A new `Security Infrastructure` PaaS platform

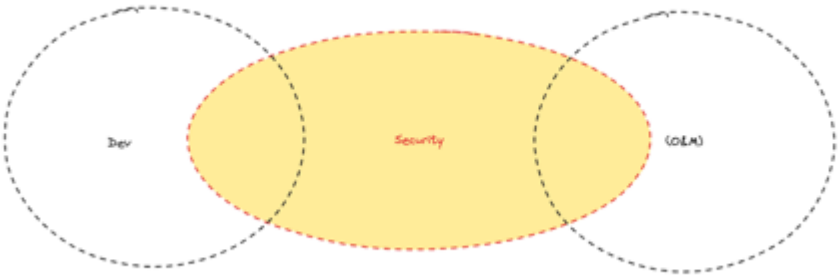
Abstract

As more and more security products are developed, We realize that much of the underlying infrastructure or code for these security products can be reused. That's where this proposal comes in. This proposal contains the following sub-proposals

1. Architecture proposal
2. Milestone
    1. Resolve existing product iterations and labor conflicts

Perspectives

Most users of Internet security products have ambiguous needs and cannot distinguish between APM and Security. But from our product development perspective, we need to re-categorize and define security product metadata.



Metadata

If we revisit the software engineering lifecycle, it's not hard to deduce what metadata we need.

Dev Stage

In the product development phase.

- We need to review the code for security to rule out the possibility of sensitive information in the code or configuration files.
- We need to check if the third-party library may have CVEs
- We need to check if the underlying middleware, e.g. `Nginx`, `Tomcat`, contains CVEs.
- ....

O&M

in the go-live phase of a new product feature:

- We need to record the `operation and maintenance personnel` operations.

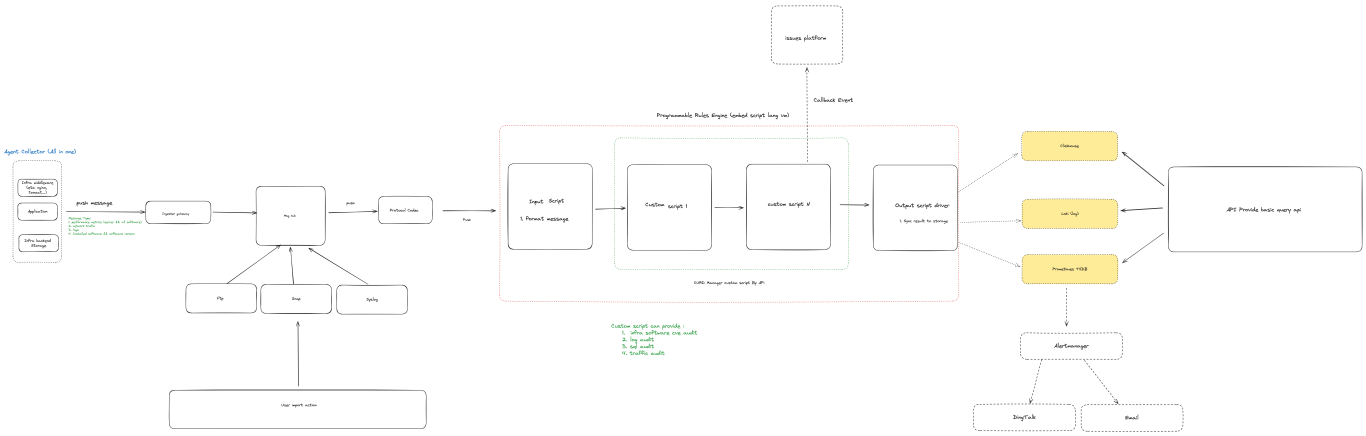
After the product launch

- we need to count the program running information, to ensure that the platform is stable. In the view of the customer platform stability is also part of the security.
- We need to analyze program logs, network traffic, to examine threats from the Internet, or some program-initiated intrusions, cross-privilege operations and so on.

