System Design — Design a Monitoring System

This post will discuss the details of designing a monitoring system, which is asked in many interviews.

How to Collect Metrics — Pull or Push

There are two models to collect data, push and pull. In monitoring system, I would always go with pull model, and the reason is as below:

1. Scalability Concern. Our infrastructure will keep growing, and we many have hundreds or thousands of services in the coming years. And our service usage, user base will grow too. If we go with the push model, then all these services will keep hitting our monitor service. If we have a service which processes 1M requests per second, and this service push the metrics to our monitoring service upon every request, then we will suffer from scalability issue frequently as we grow. So instead of getting called to get metrics, I would prefer to actively pull the data from the services.
2. Automatic Upness Monitoring — By pulling the data proactively, we can directly know if the service is alive or not. For example, if one service is not reachable, we can be aware of it immediately.
3. Easier Horizontal Monitoring — If we have two independent systems A and B, but one day we need to monitor some service in system B from system A. We can pull metrics from system B directly, no need to configure system B to push to system A.
4. Easier for Testing — We can simply spin up testing env, and copy the configuration from production, then you can pull the same metrics as prod and do testing.
5. Simpler High Availability — just spin up two servers with the same configuration to pull the same data to achieve HA.
6. Less configuration, no need to configure every service.

Base on the analysis above, my design for the pull model is below:

1. Our service will pull the data from the services regularly (for example every second). We need a real time monitoring system, but a lag of a couple of seconds is totally fine.

2. Exporters — The services should not call our monitor service to send the data. Instead, they can save the metrics to an exporter, and the data can be stored there to get pulled. So that, our monitor service will not be exhausted from getting called, and it will be more scalable. Also our monitoring system may need the data in a specific format, and the services may be designed in different technologies, and have data in different formats. So we require an exporter attached to each service, which reformats the data into the correct format for our monitor services. And our monitor will pull the data from the exporters.

3. Push Gateway — For cron jobs, they are not service based, but we may need to monitor the metrics from them too. So we can have a push gateway, which lives behind all the cron jobs, and the monitor can just pull the data from the gateway directly.

Exporter Design

Since we discussed the components for the Pull model, i.e., Exporter, and Push Gateway.

Some interview may question why not have multiple services hooked to one exporter. And I would always prefer one service per exporter, and the argument is below:

1. Operational bottleneck — the exporter will become a bottleneck if we have too many services behind it

2. Single point of failure, and one service pushes too much will block others

3. If I am only interested in the metrics of one service, I cannot get that only, I have to read all

4. No upness monitoring — if one service is not reachable, we will not be able to know.

5. Hard to get service metadata — we can store the service metadata in the exporter

Clustering?

Our monitoring system has to be very stable, so I would not go with the network clustering approach for the monitoring service. The reason is, clustering is very complicated, and easier to break. So it would be better to have on single solid node that does not depend on network.

Also, for the monitoring data, we usually care more about recent data. We usually do not care about metrics days or weeks ago. So we only need to store recent data instead of all historical data. Then there is no reason for us to go with the clustering approach.

And we can simply running 2 servers in parallel, which will be sufficient enough for HA.

DB Design

Since we only care about more recent data in the monitoring. The data usage pattern for monitor is like below:

1. recent data is very frequently accessed
2. historical data may be accessed occasionally

So we can store the recent data in memory for faster reads, and older data in disk. If we have 1M metrics to monitor, and for each metrics, there is a data point for every second, which is 16 bytes (key-value pair). Then for a server with 128GB memory, we can save around 2 hours of data. Which is good enough.

For the data in memory, we can save them in chunks, and once an older chunk is filled, we can simply compress it and save it on to a disk. For these data, querying on them will be slower, as we need to read from disk and decompress them. But I think slowness on querying old data is acceptable.

