Service Mesh 落地之后: 为 sidecar 注入灵魂

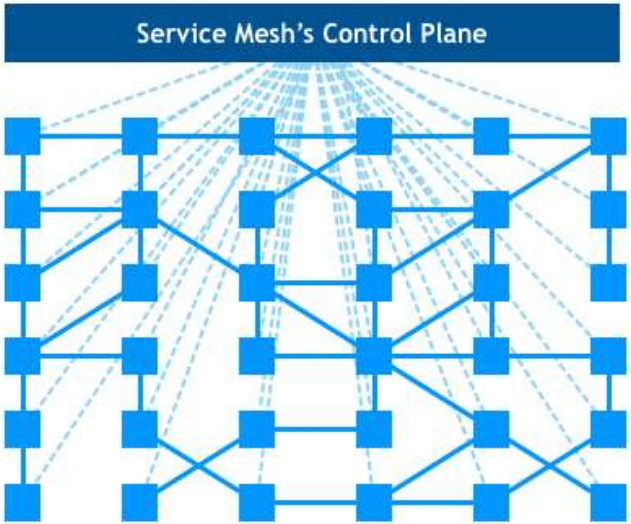
周群力

Co-founder of Layotto

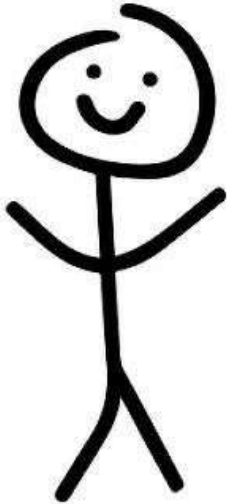
- Service Mesh 回顾
- Multi Runtime: 从 sidecar 到机甲
- Runtime API: 解决跨云部署和厂商绑定难题
- WebAssembly in sidecar: 让业务逻辑跑在sidecar里
- 展望2022：待解决的问题
- 总结

Service Mesh 回顾

由开发了 Linkerd 的Buoyant 公司提出



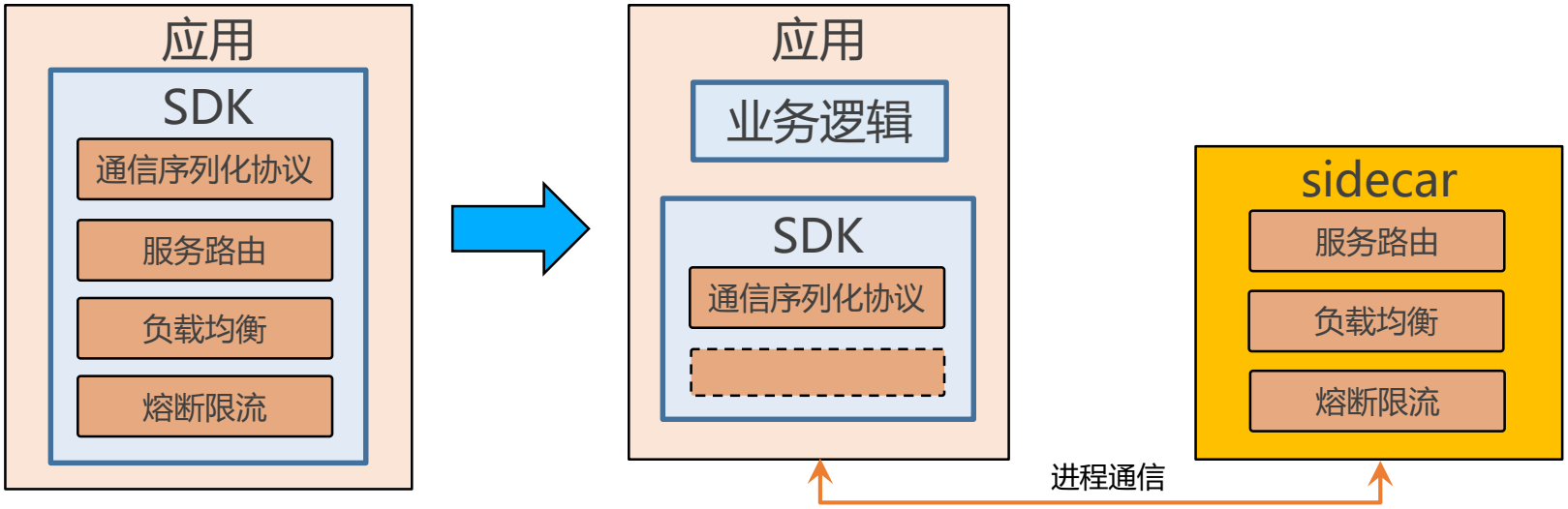
服务网格是一个基础设施层，用于处理服务间通信。云原生应用有着复杂的服务拓扑，服务网格负责在这些拓扑中实现请求的可靠传递。在实践中，服务网格通常实现为一组轻量级网络代理，他们与应用程序部署在一起，而对应用程序透明。



+

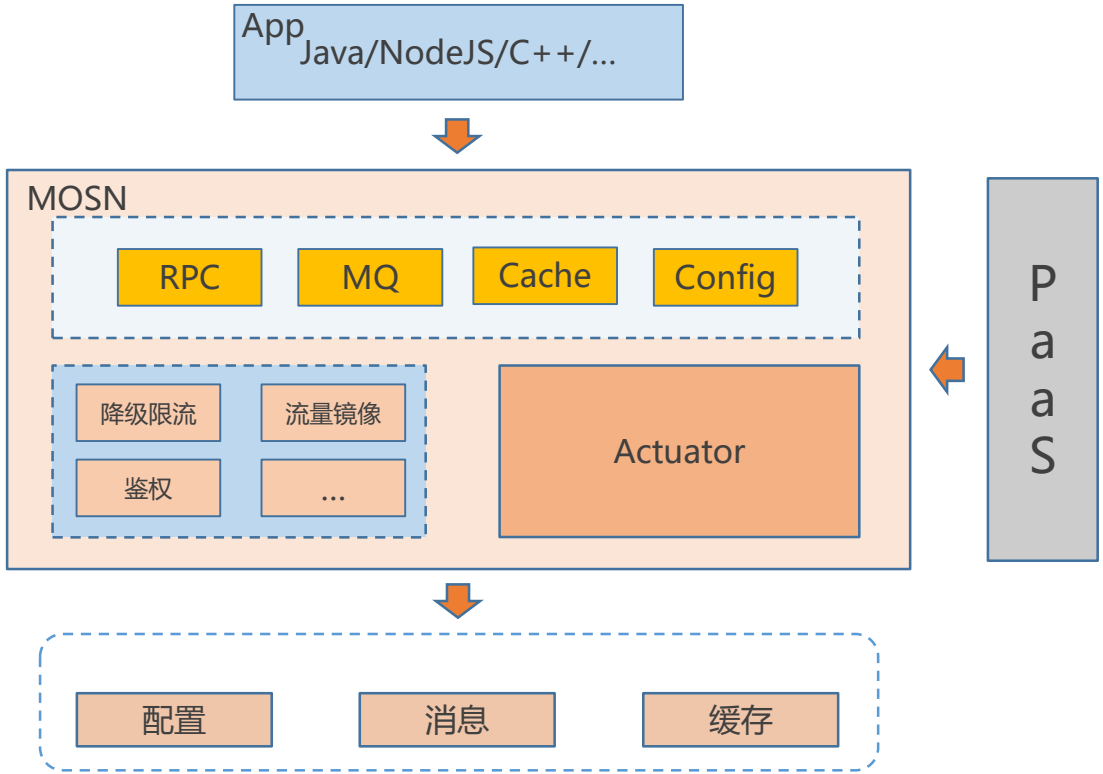
图片来源：
https://new.qq.com/omn/20190806/20190806A0SM4Q00.html

图片来源：
https://www.zhihu.com/question/55912398/answer/147967674

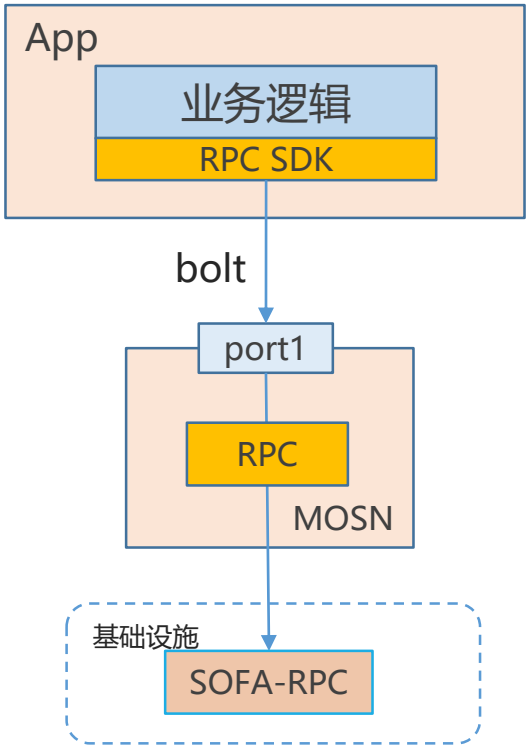


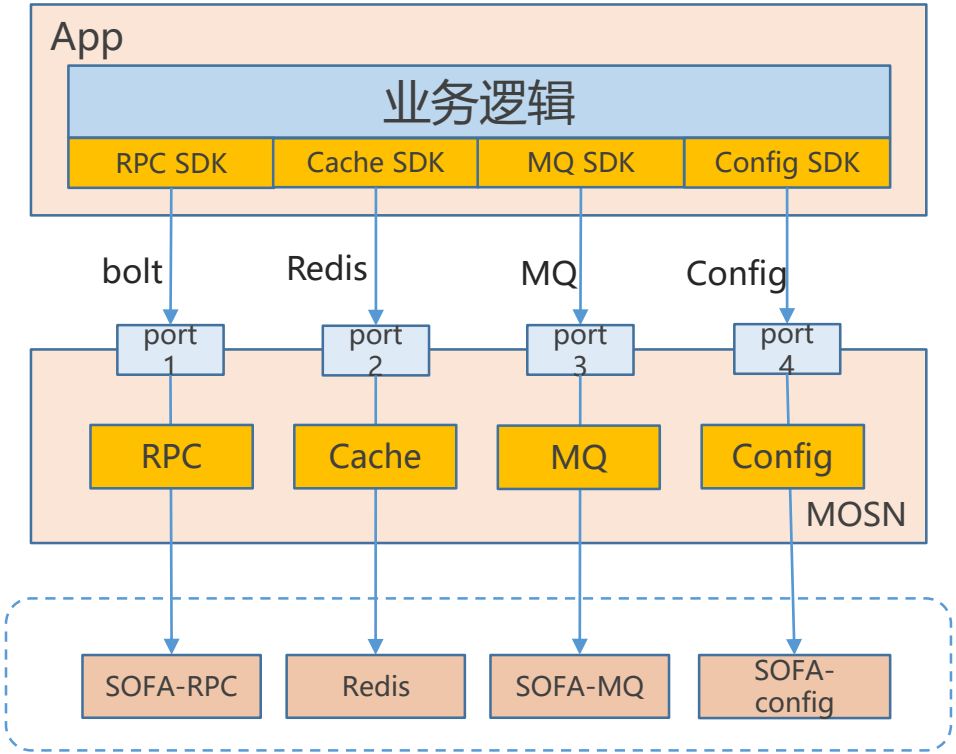
- 升级成本高
- SDK 版本不统一
- 异构语言治理能力弱

- 业务解耦
- 平滑升级
- 异构语言治理


App
Java/NodeJS/C++/...
MOSN
RPC
MQ
Cache
Config
降级限流
流量镜像
鉴权
...
Actuator
PaaS
配置
消息
缓存