Based on the above description, we summarize that the metadata we want are roughly in the following broad categories:

1. Logs
2. Network Traffic (pcap)
3. APM (Application Performance Management) metrics

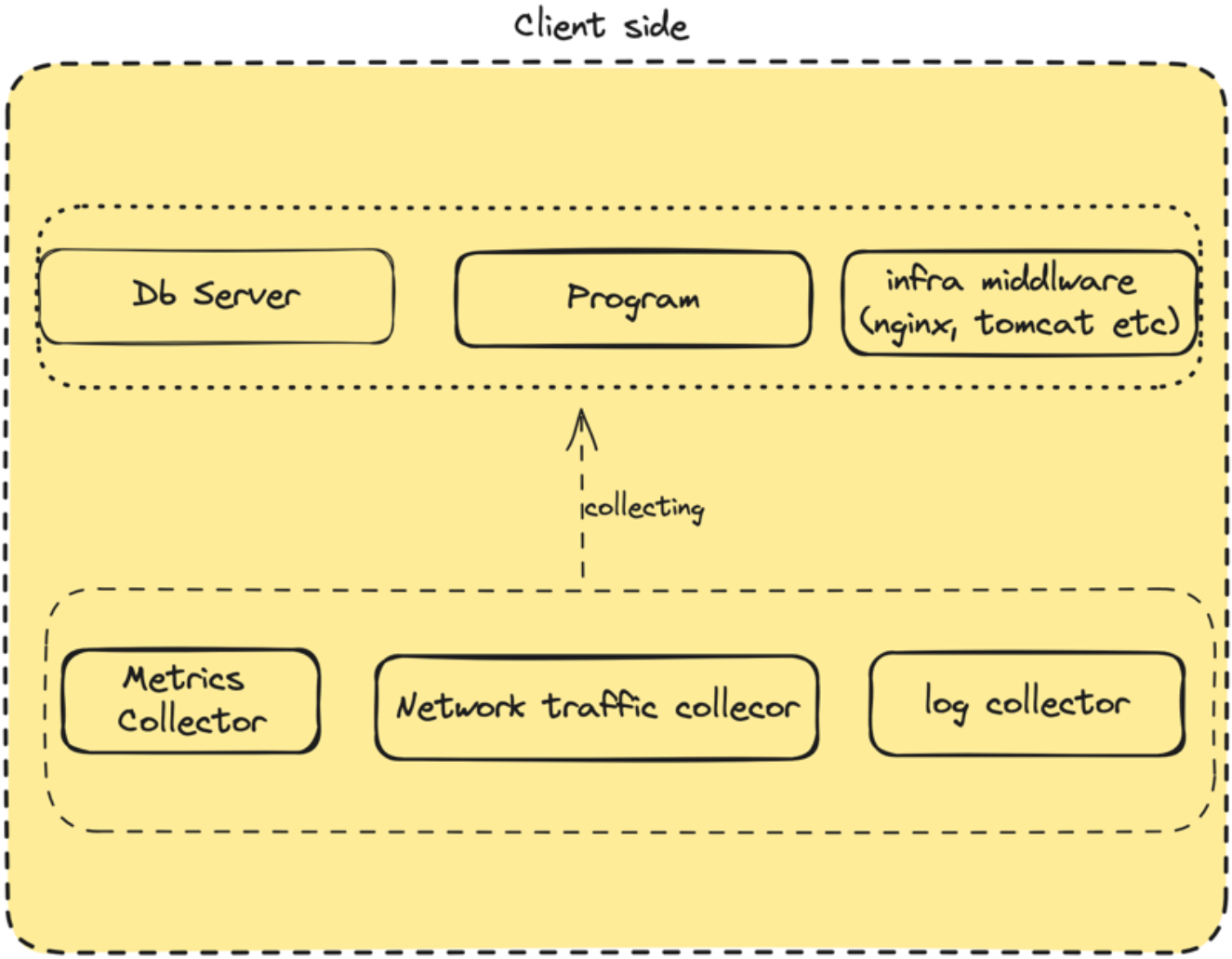
Architecture proposal



Above we can break it down into two pieces:

1. Client Side (customer's server)
2. Cloud Server Side (Provided by us)

Client Side



> All collectors are integrated in one binary file

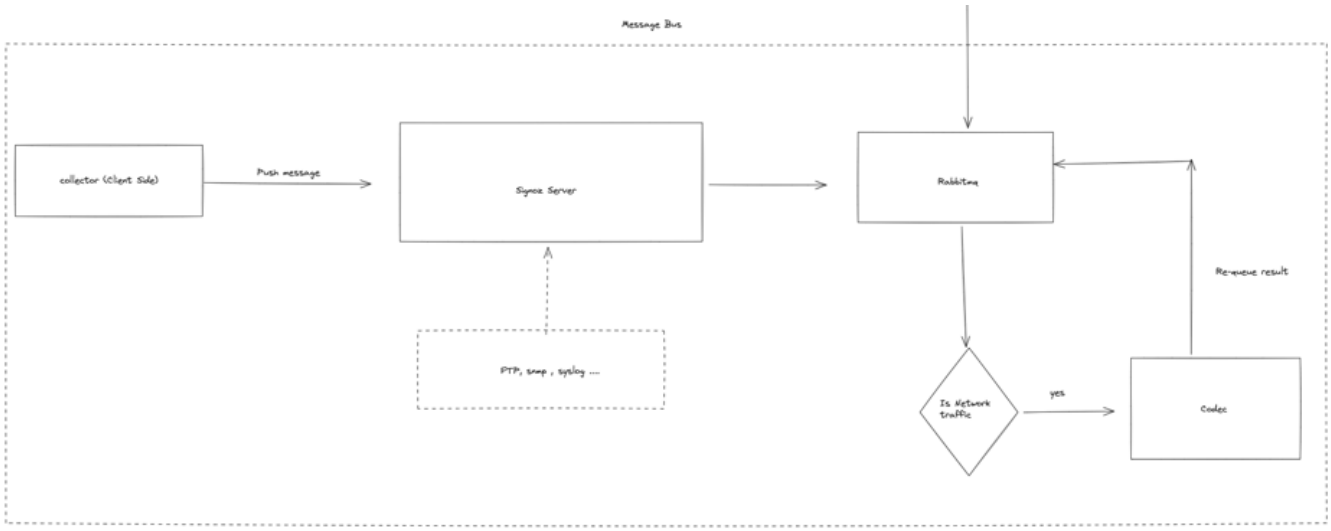
On the client side, we're primarily responsible for collecting metadata and doing light preprocessing.

Cloud Server

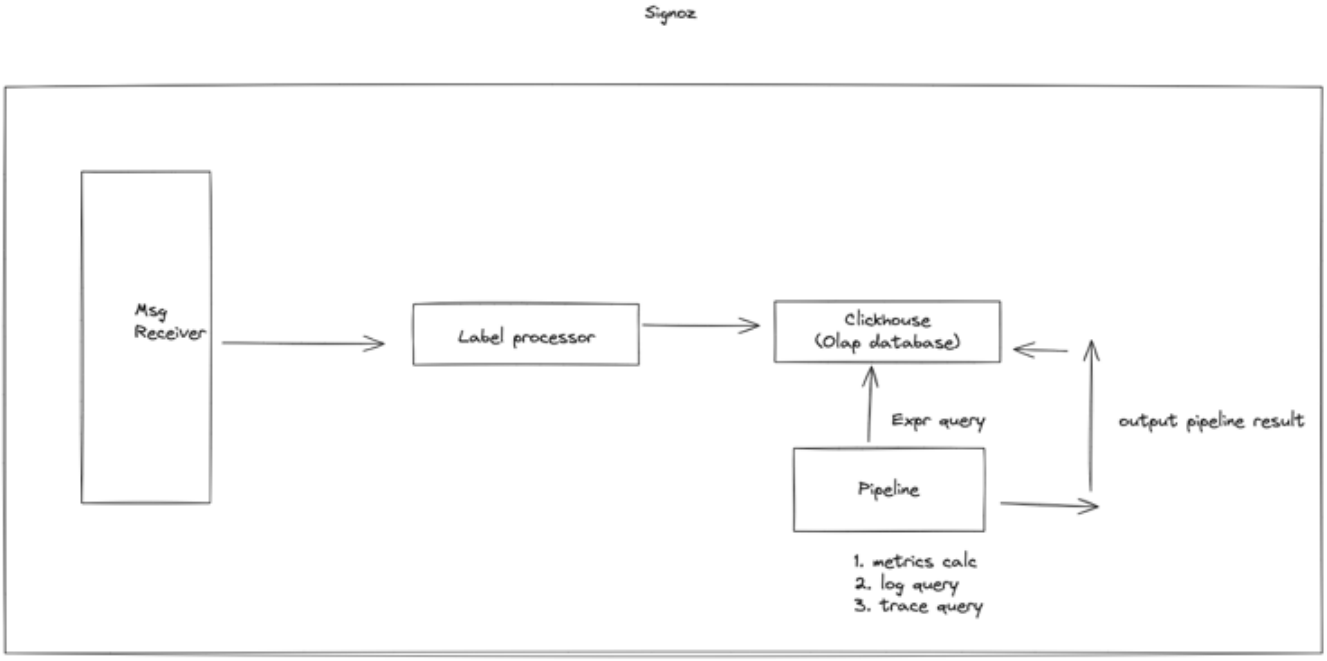
On cloud server side, We made the following technical selections:

1. Signoz
2. Rabbitmq (message bus queue)

Message Bus



Signoz part

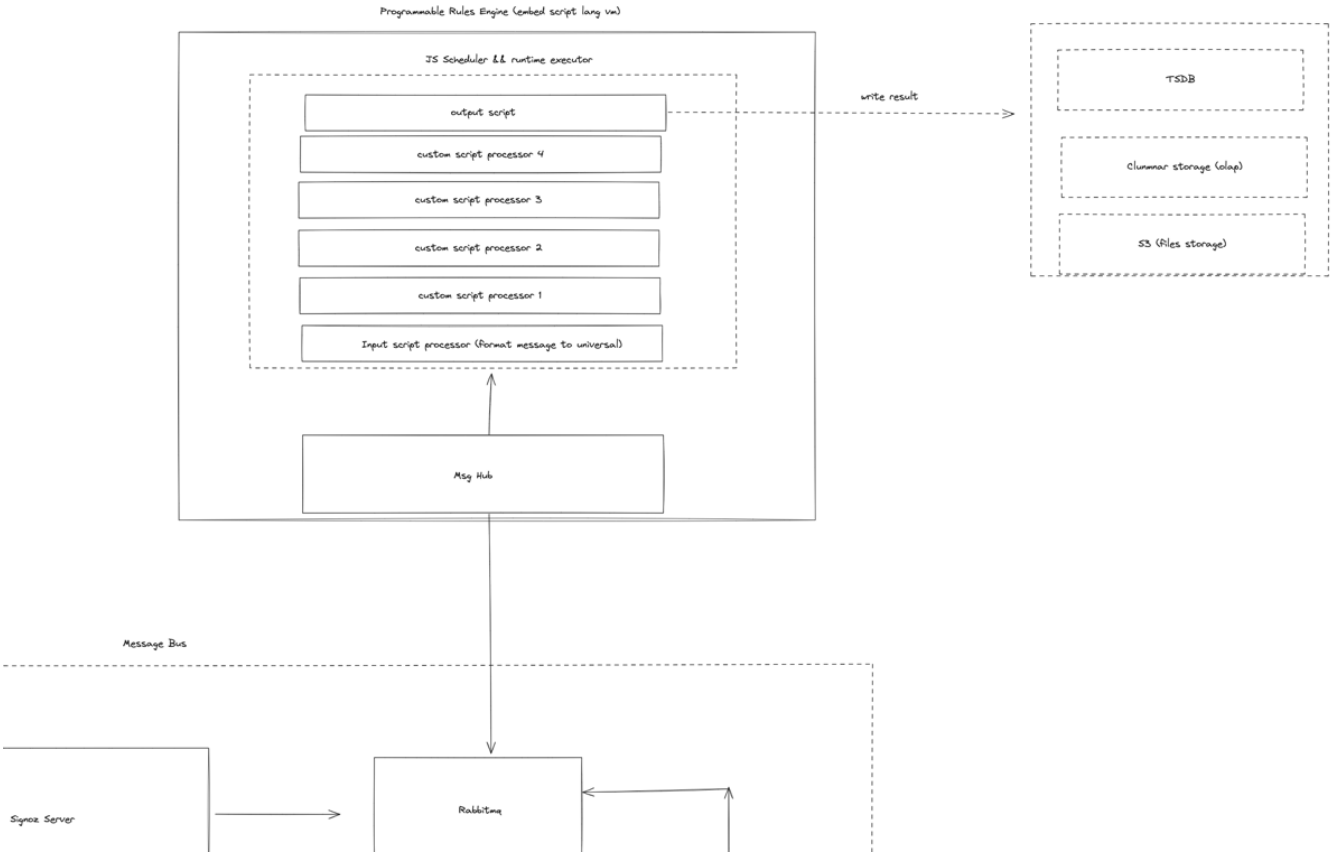


Signoz fulfills 80% of our development needs, but there are a few problems:

1. The data falls into the database first, `pipeline` look up from the library and tag it and then put it into the database, which can be fatal to the performance of columnar data.
2. Storage driver and `clickhouse` strong binding.
3. pipeline feature in-flexibility.

