Serverless 实战与进阶

Serverless in action and advanced

陈仲寅（张挺）

阿里巴巴 - 淘系前端 - 前端架构 - Node.js 架构

陈仲寅（花名：**张挺**）

就职于 阿里巴巴淘系技术部前端架构团队



负责维护 midwayjs 品牌旗下的 midway，pandora，sandbox，injection 等产品。

陈仲寅（花名：**张挺**）

就职于 阿里巴巴淘系技术部前端架构团队

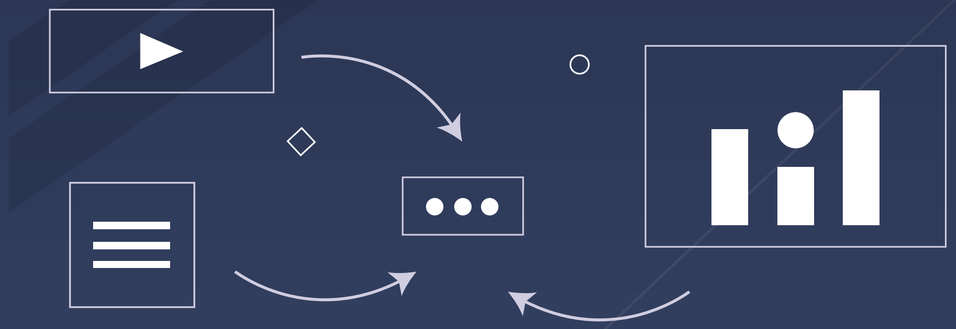
≈10%
*.ts

≈ 6%
d.ts

我们希望所有的库，API 定义都有 TypeScript 的身影，感觉任重而道远。

陈仲寅（花名：**张挺**）

就职于 阿里巴巴淘系技术部前端架构团队



负责制定集团 Serverless 规范，以及参与各个平台，工具链，在协议 和 API 层面的一致性落地。

大纲

- 业界现有的 Serverless 体系介绍
- 企业级 Serverless 开发模式
- 企业级 Serverless 体验和实践
- 从传统应用迁移到 Serverless 体系
- 私有化 Node.js 运行时方案