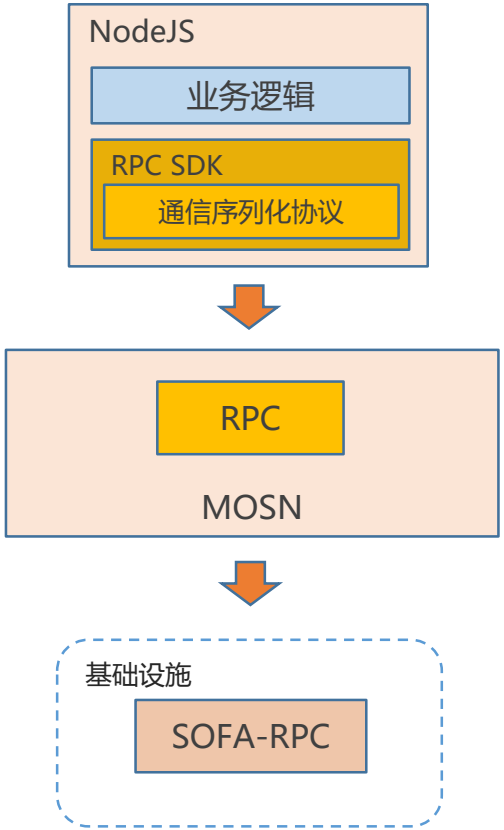
事情没有那么简单

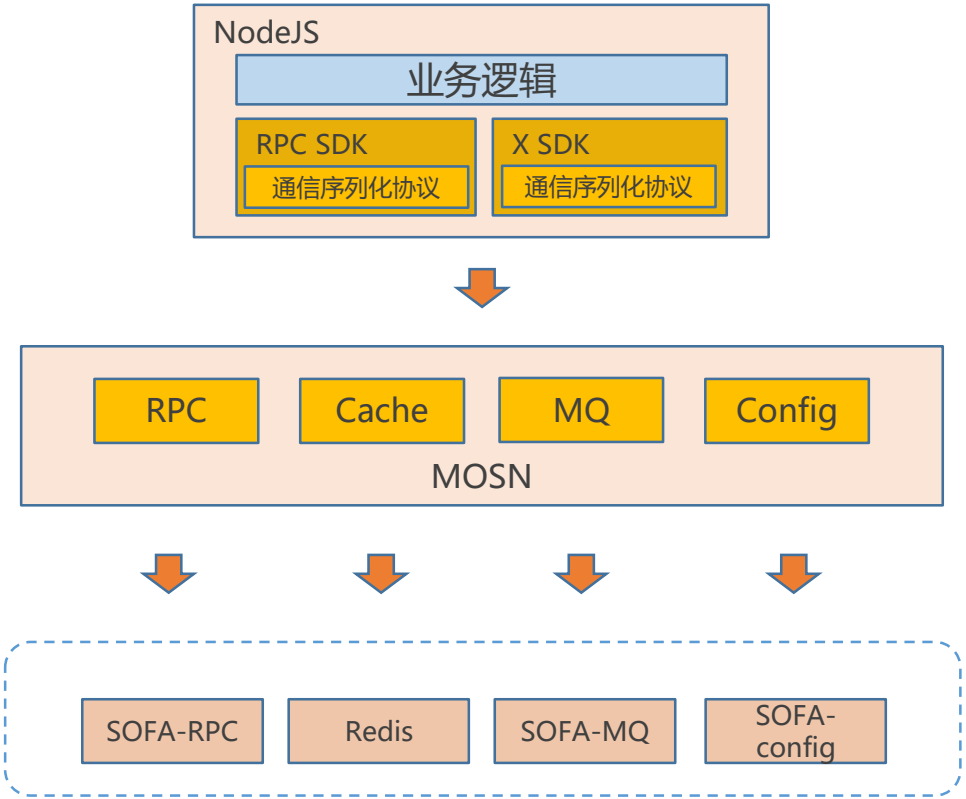

App
业务逻辑
RPC SDK
bolt
port1
RPC
MOSN
基础设施
SOFA-RPC

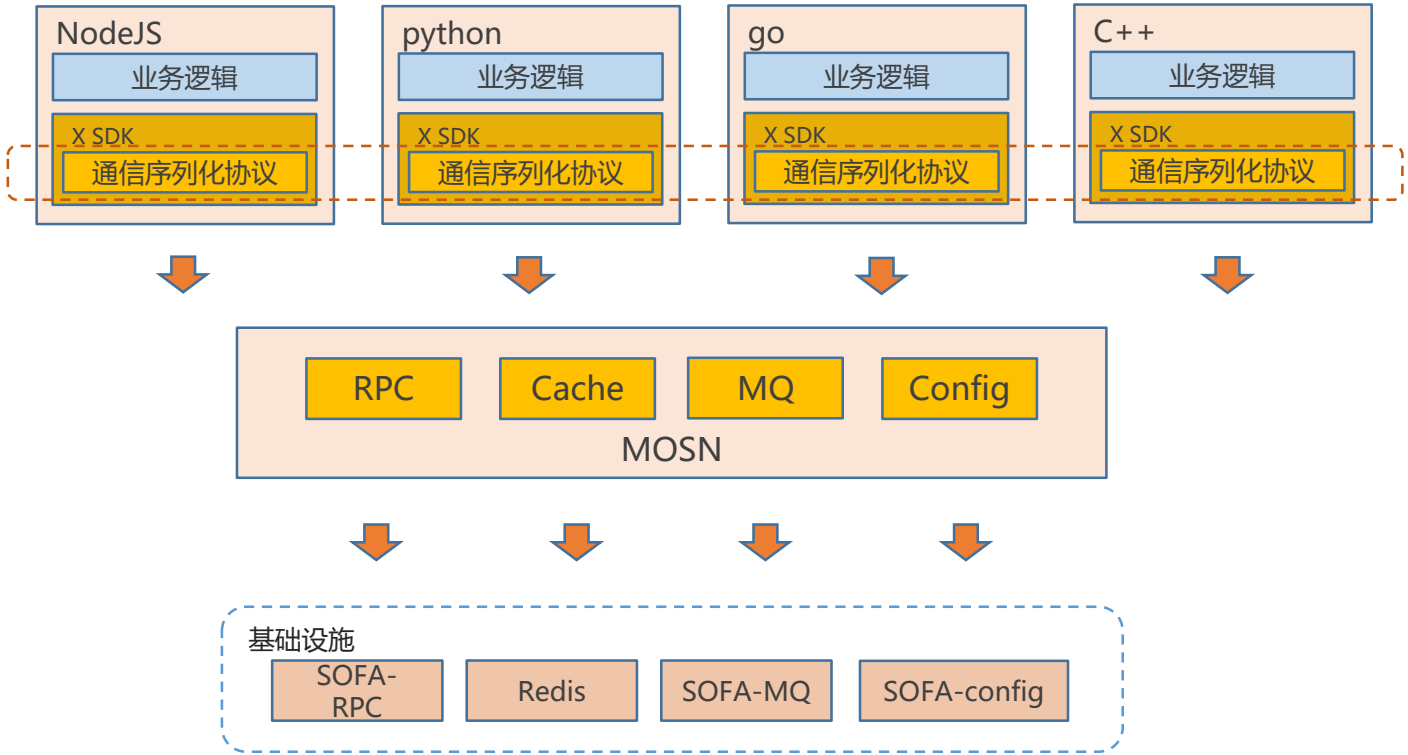


各种各样的SDK

基础设施强绑定


NodeJS
业务逻辑
RPC SDK
通信序列化协议
RPC
MOSN
基础设施
SOFA-RPC

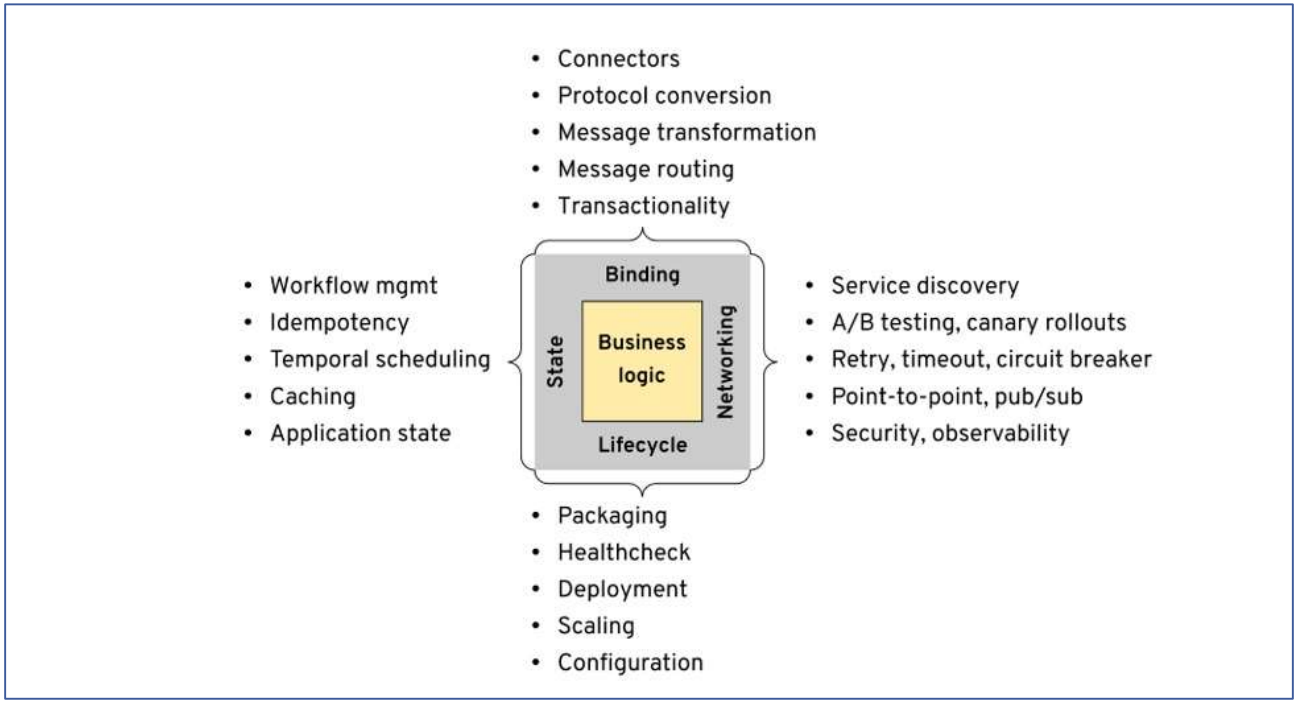

NodeJS
业务逻辑
RPC SDK
通信序列化协议
X SDK
通信序列化协议
RPC
Cache
MQ
Config
MOSN
SOFA-RPC
Redis
SOFA-MQ
SOFA-config


NodeJS
业务逻辑
X SDK
通信序列化协议
python
业务逻辑
X SDK
通信序列化协议
go
业务逻辑
X SDK
通信序列化协议
C++
业务逻辑
X SDK
通信序列化协议
RPC
Cache
MQ
Config
MOSN
基础设施
SOFA-RPC
Redis
SOFA-MQ
SOFA-config

Multi-Runtime

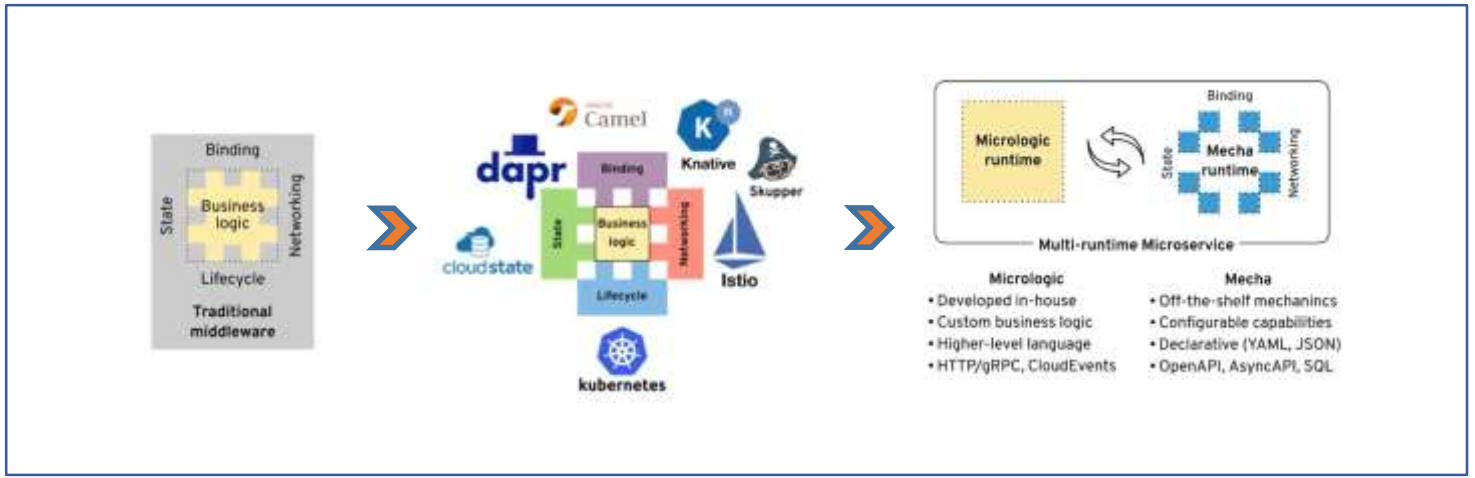
图片来源：
https://www.etsy.com/listing/648454769/avatar-aliens-amp-suit-robot-custom

什么是 Runtime?



