**关于服务器设计技术选型调研报告**

**关键词**

**保密，隔离，安全，限制，扩展**

核心需求：

1. 保密性：

核心服务要加密或者是不直接提供本地开发版本。

2. 隔离性：

客户编写服务必须有一定容忍和限制语法，还有一些库上的限制。

3. 安全性：

客户编写的一些脚本要检查，要又隔离性。

4. 扩展性：

流程，模型易于修改

**设计思路参考**

Dapr，firebase，faas

**开源库选型：**

考虑到加密性质，要使用至少是二级制部署，分为本地云部署。

所以底层技术栈选型golang，这样可以利用go的性能，云特性，分布式特性。

而且可以打包成一个二进制文件，具有一定的保密性质。

**核心库：**

**Go-meta**（正在开发）主要是将model和数据库关联，这部分正在抄sandman2, postgres-meta.

**Prest**（rest转换sql）这个库可以直接抄，不能抄也可以集成进来, 反正都是go系

**Goher-lua**（提供lua支持）这个库可以对lua有一个非常安全的沙盒。之后会详细介绍这个库。

**Gotrue**(提供鉴权)， 也可以用别的，这块我手里的库和设计方案超级多。

Realtime/Centrifugo提供对系统pg之类的事件支持，这个我手里方案超级多，上不上云都有一堆技术选型。

**Storage-api**：这个库主要是s3的存储一个封装，这个我自己写一个就行了，我对这个太熟悉了。

**本地小型版本**，

本地版本只提供部分简单功能，但是大数据接口需要使用网上开源的api。

这里考虑可以对Go-meta，Prest， Goher-lua， Gotrue，Storage-api，还有自带的事件工具结合起来。变成一个库，或者提供一个docker景象，便于开发

**云版本**

利用go的pkg特性，任意功能可以放在sidercar和中台上面，可以参考dapr和firebase设计，中台提供model，查询能力，sidercar提供服务负载，sdk功能，同时为了安全，lua的脚本应该在sidercar里面，容器提供隔离能力。这样无论是客户使用的lua脚本出问题和客户的例如js等代码出问题，都不会影响到中台上。

