Why do we need a time-series database

2020141461124

刘进益

Before we answer the question "Why do we need a time-series database", we are supposed to know something about the data it stores - time-series data to understand this problem better.

Time-series data is a sequence of data points collected over time intervals, giving us the ability to track changes over time. Time-series data can track changes over milliseconds, days, or even years.

This detailed data doesn’t just include time as a metric, but as a primary component that helps to analyze our data and derive meaningful insights. Regardless of the scenario use case, all time-series datasets have 3 things in common:

1. The data that arrives is almost always recorded as a new entry
2. The data typically arrives in time order
3. Time is a primary axis (time-intervals can be either regular or irregular)

So we are supposed to know that time-series data workloads are generally "append-only".

But how is time-series data different ?

There is one interesting thing in time-series database is that it often performs `insert` instead of `update`, because we can only analyze the changes in state over time if we insert a new reading each time. This practice of recording each and every change to the system as a new, different row is what makes time-series data so powerful. It allows us to measure and analyze change: what has changed in the past, what is changing in the present, and what can we forecast changes may look like in the future.

We can give a brief definition of time-series data: a collection of values that represents how a system/process/behavior changes over time.

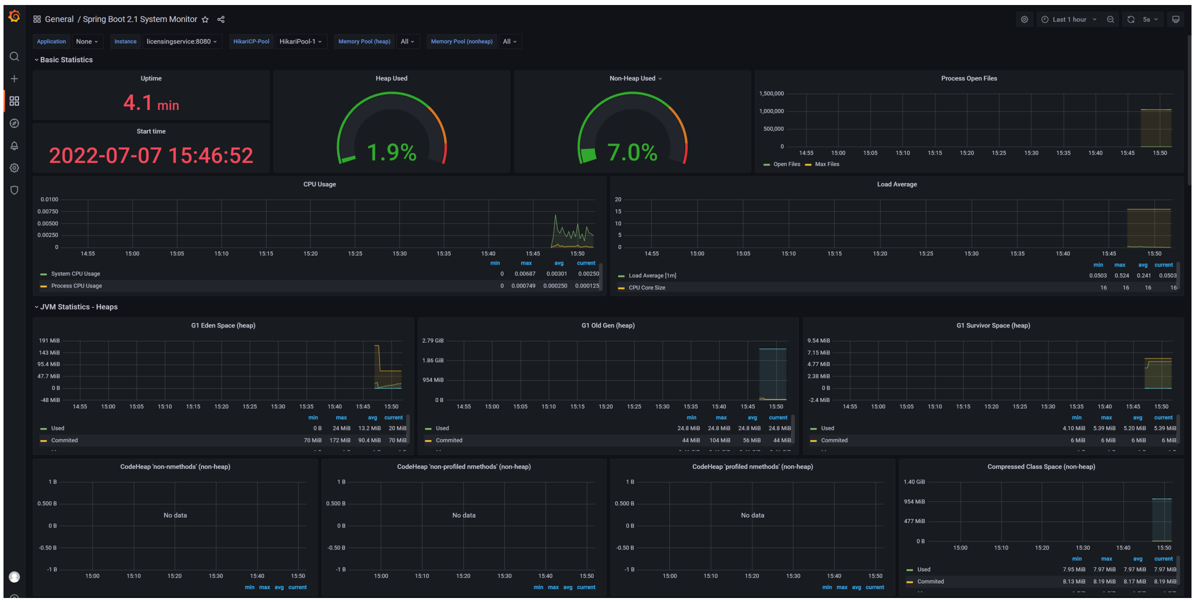
This is more than just an academic distinction. By centering our definition around “change,” we can identify time-series datasets that we aren’t collecting today and identify opportunities to start collecting that data now, so that we can harness its value later.

Let's take a web application for example and image you are an administrator. Every time a user logs in, you may just update a “last_login” timestamp for that user in a single row in your “users” table. But, what if you treated each login as a separate event, and collected them over time? With that kind of time-series data, you could analyze historical login activity, see how usage is (in-/de-)creasing over time, bucket users by how often they access the app, and more.

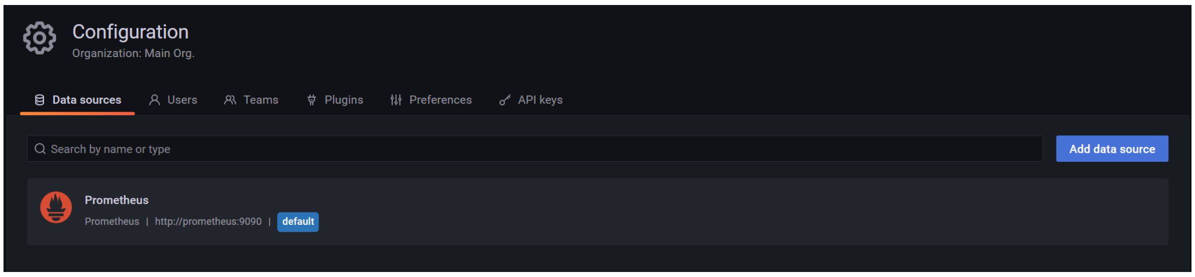
As we discussed, until now we’ve only tracked the last time a user logged in as a field in the “users” table and always update the previously stored value with the new login information. While this allows us to query how many people have logged in over a week or a month, we’re unable to analyze how often they log in, for how long, or drill into any other aspects that might tell us more about our users’ experience or their usage patterns.