Reference:https://www.infoq.com/articles/multi-runtime-microservice-architecture/

什么是 Runtime?



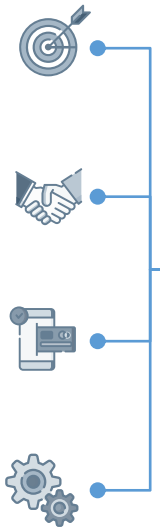
Reference: https://www.infoq.com/articles/multi-runtime-microservice-architecture/

定位
网络代理

交互方式
流量拦截

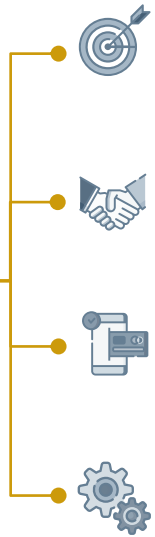
通信协议
私有协议

能力丰富度
低



Service Mesh

Runtime



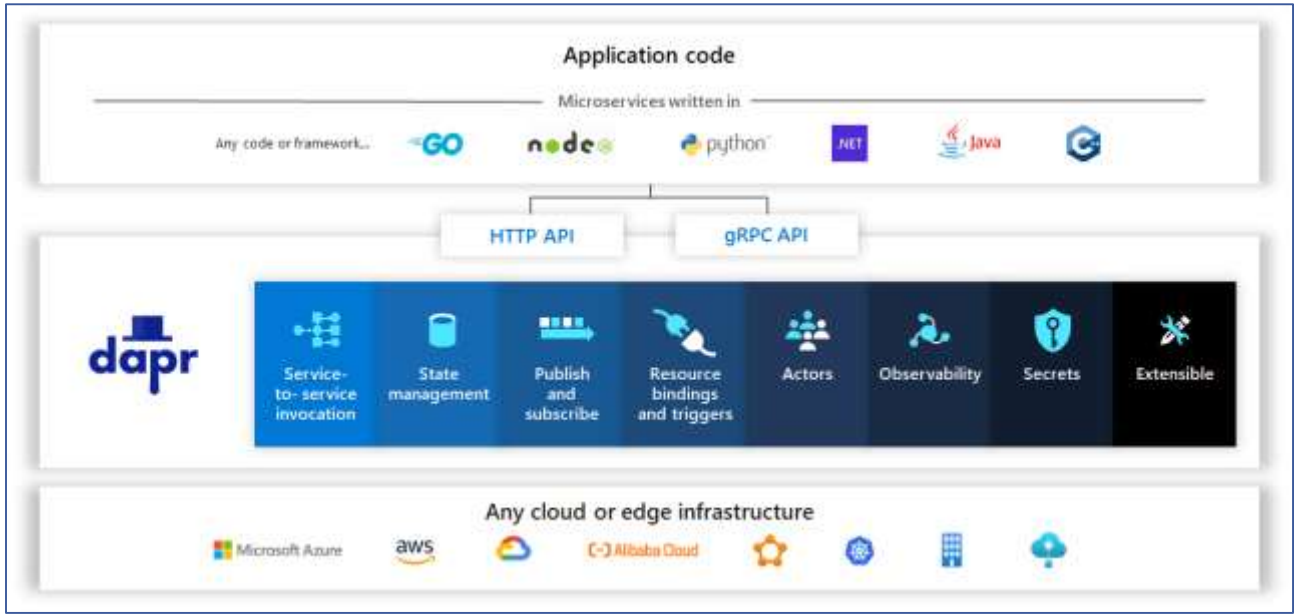
定位
分布式能力原语合集

交互方式
具有明确语义的 API

通信协议
Http/gRPC 标准协议

能力丰富度
高

Dapr

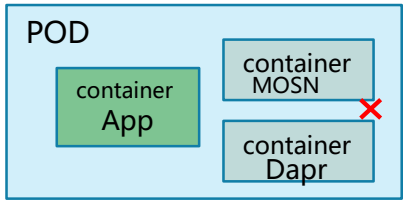


- 提供多种分布式能力
- 对接了丰富的基础组件
- 厂商解绑，跨云部署

事情没有那么简单

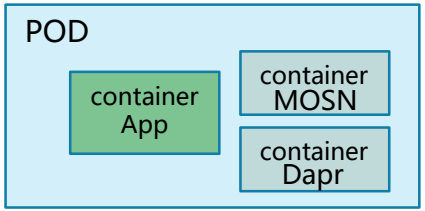
怎么落地

方案 1：替换



- 缺失 Service Mesh 能力
- 稳定性有待验证

方案 2：共存

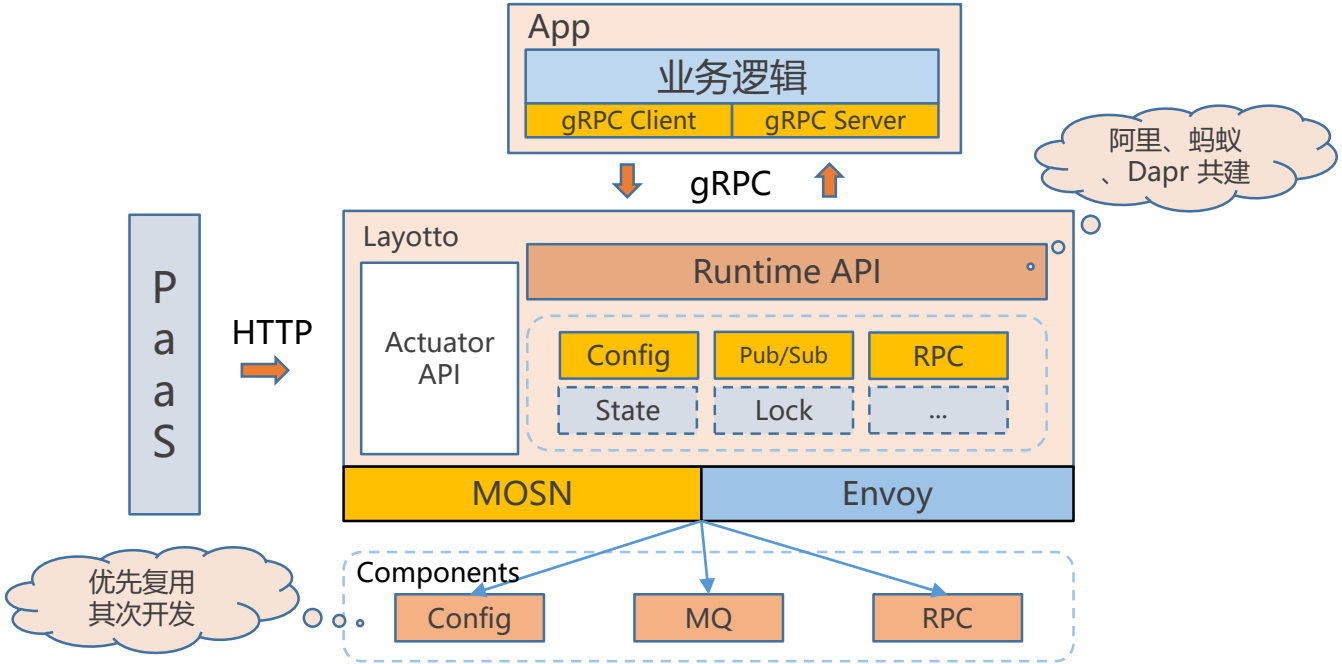


- 运维成本飙升
- 稳定性更难保证

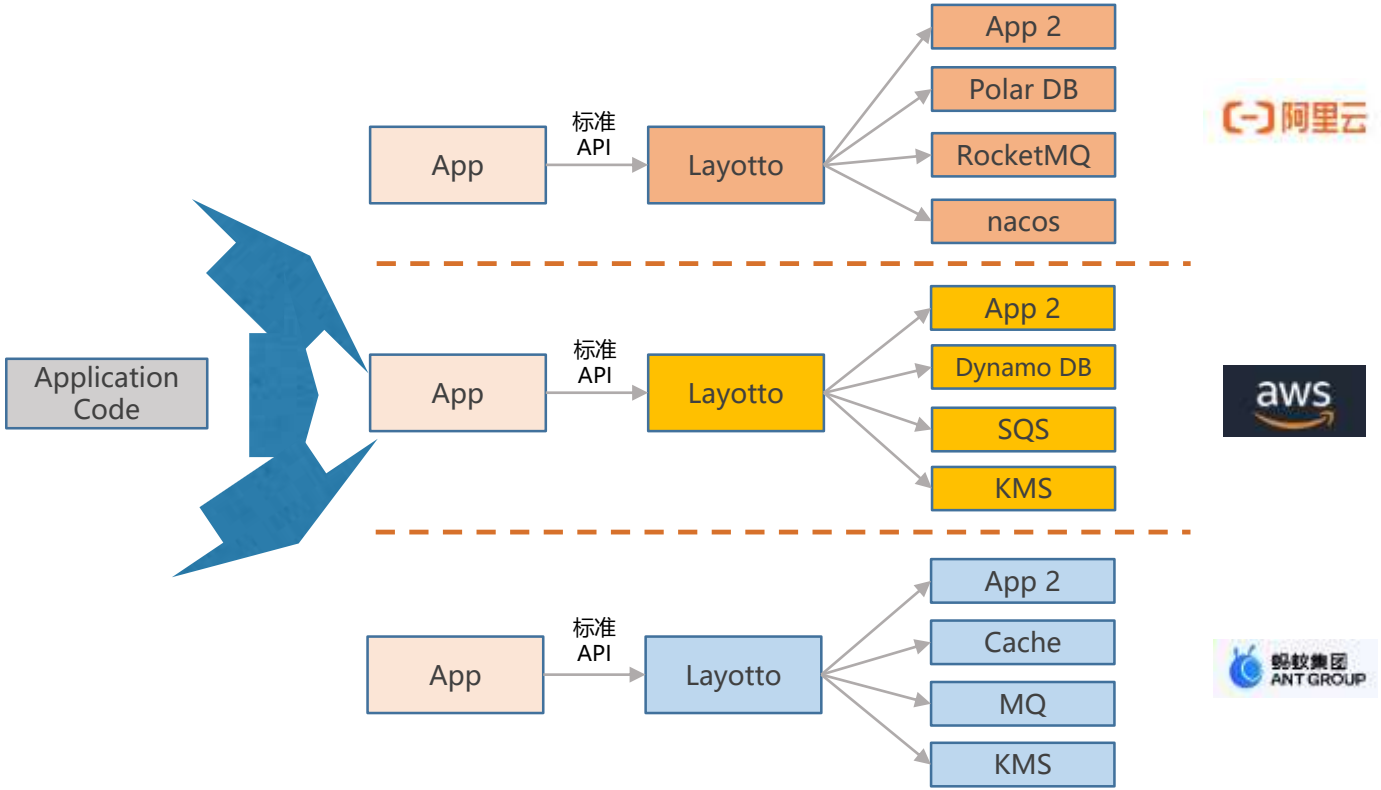
《The ABC of Lock-In》

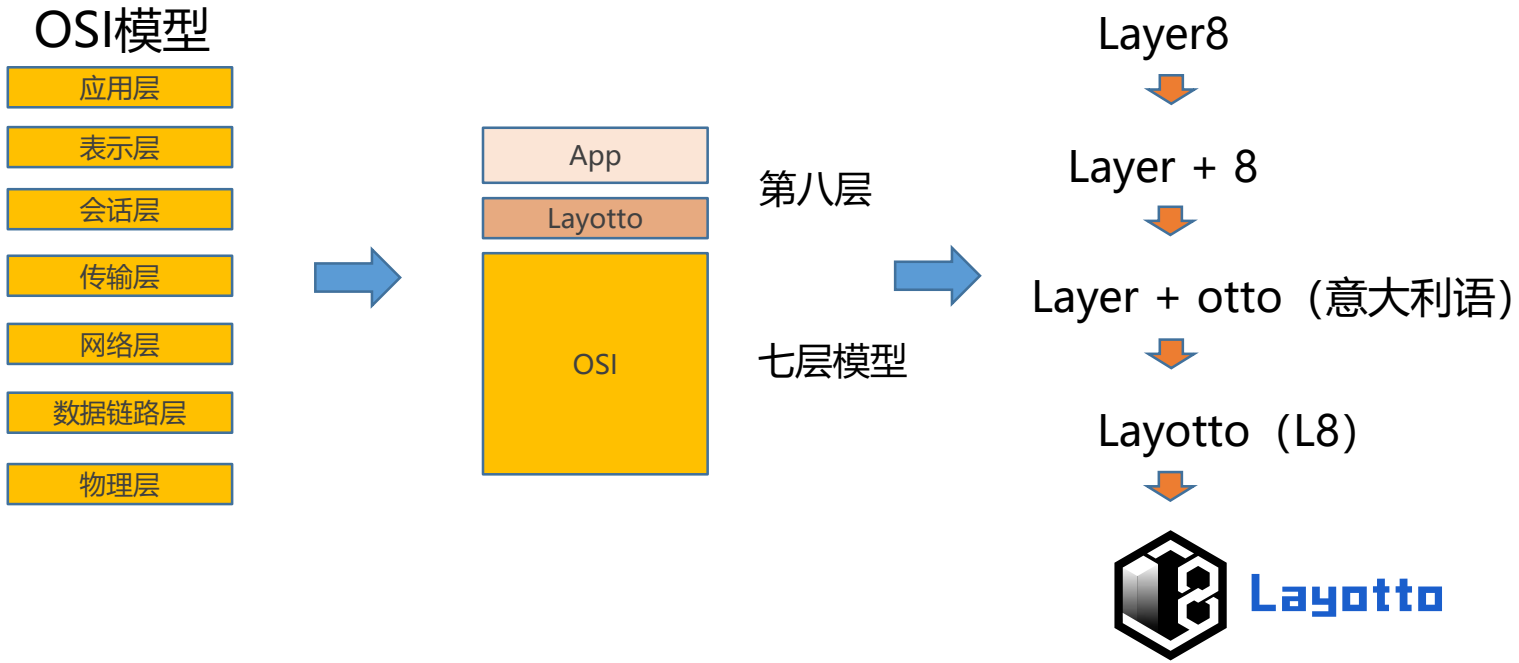
2012 年 2 月，一篇文章讲了一个有趣的故事
- 企业上了 A 云，用了 A 云的专有服务
- 被 A 云锁定
- 供应商 C 找上门 B 云也很好，你通过我的服务可以无缝使用 A 云和 B 云
- 被 C 锁定

Layotto


App
业务逻辑
gRPC Client
gRPC Server
gRPC
阿里、蚂蚁
、Dapr 共建
P
a
a
S
HTTP
Layotto
Runtime API
Actuator
API
Config
Pub/Sub
RPC
State
Lock
...
MOSN
Envoy
优先复用
其次开发
Components
Config
MQ
RPC

Layotto 移植性


App 2
Polar DB
RocketMQ
nacos
App
标准
API
Layotto