So we need to hack the signoz code. but the current state of affairs is sufficient for `log audit`.

Server Side



Why need `Programmable Rules Engine`

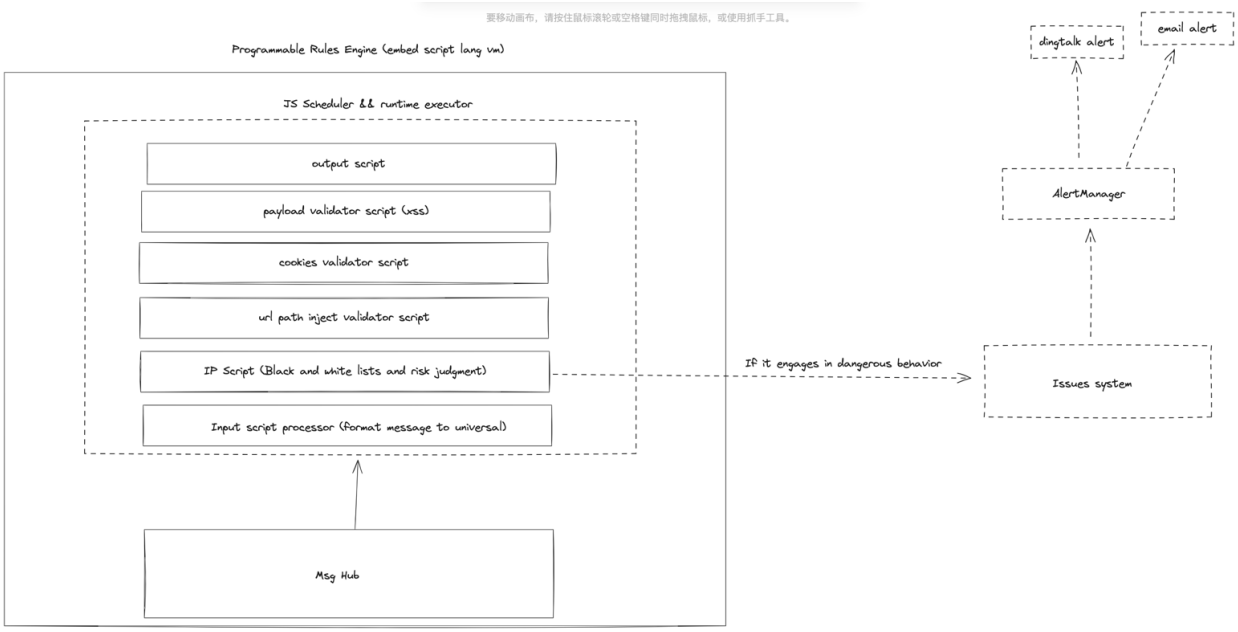
As we've discussed before, we want to make a product that will provide a foundational capability for all security product development.

Imagine we're developing a "Log Audit", we're collecting `Nginx` logs, we need to audit them.

```
192.168.70.137 - - [02/Apr/2024:03:27:39 -0400] "GET /api/v2/info/security
HTTP/1.1" 200 159 "http://192.168.10.228:8080/" "Mozilla/5.0 (Macintosh; Intel Mac
OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0
Safari/537.36" "-"
```

1. Signoz Formatting Log and labeling it.
2. Re-queue rabbitmq.
3. Msg-hub received logs

4. Rule engine will do:



The rules engine is intended for business development, who can program custom scripts to reorganize data to meet their business requirements.

Of course, the above process also applies to `database audit system`, `data audit system`.

Milestone

Now that we've discussed the technical implementation, let's take a look at the milestones. In terms of milestones, in addition to the manpower issue, I believe that the current team's ability to attack the technology

- stage 1 : We're going to introduce signoz in the `log audit` to fulfill the stage-1 milestone.
- stage 2 :
  - Extend signoz to unbind him from the clickhouse.
  - Ruler engine developing
- stage 3:
  - We introduced the functionality provided by this architecture as the underlying infrastructure in other systems.

Conclusion

For this new infrastructure PaaS system, (we) By completing one of the sub-system, thus gradually improving the whole system.

The whole program is practical, not just a mouthful.