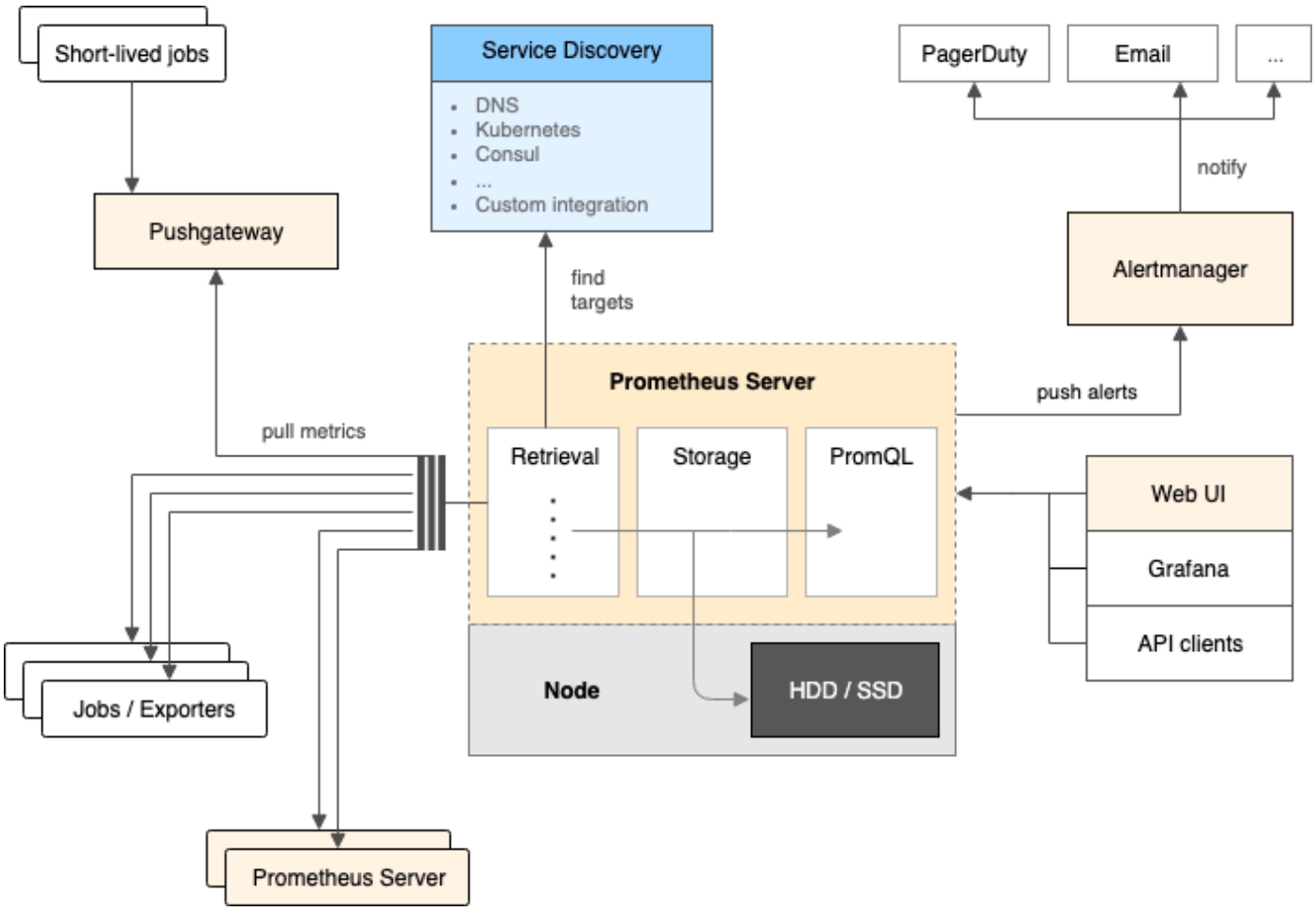
For much older data, like data months ago, we can store the compress data into a cheaper data storage offsite.

Since the recent monitored data are in memory, we will need a recovery system for them. If the server crashes, in order not to lose all the data, we need to create snapshots of the memory maybe every few minutes.

Also, we need to keep a monitor on the memory usage on the monitor service, in case our server is running out of memory during peak usages. When the memory usage is high, we may need to speed up the compress and save to disk process.

The DB we need to use for monitoring service would be time series DB.

High Level Design



Base on the discussion above, this is a high level design for a monitor service.

- Exporter — Pulls metrics from targets and convert them to correct format
- Push Gateway — Kron jobs to push metrics to at exit, then we can pull metrics from it
- Data retrival workers — pull data
- Time series storage — Local SSD / Remote Storage
- Query Service — visualize data
- Alert manager — to send alerts to different channels
- Service Discovery — Configuration for the targets to pull metrics from