阿里云
Application
Code
App 2
Dynamo DB
SQS
KMS
App
标准
API
Layotto
aws
App 2
Cache
MQ
KMS
App
标准
API
Layotto
蚂蚁集团
ANT GROUP


OSI模型
应用层
表示层
会话层
传输层
网络层
数据链路层
物理层
App
Layotto
OSI
第八层
七层模型
Layer8
Layer + 8
Layer + otto（意大利语）
Layotto（L8）
Layotto

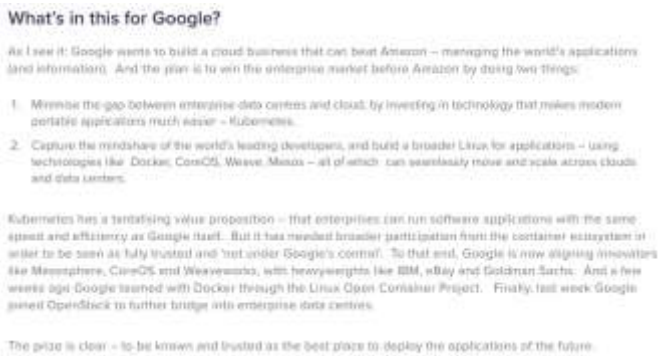
问题
解决了
吗?

如何解决厂商绑定的 ABC 问题?

遥想谷歌当年,如何推广 Kubernetes ?

- 联盟。联合众多厂商成立、参与 CNCF，让企业用户相信 Kubernetes 完全可信且『不受 Google 控制』

- 中立 API。让用户在各种环境中用 Kubernetes，这样更便于迁移到云上



What's in this for Google?

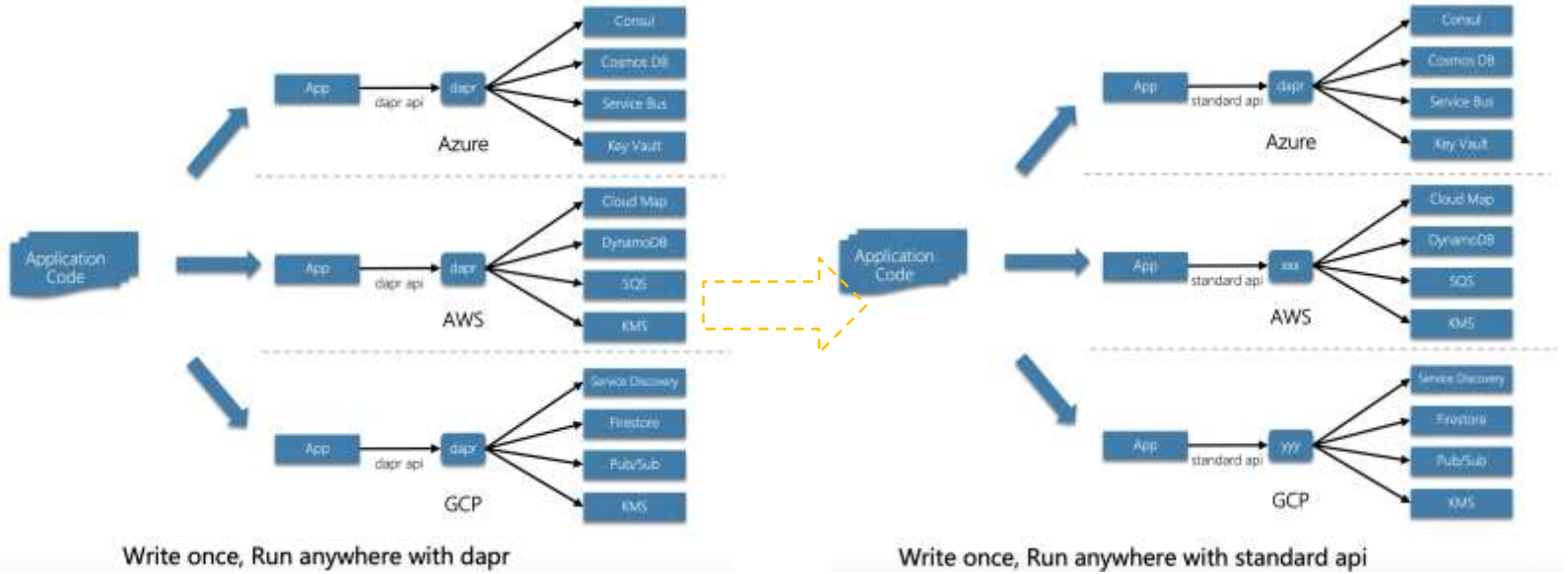
As I see it: Google wants to build a cloud business that can beat Amazon – managing the world's applications (and information). And the plan is to win the enterprise market before Amazon by doing two things:

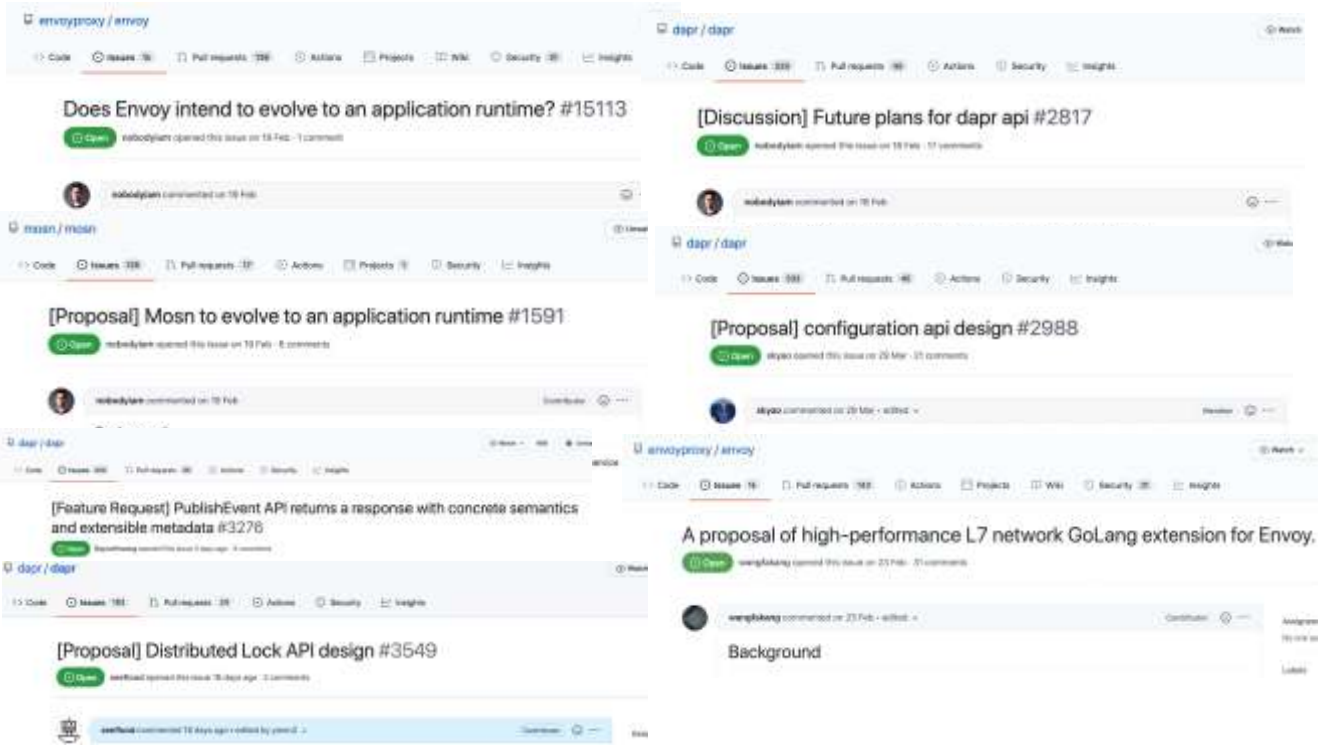
1. Minimise the gap between enterprise data centres and cloud, by investing in technology that makes modern portable applications much easier – Kubernetes.
2. Capture the mindshare of the world's leading developers, and build a broader Linux for applications – using technologies like Docker, CoreOS, Weave, Mesos – all of which can seamlessly move and scale across clouds and data centers.