We can quickly improve upon this by tracking information about every login, not just the most recent one. To do this, we'll start logging the timestamp of each login and the type of device used to access our application (e.g., phone, tablet, desktop). This small change - tracking just one more property about the user login experience - provides immediate value, allowing us to answer questions like, "what kind of devices are most frequently used (by individual users and across all users)?" and "what time of day are users the most active?". From there, we can better inform the features we prioritize - such as mobile-specific capabilities , the times we display certain promotional messages, and beyond

Another obvious example that we students can touch is the data of operational metrics. When we use Spring Boot, we can find there is one dependency called "spring-boot-starter-actuator". Spring Boot's 'Actuator' dependency is used to monitor and manage the Spring web application. To be honest, perhaps this is the closest data I can reach when it comes to time-series data, because these days I am studying distributed system and microservices, I also enabled the monitor system in my application. And it mainly uses "Prometheus" as its monitor system and time-series database. I can monitor the whole system status like this:



And this is through `Prometheus` as its data source:



> I also notice that TDengine is also compatible with Grafana, but I don't know if the data from Spring Boot Actuator can be pushed into TDengine in a simple way.

This kind of time-series metric data is crucial to keeping the services that we rely on, running without interruption. By tracking the changes in each metric, IT departments can quickly identify problems, plan for capacity increases during upcoming events, and diagnose if an application update resulted in changed user behavior, for better or worse.

These examples illustrate a key point: preserving the inherent time-series nature of our data allows us to preserve valuable information about how that data changes over time.

Of course, storing data at this resolution comes with an obvious problem: you end up with a lot of data, rather fast. So that's the catch: being able to analyze increased amounts of time-series data is more valuable than ever, but it piles up very quickly.

Having a lot of data creates a different set of problems, both when recording it and when trying to query it in a performant way, which is why people are turning to time-series databases in greater numbers than ever before. The world is demanding that we make better data-driven decisions, faster. The static snapshots found in traditional data won't cut it. To satisfy the demand, we need to be collecting data at the highest fidelity possible – and that's what time-series data provides: the dynamic movie of what's happening across your system (whether it's your software, your physical power plant, your game, or customers inside your application).

Finally, we come to the question "Why do we need a time-series database ?".

This question actually implies another question "Why can't we just use a normal database ?".

There are at least two reasons why TSDBs are popular: scale and usability.

Scale: Time-series data accumulates very quickly, and normal databases are not designed to handle that scale (at least not in an automated way). Traditionally, relational databases fare poorly with very large datasets, while NoSQL databases are better at scale . In contrast, time-series databases - whether they're relational or NoSQL-based - introduce efficiencies that are only possible when you treat time as a first-class citizen. These efficiencies allow them to offer massive scale, from performance improvements, including higher ingest rates and faster queries at scale (although some support more queries than others) to better data compression.

Usability: TSDBs also typically include built-in functions and operations common to time-series data analysis, such as data retention policies, continuous queries, flexible time aggregations, etc. Even if you're just starting to collect this type of data and scale is not a concern at the moment, these features can still provide a better user experience and make data analysis tasks easier. Having built-in functions and features to analyze trends readily available at the data-layer often leads you to discover opportunities you didn't know existed, no matter how big or small your dataset

So there are several benefits of a time series database :

1. More accurate and meaningful time series measurement
   A time series database makes it easy to measure how datasets change over time. You can concurrently view past, present, and future datasets for reporting that is more accurate and meaningful.
2. Resource-efficient data storage
   By the very nature of the data type, processing it can require massive amounts of storage, which can be difficult to manage. It's also very expensive. Time series databases have tooling to aggregate data into predetermined time periods and to eliminate any data streams as needed. There are also compression algorithms that optimize data storage.
3. Lightning-fast data queries
   A TSDB can also make it easy to query and retrieve data based on specific periods. Imagine that you can't remember the title of a book you recently read, but you know it was three months ago. Time series databases can help you figure out what the book was without having to use a bunch of wildcard searches.

Among many time-series database, I would like to introduce one of the best - TDengine to you.

TDengine is a highly efficient platform to store, query, and analyze time-series data. It is specially designed and optimized for IoT, Internet of Vehicles, Industrial IoT, IT Infrastructure and Application Monitoring, etc.

It has following advantages:

1. Scalable: TDengine provides out-of-box scalability and high-availability through its native distributed design. Nodes can be added through simple configuration to achieve greater data processing power. (This is one of the reasons I mention above)
2. High Performance: TDengine outperforms other time series databases in data ingestion and querying while significantly reducing storage cost and compute costs, with an innovatively designed and purpose-built storage engine.
3. SQL Support: TDengine uses SQL as the query language, thereby reducing learning and migration costs, while adding SQL extensions to handle time-series data better, and supporting convenient and flexible schemaless data ingestion.