**Goher-lua**

GopherLua可以创建一个非常干净的Lua解释器实例，不加载任何系统模块。然后由程序员自己提供的模块注册进去，给内嵌脚本提供一个安全的沙箱运行环境。

GopherLua除了可以满足基本的lua需要，还将Go语言特有的高级设计直接移植到lua环境中，使得内嵌的脚本也具备了一些高级的特性

可以使用context.WithTimeout对执行的lua脚本进行超时

可以使用context.WithCancel打断正在执行的lua脚本

多个lua解释器实例之间还可以通过channel共享数据

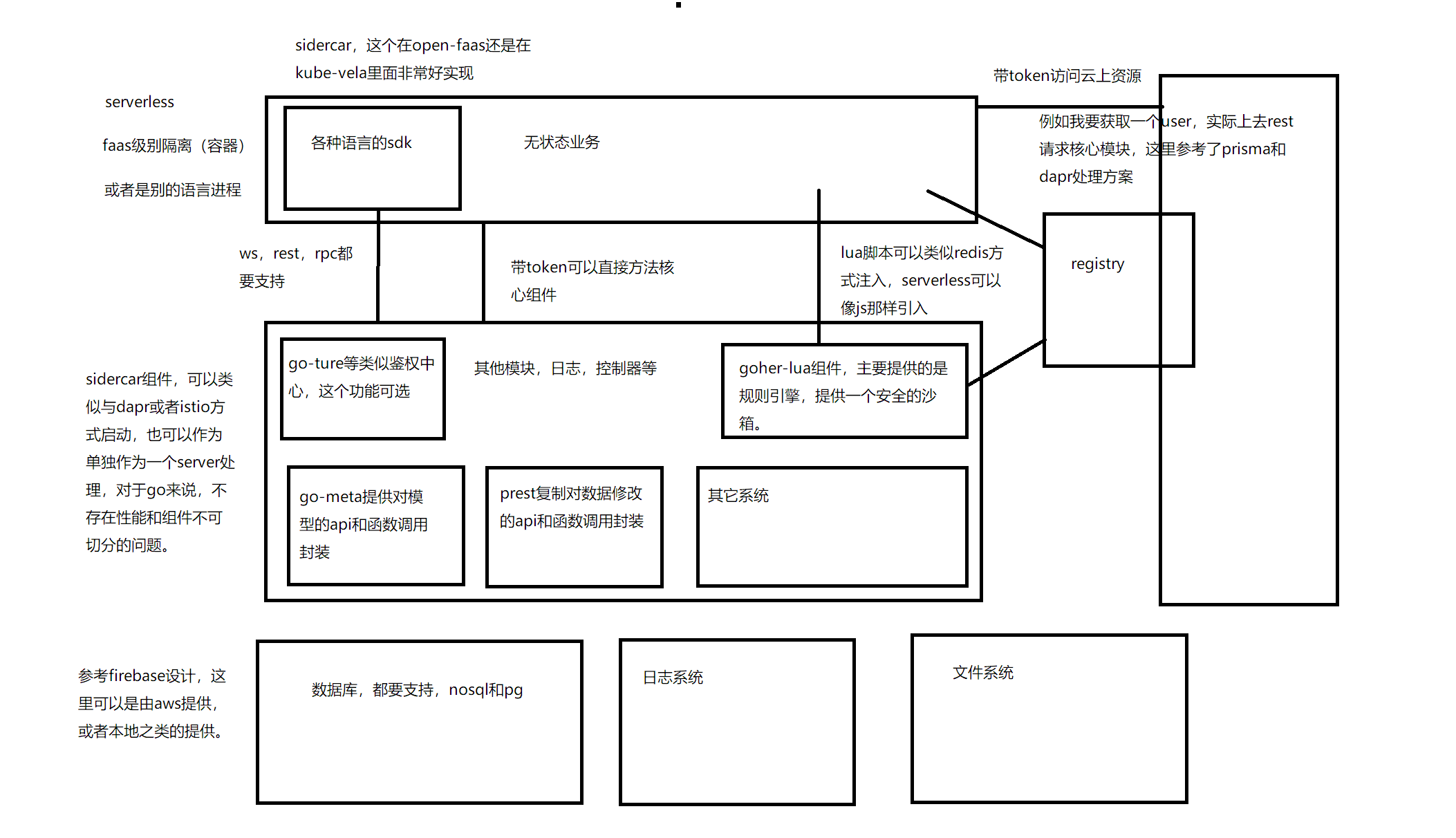
支持多路复用选择器select

<https://blog.csdn.net/codehole_/article/details/100892296>

https://www.136.la/jingpin/show-121333.html

**基础设计**

**核心**



利用go的包管理特性，以及打包后体积非常小特性，可以封装不同组件一下列举几种情况：