Kubernetes has a tantalising value proposition – that enterprises can run software applications with the same speed and efficiency as Google itself. But it has needed broader participation from the container ecosystem in order to be seen as fully trusted and 'not under Google's control'. To that end, Google is now aligning innovators like Mesosphere, CoreOS and Weaveworks, with heavyweights like IBM, eBay and Goldman Sachs. And a few weeks ago Google teamed with Docker through the Linux Open Container Project. Finally, last week Google joined OpenStack to further bridge into enterprise data centres.

The prize is clear – to be known and trusted as the best place to deploy the applications of the future.

中立的 Runtime API spec




envoyproxy / envoy
Does Envoy intend to evolve to an application runtime? #15113
mosn / mosn
[Proposal] Mosn to evolve to an application runtime #1591
dapr / dapr
[Feature Request] PublishEvent API returns a response with concrete semantics and extensible metadata #3276
dapr / dapr
[Proposal] Distributed Lock API design #3549
dapr / dapr
[Discussion] Future plans for dapr api #2817
dapr / dapr
[Proposal] configuration api design #2988
envoyproxy / envoy
A proposal of high-performance L7 network GoLang extension for Envoy.
Background

	Dapr	Layotto
RPC 通信	✓	✓
RPC 治理	✗	✓
Config	🚀	✓
Pub/Sub	✓	✓
State	✓	✓
Actor	✓	✗
Sequencer	✗	✓
Lock	🚀	✓
Metadata	✓	✗
Actuator	✗	✓

✓ 支持　✗ 不支持　🚀 建设中

另一种视角
看待Runtime API

OS

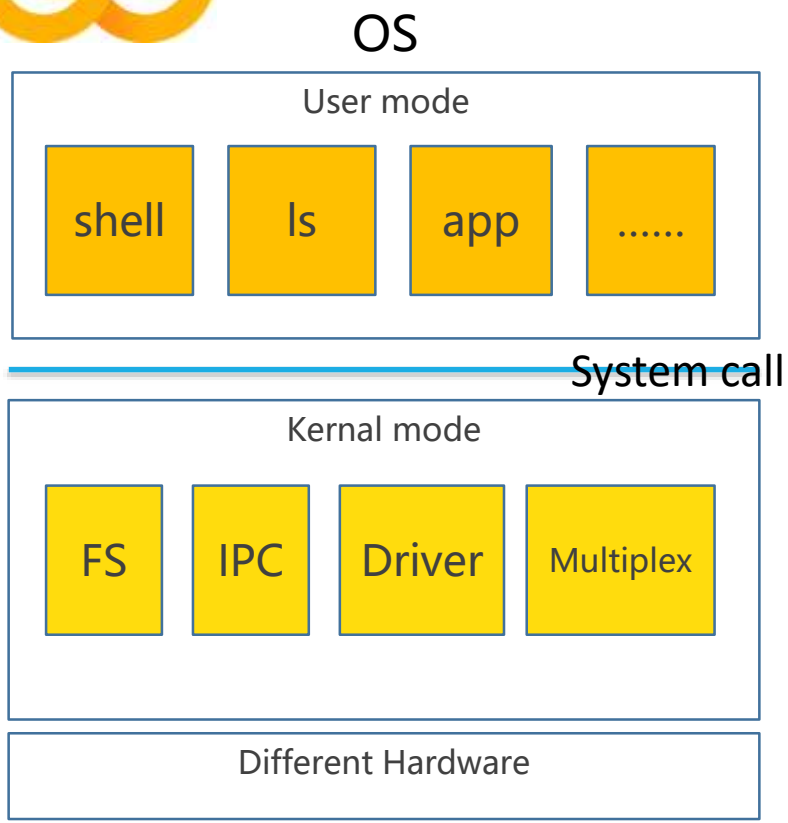
抽象的看：
OS=治理软件 + 抽象硬件
(把不同硬件抽象成一样的 API，让编程更简单)

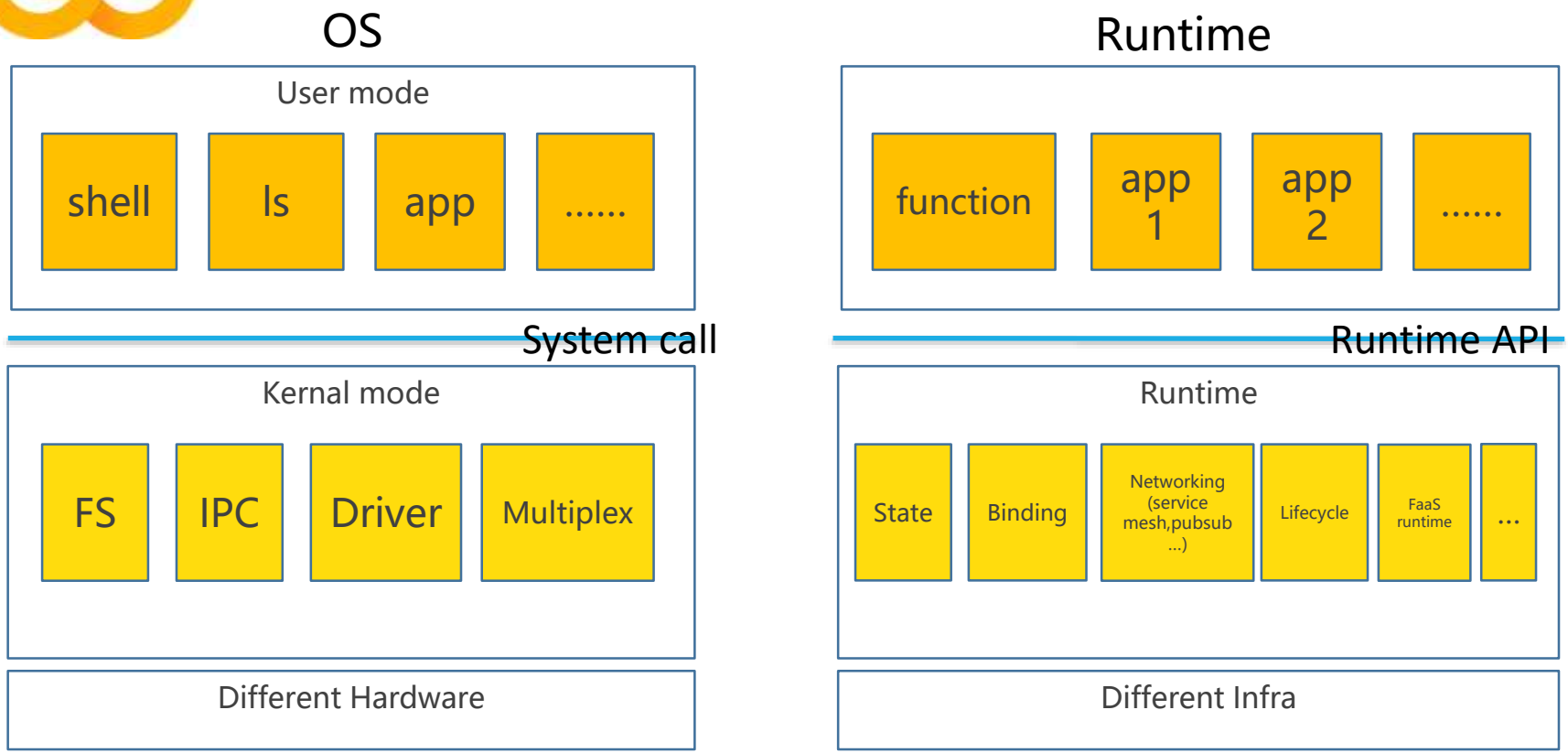
数据中心 OS

K8S = 治理软件（容器）

Runtime API = 抽象基础设施

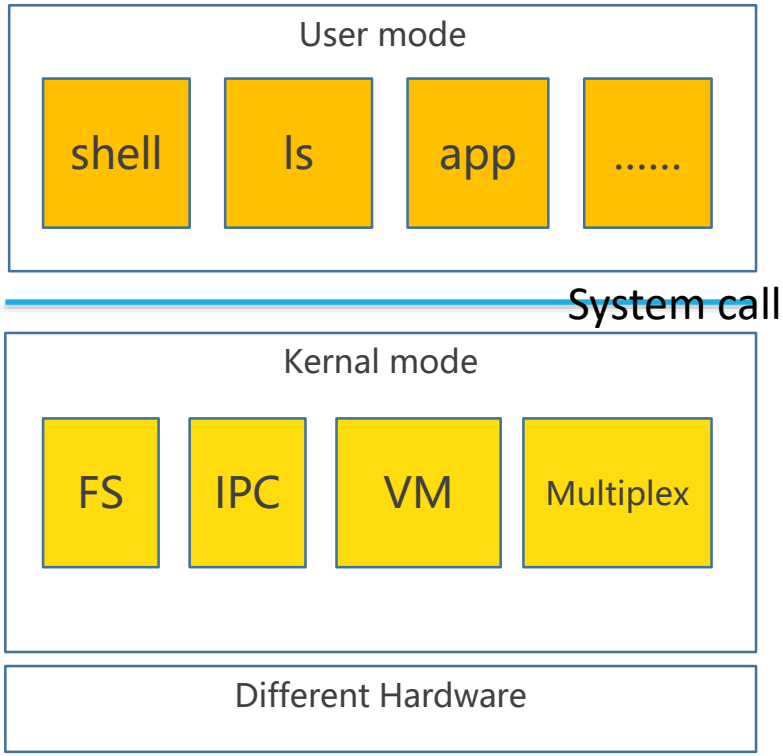
Runtime API + K8S = 可能是下一代分布式 OS


OS
User mode
shell
ls
app
.....
System call
Kernal mode
FS
IPC
Driver
Multiplex
Different Hardware


OS
User mode
shell
ls
app
.....
System call
Kernal mode
FS
IPC
Driver
Multiplex
Different Hardware
Runtime
function
app 1
app 2
.....
Runtime API
Runtime
State
Binding
Networking (service mesh,pubsub ...)
Lifecycle
FaaS runtime
...
Different Infra

