Gnocchi

Gnocchi

Gnocchi is the project name of a TDBaaS (Time Series Database as a Service) project started under the Ceilometer program umbrella.

NOTE: The information below is historical and reflect the state of the project during its origin. For up to date information, please visit the official documentation [1]

Motivation

From the beginning of the Ceilometer project, a large part of the goal was to store time series data that were collected. In the early stages of the project, it wasn't really clear what and how these time series were going to be handled, manipulated and queried, so the data model used by Ceilometer was very flexible. That ended up being really powerful and handy, but the resulting performance has been terrible, to a point where storing a large amount of metrics on several weeks is really hard to achieve without having the data storage backend collapsing.

Having such a flexible data model and query system is great, but in the end users are doing the same request over and over and the use cases that need to be addressed are a subset of that data model. On the other hand, some queries and use cases are not solved by the current data model, either because they are not easy to be expressed or because they are just too damn slow to run.

Lately, during the Icehouse Design Summit in Hong-Kong, developers and users showed interest in having Ceilometer doing metric data aggregation, in order to keep data in a more long running fashion. No work has been done during the Icehouse cycle on that, probably due to the lack of manpower around the idea, even if the idea and motivation was validated by the core team back then.

Considering the amount of data and metrics Ceilometer generates and has to store, a new strategy and a rethinking of the problem was needed, so Gnocchi is a try on that.

Rethinking the problem

Ceilometer is nowadays trying to achieve two different things:

• Store metrics, that is a list of (timestamp, value) for a given entity, this entity being anything from the temperature in your datacenter to the CPU usage of a VM.

• Store events, that is a list of things that happens in your OpenStack installation: an API request has been received, a VM has been started, an image has been uploaded, a server fell of the roof, whatever

These two things are both very useful for all the use cases Ceilometer tries to achieve. Metrics are useful for monitoring, billing and alarming, where events are useful to do audit, performance analysis, debugging, etc.

However, while the event collection of Ceilometer is pretty solid and ok (but still needs to be working on), the metrics part suffers terrible design and performance issues.

Having the so called free form metadata associated with each metric generated by Ceilometer is the most problematic design we have. It stores a lot of redundant information that it is hard to query in a efficient manner. On the other hand, systems like RRD have existed for a while, storing a large amount of (aggregated) metrics without much problem. The metadata associated to these metrics being another issue.

So that left us with two different problem to solve: Store metrics and store information (the so called metadata) about resources.
**Prototype implementation (OUTDATED)**

jd has started a prototype of that solution with the Gnocchi project. It provides a time series storage and a resource indexer, which are both fast and scalable.

It provides a REST API. The REST API provides two types of resources:

- Entity, which are things you can measure.
- A resource, which has various information, and is linked to any number of entities.

Here's how the software is architected.

The two types of objects managed and provided by Gnocchi are stored in two different data stores, because these two types of things are definitely different. Relying on the same type of data storage would break things, just like what we saw in Ceilometer.

**Time series storage**

Like in Ceilometer the storage driver is abstracted so you can write your own using whatever technology you want.

The storage driver is in charge of storing the metric in an aggregated manner. Yes that means you aggregate according to what the user requested when they created the entity, before you store the metrics.

The canonical implementation of the time series storage is based on the use of Pandas\(^2\) and Swift. For the record, the first version was based on Whisper\(^3\), but it turned out to be a bad option (terrible base code which also has a lot of assumption about using file based storage). Using pandas and building our own serialization format is much better.

Swift provides an almost infinite of space to store data, is likely already part of your cloud (so it's not something else to deploy for your operators) and it is very scalable.
**Resource indexer**

Like for the TSD, the driver is abstracted.

The canonical implementation is based on SQLAlchemy, just because SQL is fast, indexable, etc. So it's a great storage for that. And SQL is already deployed in your OpenStack environment, so again: It's not yet another thing to deploy.

The plan is to describe some resources (instances, images, etc) so they have real schemas that can be indexed and queried efficiently.

**Current REST API**

The API is still evolving for now. We currently have this:

- **POST /v1/entity**: create an entity. You have to specify the list of archives you want to store. An archive is composed of a tuple (granularity, number of measures). It defines the number of measure you want to keep and how often you want them. For example, (5, 60) will store 60 measure with a granularity of 5 seconds.
- **DELETE /v1/entity**: delete an entity
- **POST /v1/entity/<name>/measures**: post a list of [timestamp: <ts>, value: <v>] to store as measurements
- **GET /v1/entity/<name>/measures**: get the list of measures for this entity. You can specify an interval with start= and stop=, and the type of aggregation you want to retrieve (mean, median, last, min, max, first…)
- **POST /v1/resources**: create a resource

And I'm still working on that for now.

**References**

Article Sources and Contributors


Image Sources, Licenses and Contributors


License

Attribution 3.0 Unported (CC BY 3.0)
http://creativecommons.org/licenses/by/3.0/