**1客户需要完全私有本地化：**

提供一个docker或者一个二进制的ctl工具用双进程方法启动，这时候这个docker或者ctl工具要提供大部分能力，且还能提供保密性质。

**2.客户需要本地半联网情况（非云环境）：**

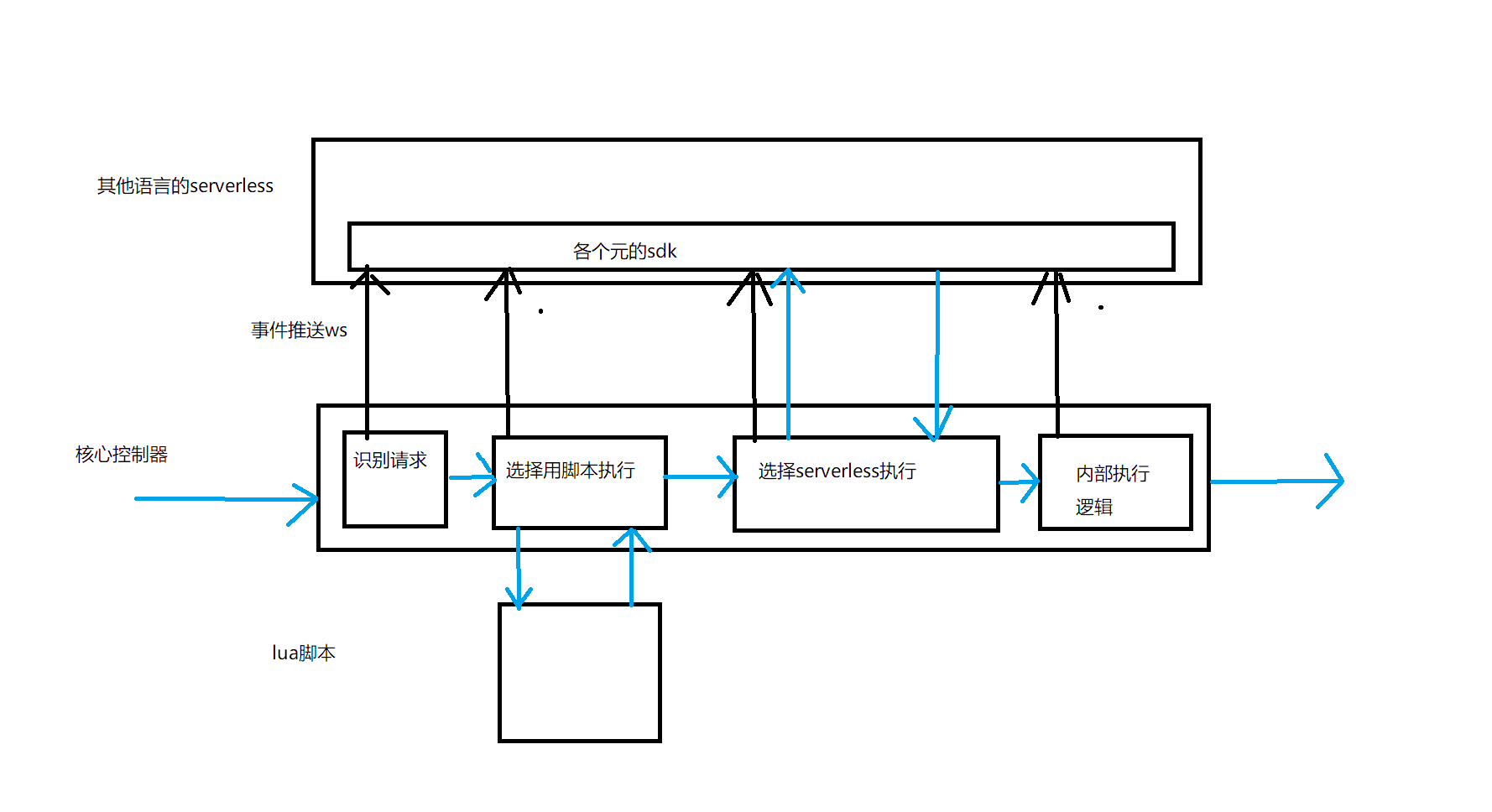
提供一个docker或者二进制的ctl工具双进程方法启动，这docker此时可能只提供registry功能或者本地存储功能或者部分本地业务（例如本地打印机，本地物联网之类的功能），可以和外部api有一定交互，或者客户直接使用外部api。

**3.完全云环境**

那么这个就随便玩了，之后有个serverless选型。

**4.完全云且本地冗余同步。**

本地和云数据有个同步逻辑，允许切换部分业务。



**流程控制器**

流程控制器可以用sidercar启动，或者一个pod启动，要看是不是要和中台放在一起，推荐是如果在云上，就放在sider里面，在本地成本比较小，就放到核心的docker上面