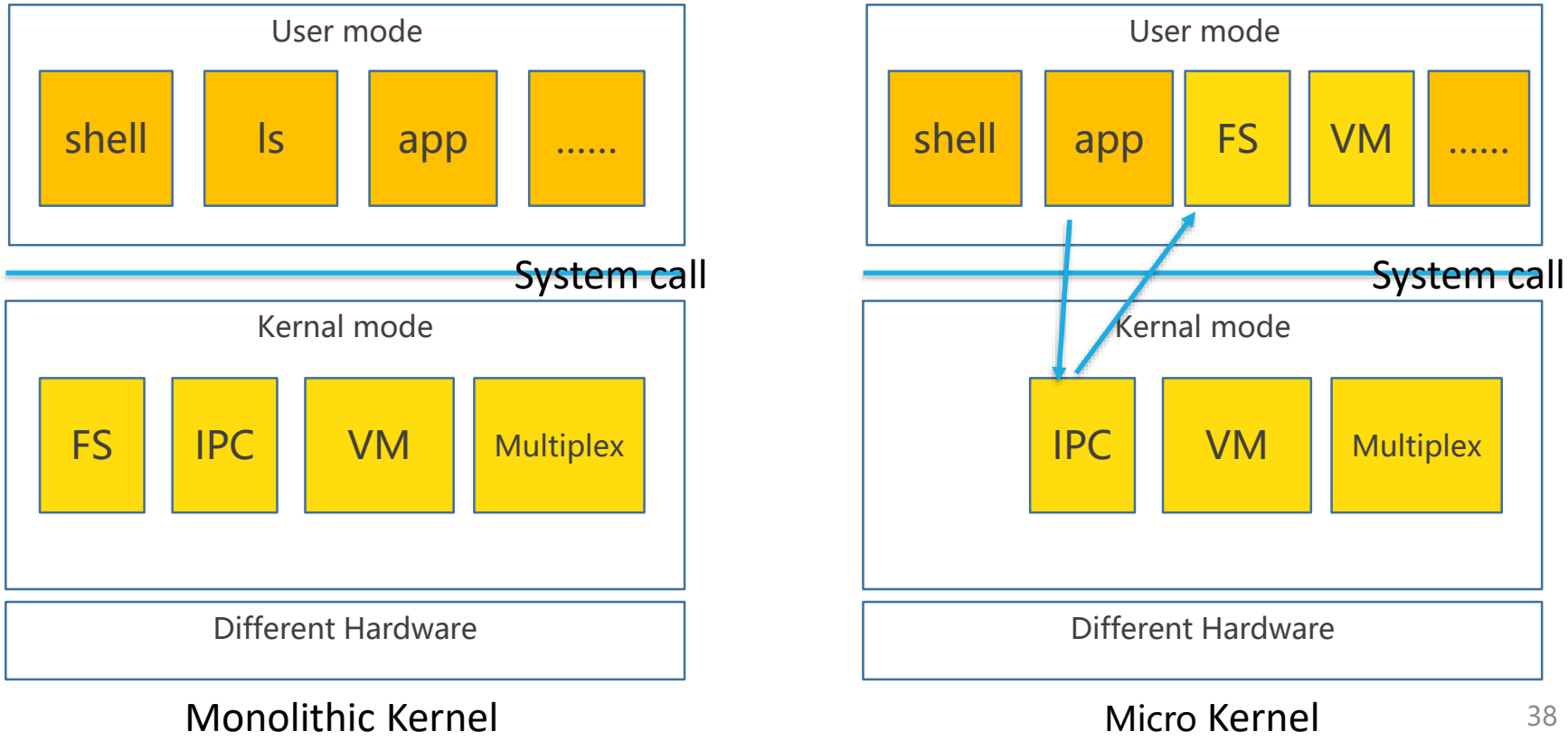
是不是感觉有点像？
别急，
再看看他们面临的设计问题……

设计OS kernel: 宏内核(Monolithic Kernel)
还是 微内核(Micro Kernel)?

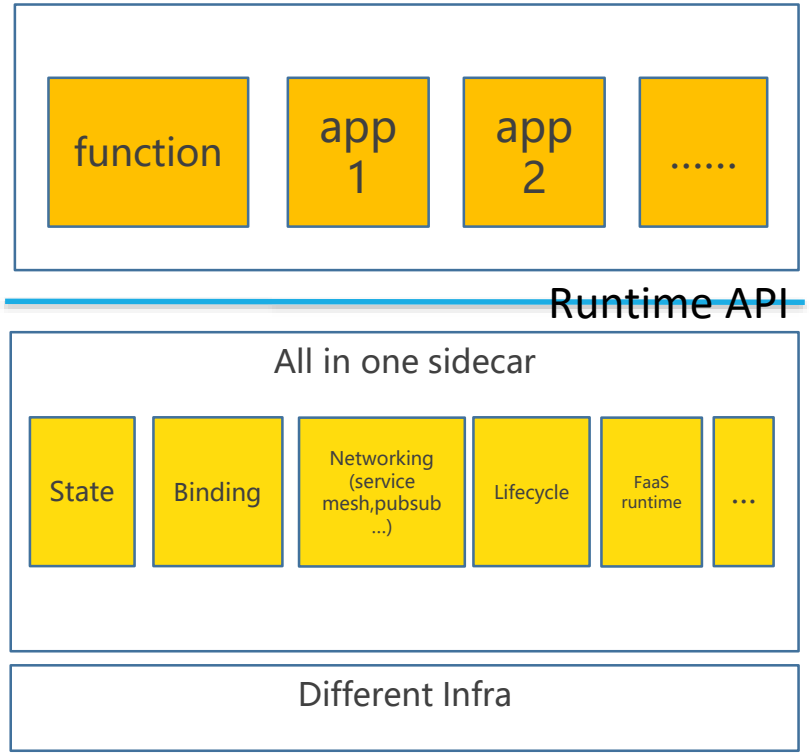


Monolithic Kernel

设计OS kernel: 宏内核(Monolithic Kernel)
还是 微内核(Micro Kernel)?

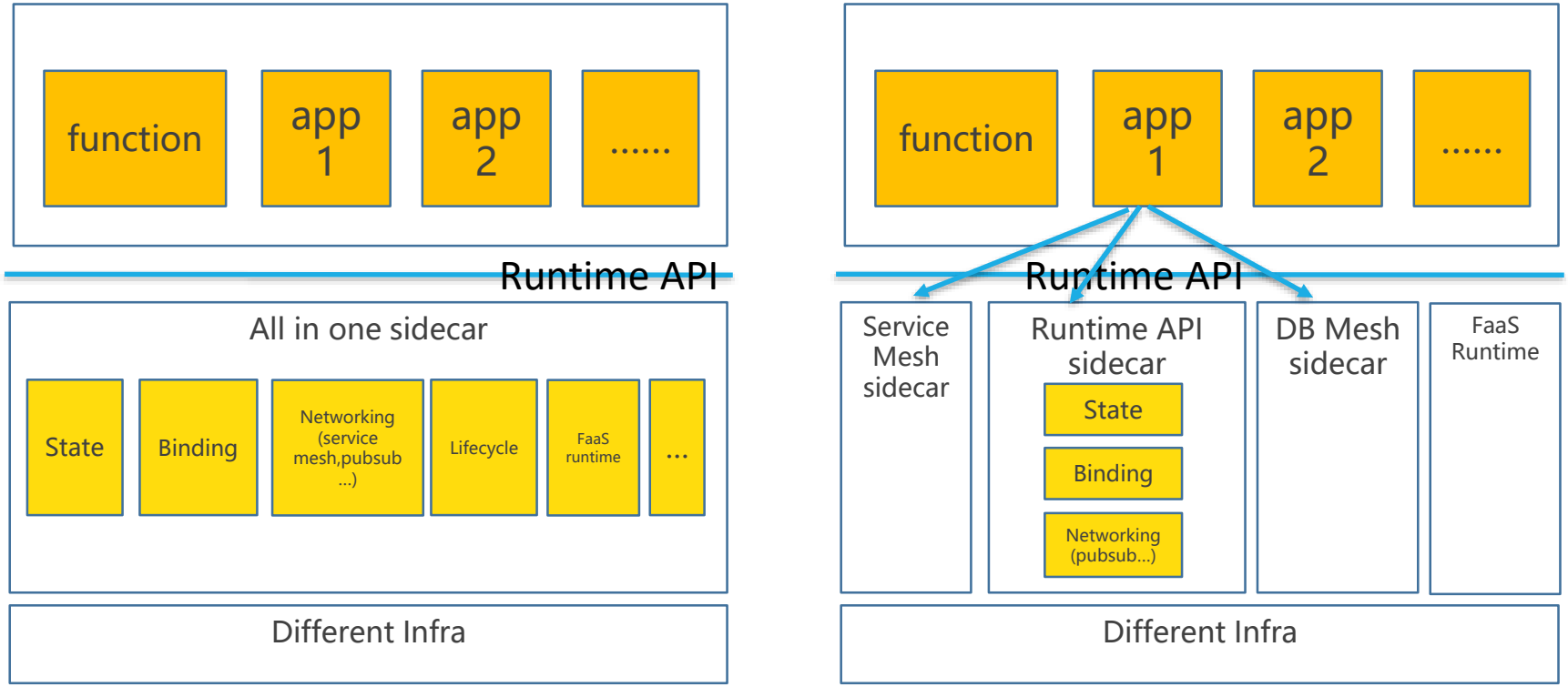


设计Runtime: 单体sidecar(Monolithic sidecar)
还是 微sidecar(Micro sidecar)?

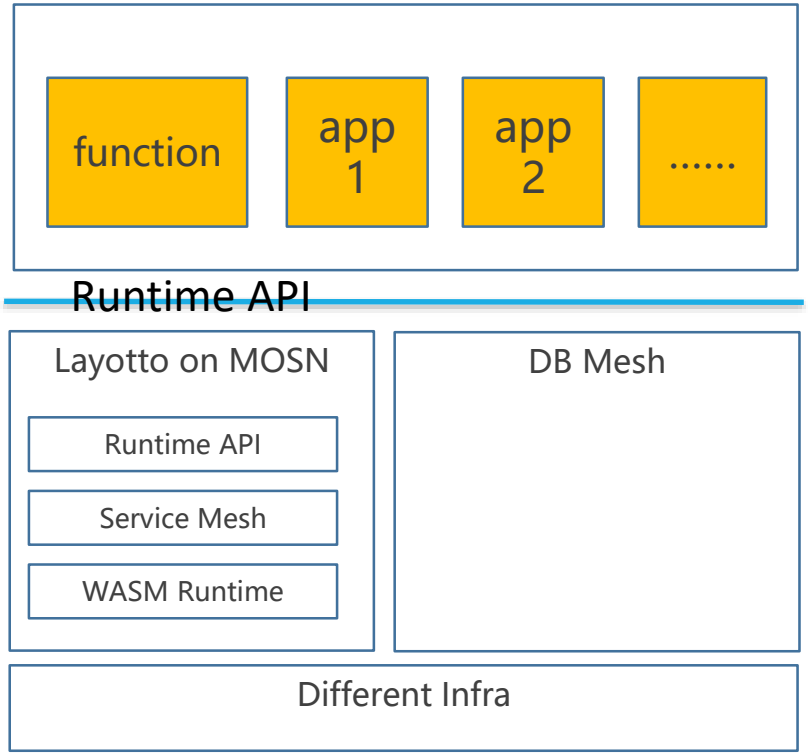


Monolithic sidecar

设计Runtime: 单体sidecar(Monolithic sidecar)
还是 微sidecar(Micro sidecar)?

