让数据库从采集端拉取数据。这两种方法各有优缺点：

|  | Pros 优点 | Cons 缺点 |
| --- | --- | --- |
| Push推送 | Easier to replicate data to multiple destinations. 更容易复制数据到不同的输出端。 | The TSDB has no control over how much data gets sent. 这个TSDB不能控制发送多少数据。 |
| Pull拉取 | Better control of how much data that gets ingested, and its authenticity. 能更好的控制拉取多少数据和它的可靠性。 | Firewalls, VPNs or load balancers can make it hard to access the agents. 防火墙，VPN或者负载平衡都会使得访问终端变得很困难。 |

Since it would be inefficient to write every measurement to the database, collectors pre-aggregate the data and write to the time series database at regular intervals.

由于把每个测量值都写入数据库太过于低效，采集端预先聚合好数据，并定期写入数据库是一个好的选择。