**Serverless和规则引擎**

**Serverless:**

安全性：容器化隔离

限制性：通过各语言的sdk封装

**规则引擎：**

安全性：goher-lua提供的安全vm

限制性：goher-lua提供lib引入

这里推荐

复杂操作和客户开发由serverless负载

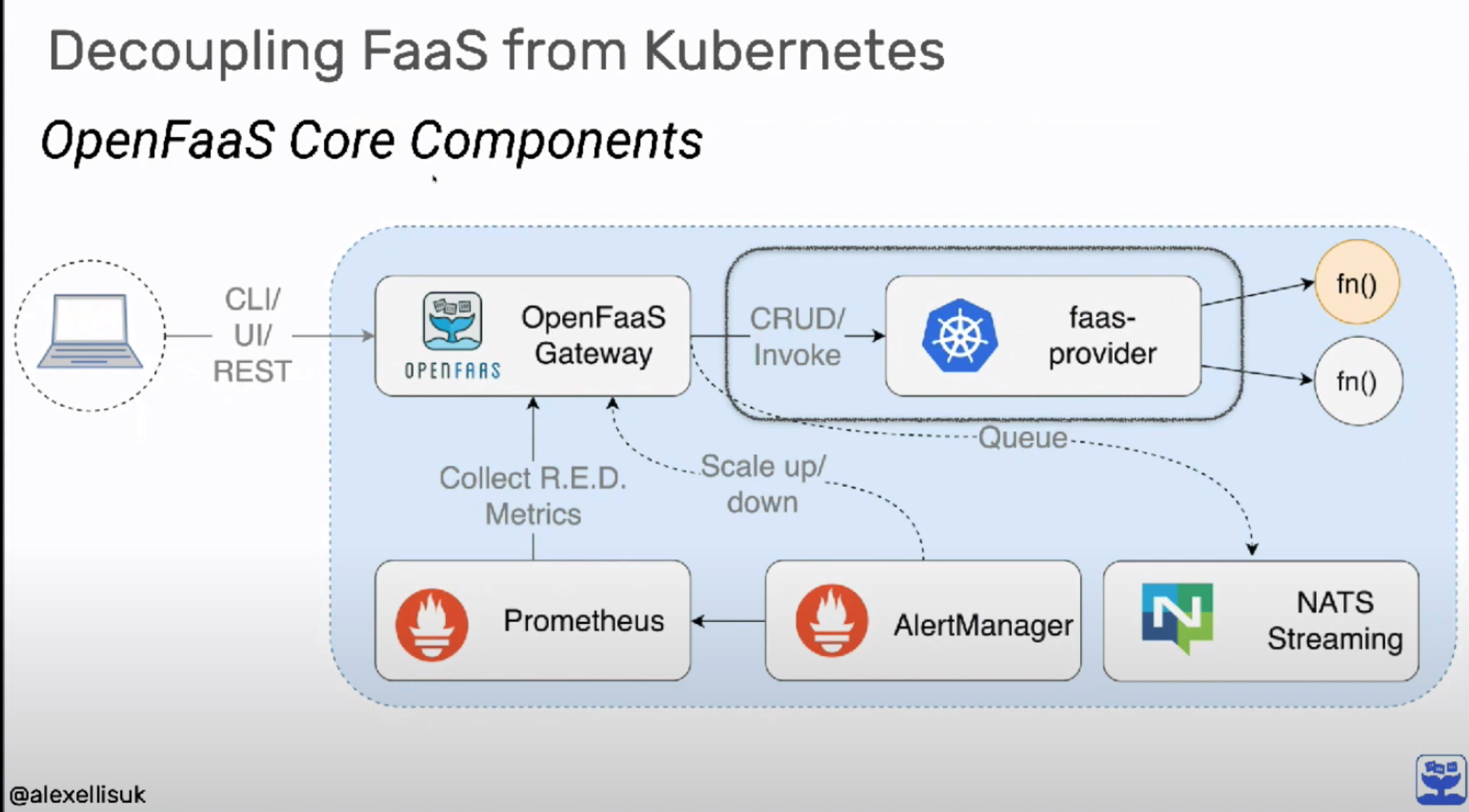