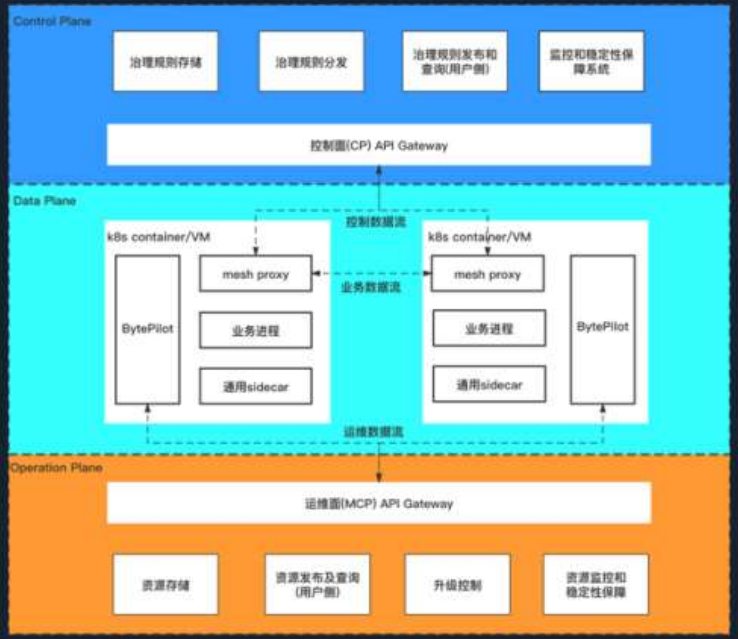


蚂蚁的选择


function
app 1
app 2
.....
Runtime API
Layotto on MOSN
Runtime API
Service Mesh
WASM Runtime
DB Mesh
Different Infra

字节跳动的选择

字节的Service Mesh - ByteMesh架构



https://xie.infoq.cn/article/79f2f511e1493176c95be275b

Monolithic sidecar vs Micro sidecar

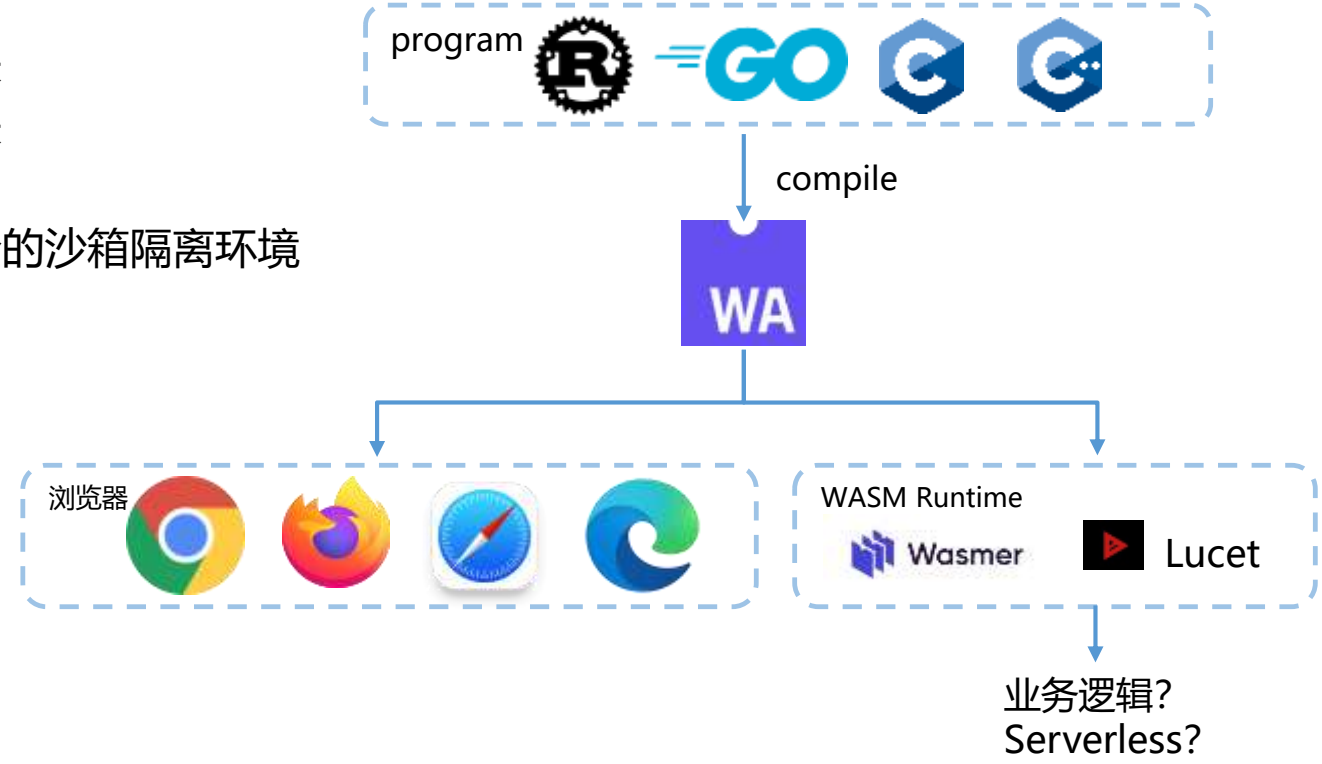
Who is right?

Who cares!

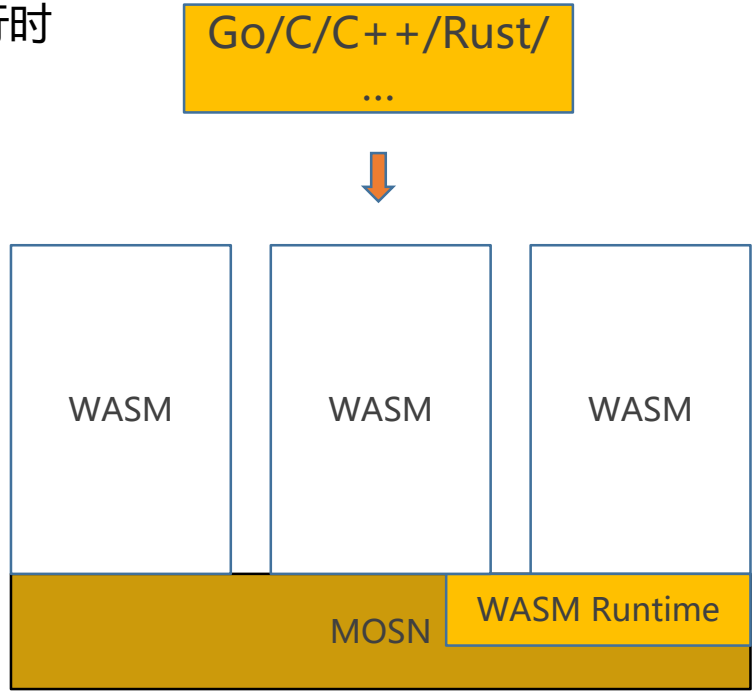
- 暴露出抽象API即可, 应用不关心有几个sidecar
- 具体部署几个全看取舍

WebAssembly in sidecar:
让业务逻辑跑在sidecar里

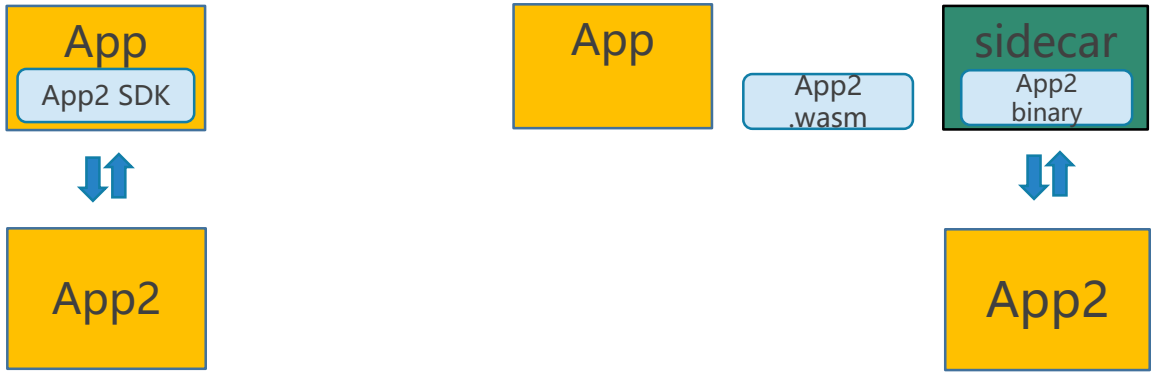
- 语言无关
- 平台无关
- 可移植
- 内存安全的沙箱隔离环境



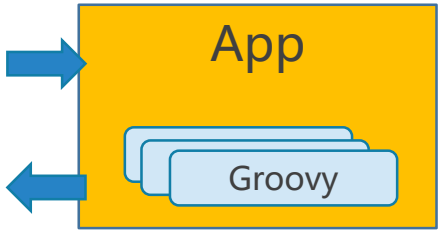
MOSN集成了WASM运行时

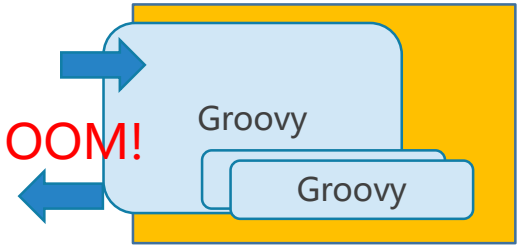


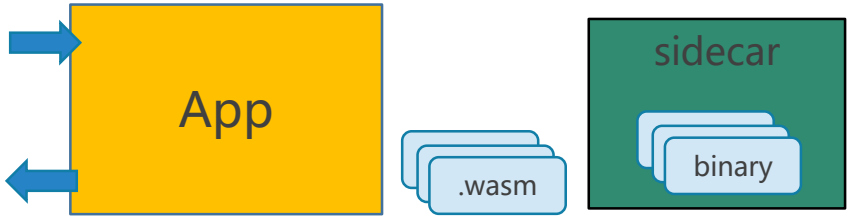
Reloadable SDK



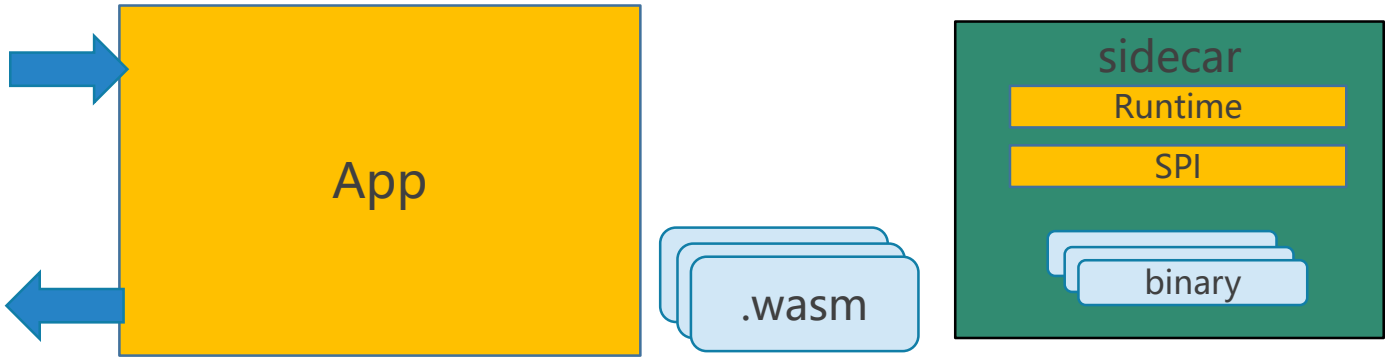
Sandbox as a service



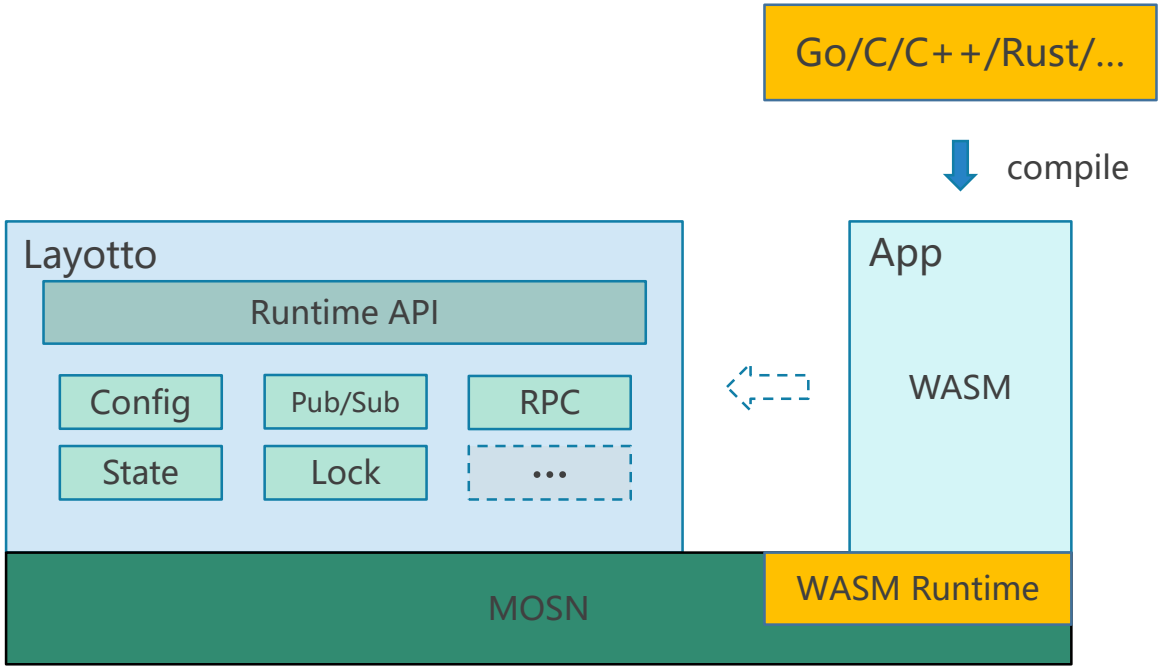




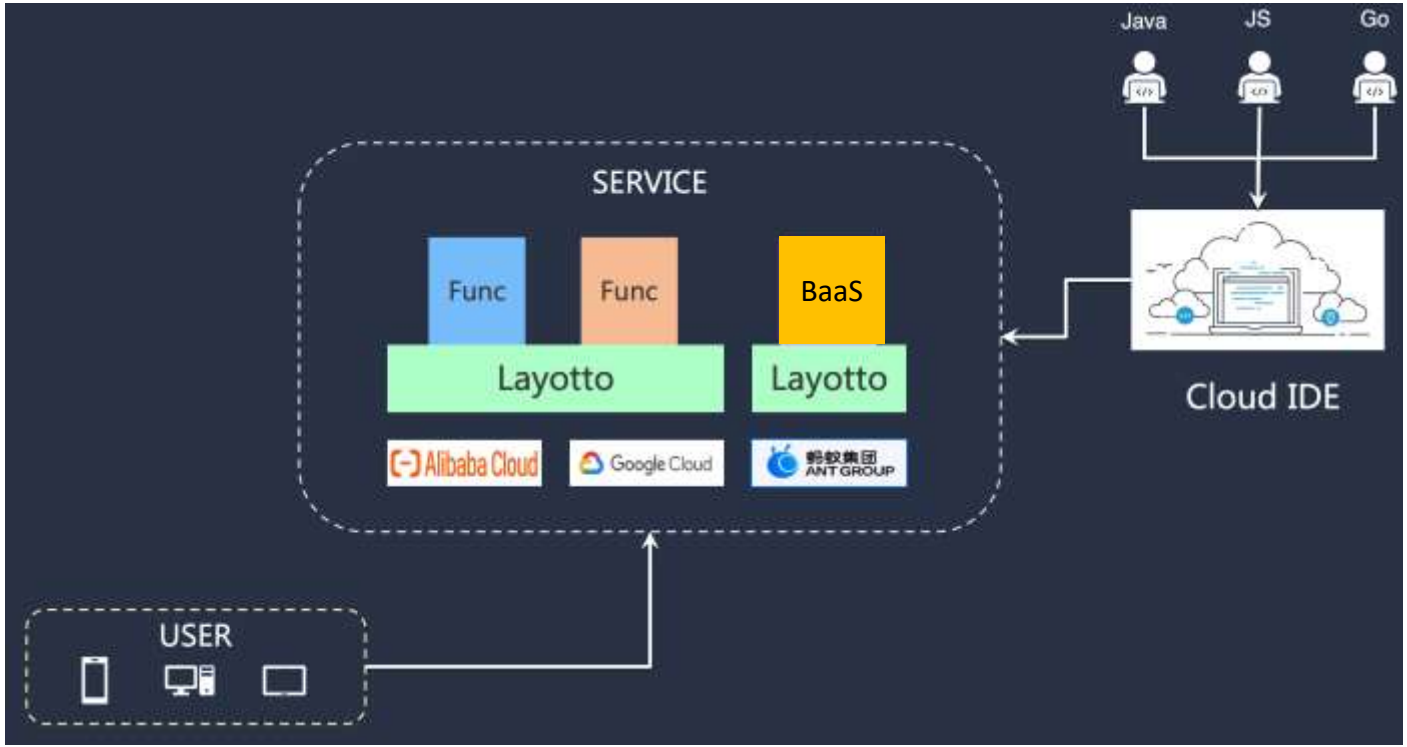
sidecar extension



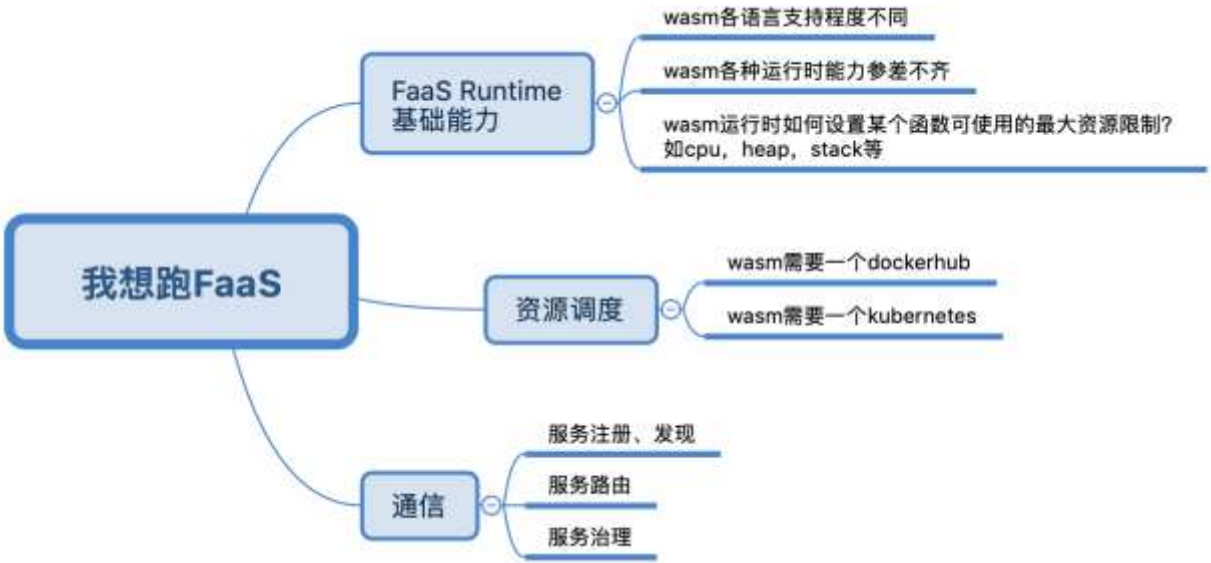
FaaS


Go/C/C++/Rust/...
compile
Layotto
Runtime API
Config
Pub/Sub
RPC
State
Lock
...
App
WASM
MOSN
WASM Runtime

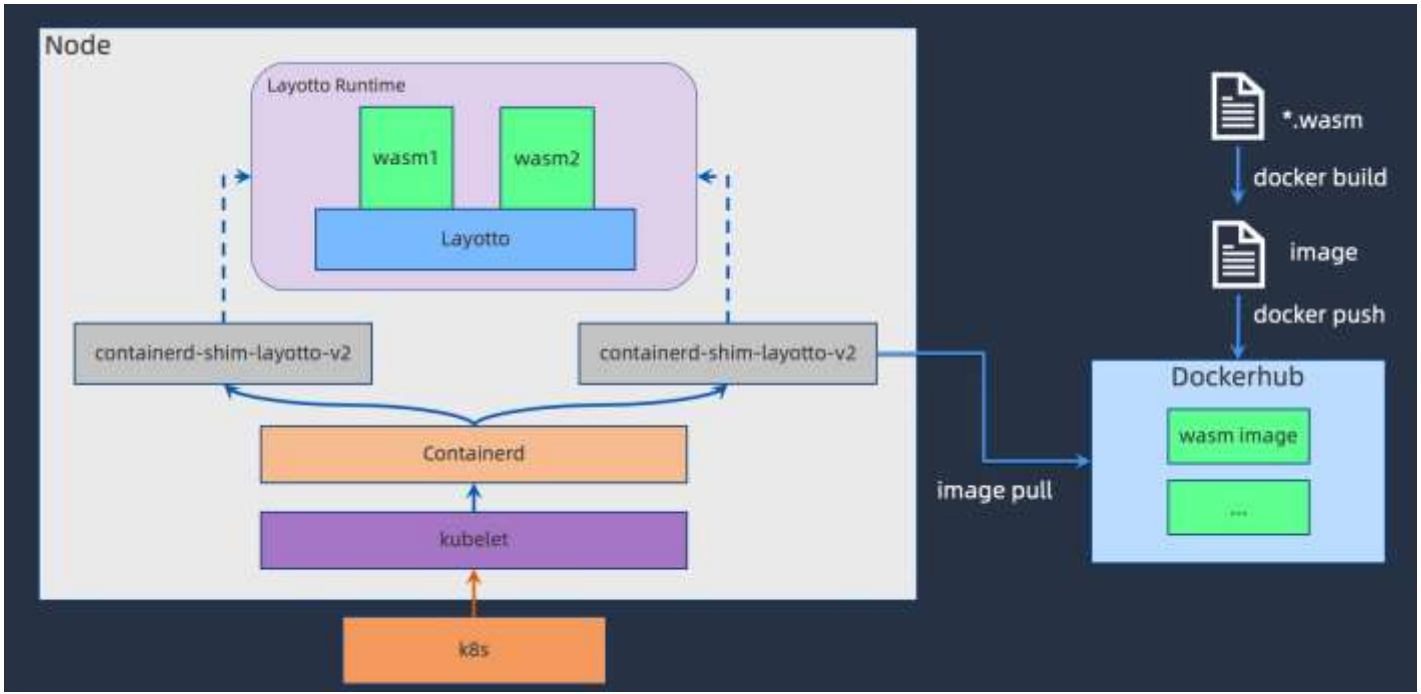
理想的serverless研发模式



现实……
想拿Webassembly跑FaaS?
目前的科技树还不成熟



Layotto的探索



https://mosn.io/layotto/#/zh/design/faas/faas-poc-design

展望2022：
待解决的问题有哪些？

欢迎讨论

https://github.com/mosn/layotto/pull/270

Multi-Runtime 2022：待解决的问题

1. API标准建设

根据落地用户的生产需求继续建设API标准，提交给Dapr社区共建。比如：

- 分布式锁API
- 配置API
- 延迟消息API

2. 生态共建

如何让已经落地Service Mesh的用户平滑迁移到Multi-Runtime？目前在做的一件事是Layotto on Envoy支持；

能否让Runtime API更好的融入K8S生态？目前在做的事是Layotto集成进k8s生态；

3. 服务早期生产用户

开源要做通用的，解决生产问题的功能。观察早期生产用户，目前面临以下问题：

3.1. 扩展性

让整个项目可扩展，比如某个公司想用layotto但是又想扩展一些自己的功能，要么能自己起一个项目，import开源layotto后通过钩子做一些扩展，要么能通过动态连接库之类的办法去扩展layotto二进制文件。目前这两种办法,dapr和layotto都没法做到。想扩展只能fork出来改代码

3.2. 稳定性风险

import开源Layotto之后，panic风险巨大,因为依赖了Dapr所有组件，这些组件用的库五花八门，可能panic，可能依赖冲突。能否通过按需编译、隔离性设计来减少panic风险？

目前开源项目的测试投入相对于公司里的测试流程来说少太多了。怎么建设开源测试体系：

3.3. 可观测性

> 以前没service mesh的时候，有问题我能自己查；后来有了service mesh，遇到问题我只能找别人来查了 ——某测试同学

总结

- Service Mesh : 通信中间件下沉，sidecar 实现组织架构上的解耦
- Multi Runtime: 所有中间件下沉:

  业务逻辑和基础设施分离
  多语言治理
  同一套代码移植到不同组件

- Runtime API：真正的供应商解绑
- WebAssembly in sidecar：让业务逻辑跑在 sidecar 里

Service Mesh 落地之后，架构演进的思路是？

为 sidecar

注入

灵魂

终

https://github.com/mosn/layotto

微信扫码进群
与五湖四海的开发者们
进行技术交流，探索技术创新

扫码关注公众号，参与活动抽奖
与 2.8W+ 技术精英
交流技术干货 & 开源组件