简单流程上的控制lua负责

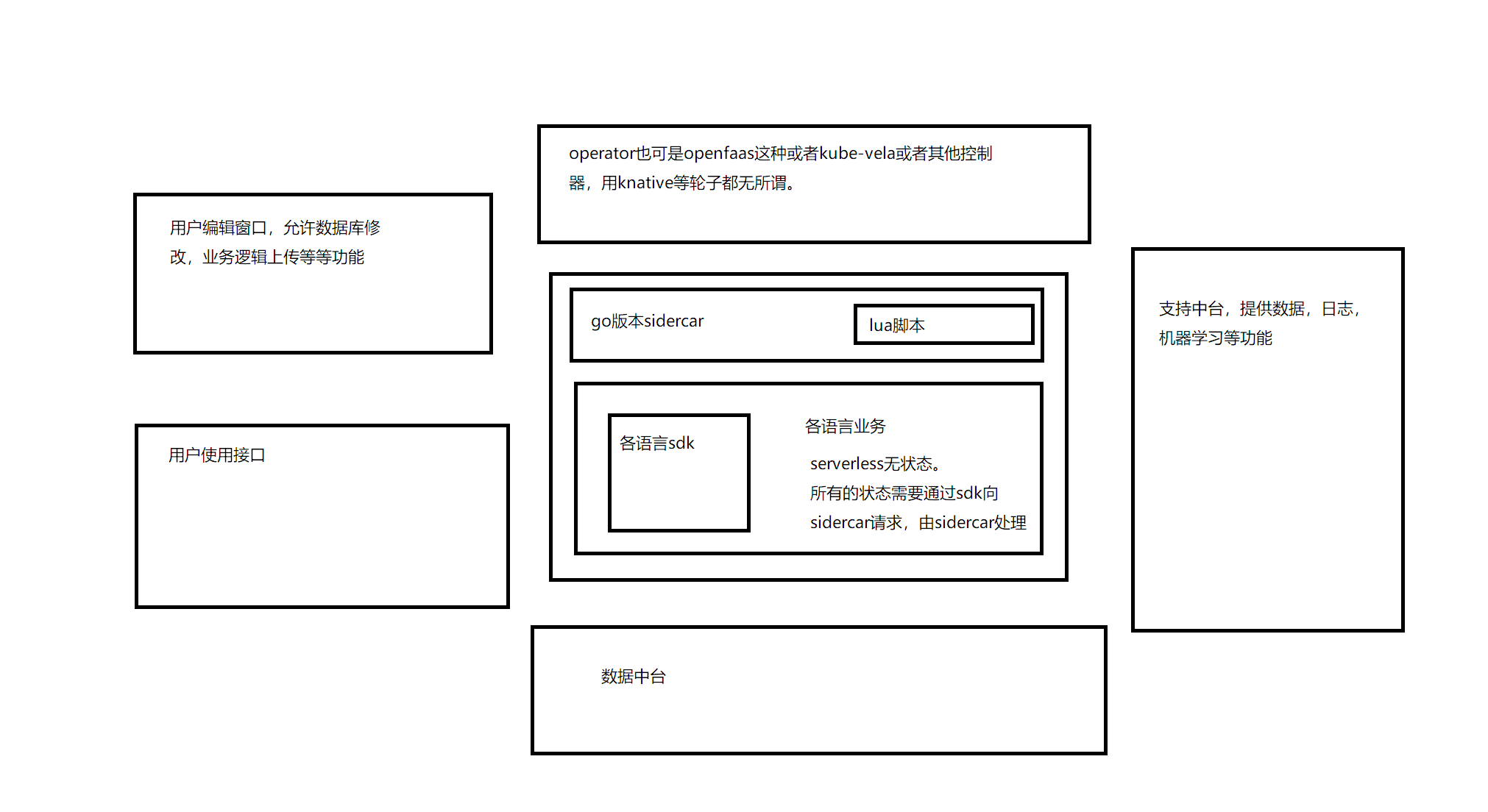
**云环境设计**

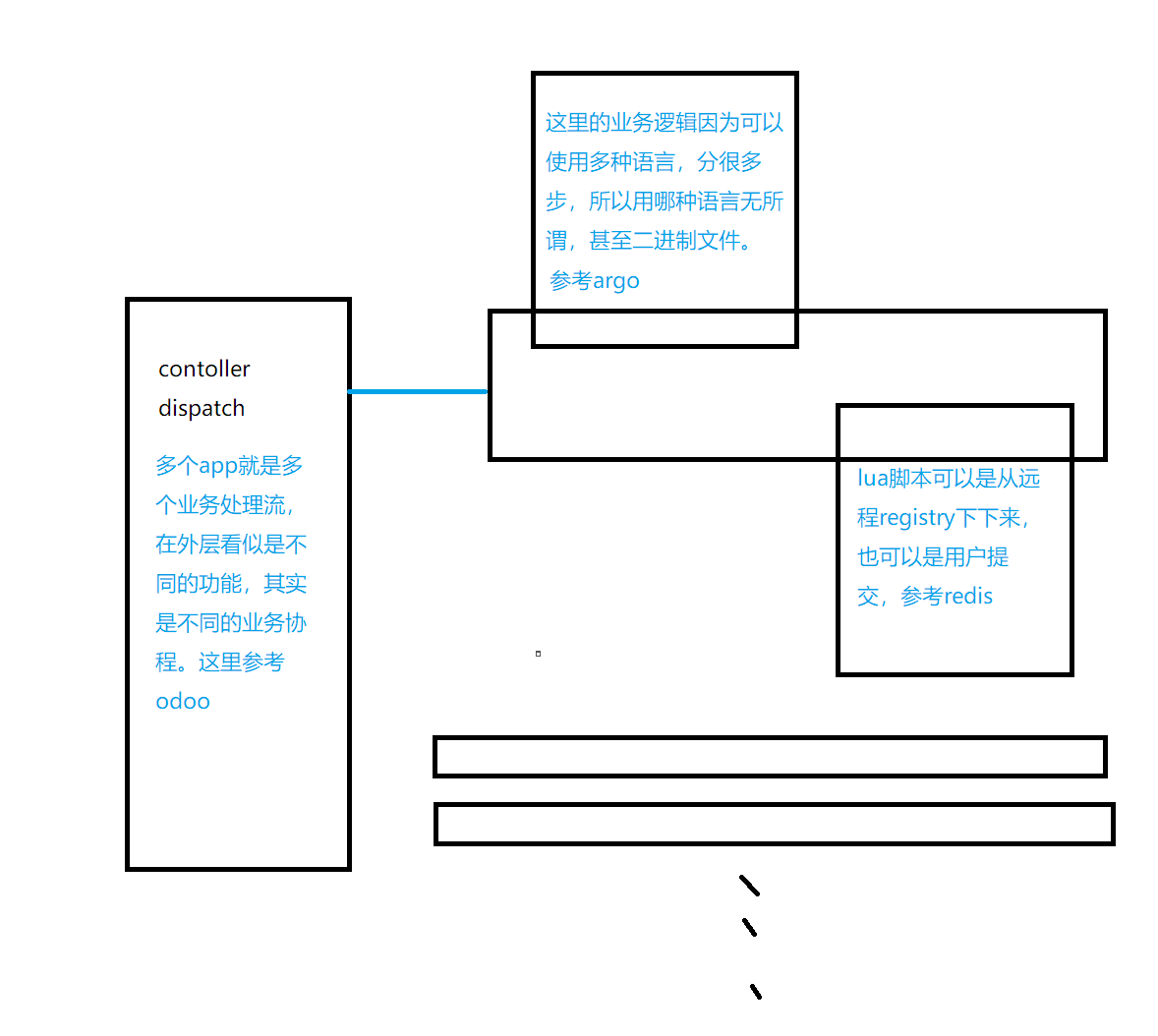
**整体部署参考openfaas，kube-vela，knative上层封装**

**单体部署设计也要参考operator-controller设计**

**其实基本随便玩，我可能会用faasd部署一个超轻量级的。**







**总结**

**核心点在于：**

**1提供一个serverless能力的进程**

**2 goher-lua提供一个安全的lua环境**

**3 go提供了一个二进制安全，又有性能的语言支持。**