准备工作

- Node.js > 10 （nvs，nvm），自带 npm
- 尽量 mac os
- 阿里云 + 腾讯云 账户

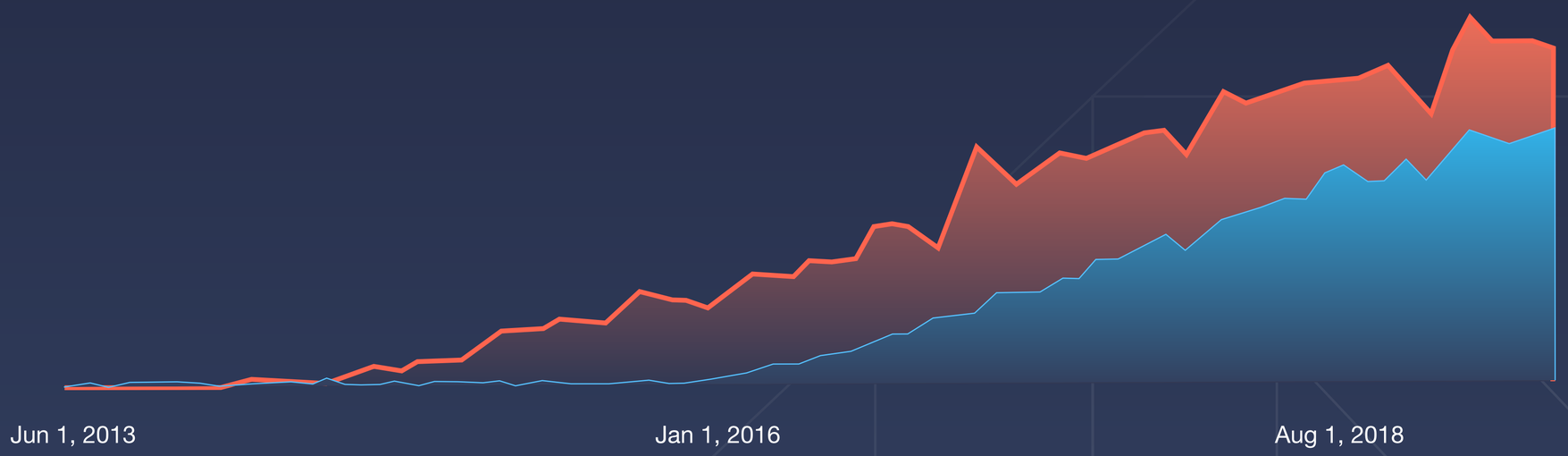
知识储备

- 有开发 Node.js Web 应用或者工具链基础
- 常见的云平台使用常识
- Serverless 体系的基础知识，或者更多的使用

第一章节
现有的 Serverless 体系介绍

社区和生态

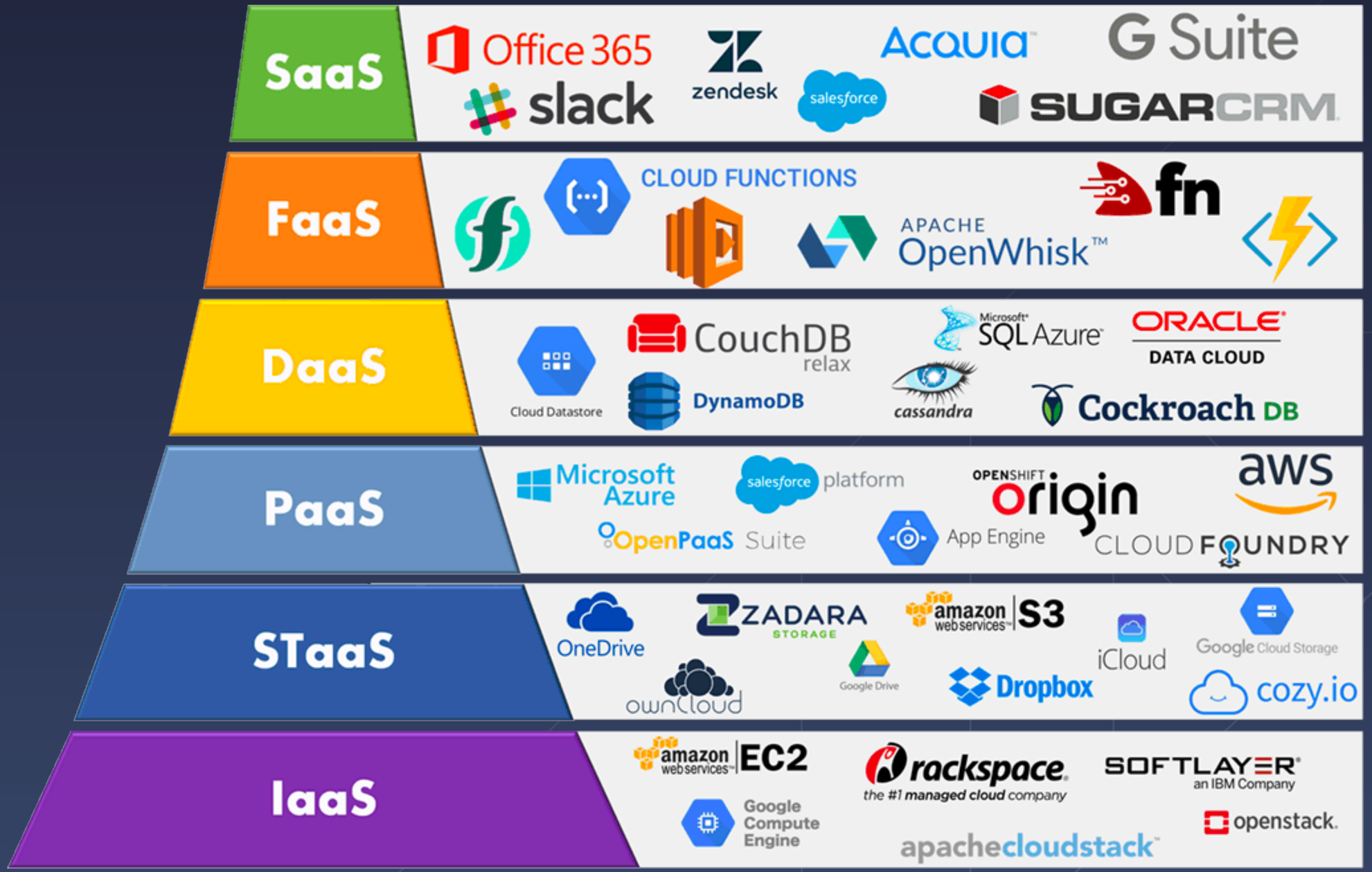
Serverless 的热情在不断高涨



其中红色是 **Microservices**，蓝色是 **Serverless**。

Top Growing Cloud Services

Place	Service	Growth Rate	2017 Use	2018 Use
#1	Serverless	75%	12%	21%
#2	Container-as-a-service	38%	14%	19%
#3	DBaaS SQL	26%	35%	44%
#4	DBaaS NoSQL	22%	23%	28%


SaaS
Office 365
slack
zendesk
salesforce
Acquia
G Suite
SUGARCRM
FaaS
CLOUD FUNCTIONS
APACHE
OpenWhisk™
fn
DaaS
Cloud Datastore
CouchDB
relax
DynamoDB
Microsoft
SQL Azure
cassandra
ORACLE
DATA CLOUD
Cockroach DB
PaaS
Microsoft
Azure
salesforce platform
OpenPaaS Suite
App Engine
OPENSHIFT
origin
aws
CLOUD FOUNDRY
STaaS
OneDrive
ownCloud
ZADARA
STORAGE
Google Drive
amazon
web services
S3
Dropbox
iCloud
Google Cloud Storage
cozy.io
IaaS
amazon
web services
EC2
Google
Compute
Engine
rackspace
the #1 managed cloud company
SOFTLAYER
an IBM Company
openstack.
apachecloudstack

Serverless 大公司

AWS Lambda，最早被大众所认可的 Serverless 实现。

Function Compute，阿里云自研的 Serverless平台

Tecent SCF，腾讯云函数，Serverless 平台

Azure Functions，来自微软公有云的 Serverless 实现。

OpenWhisk，Apache 社区的开源 Serverless 框架。

Kubeless，基于 Kubernetes 架构实现的开源 Serverless 框架。

Fission，Platform9 推出的开源 Serverless 框架。

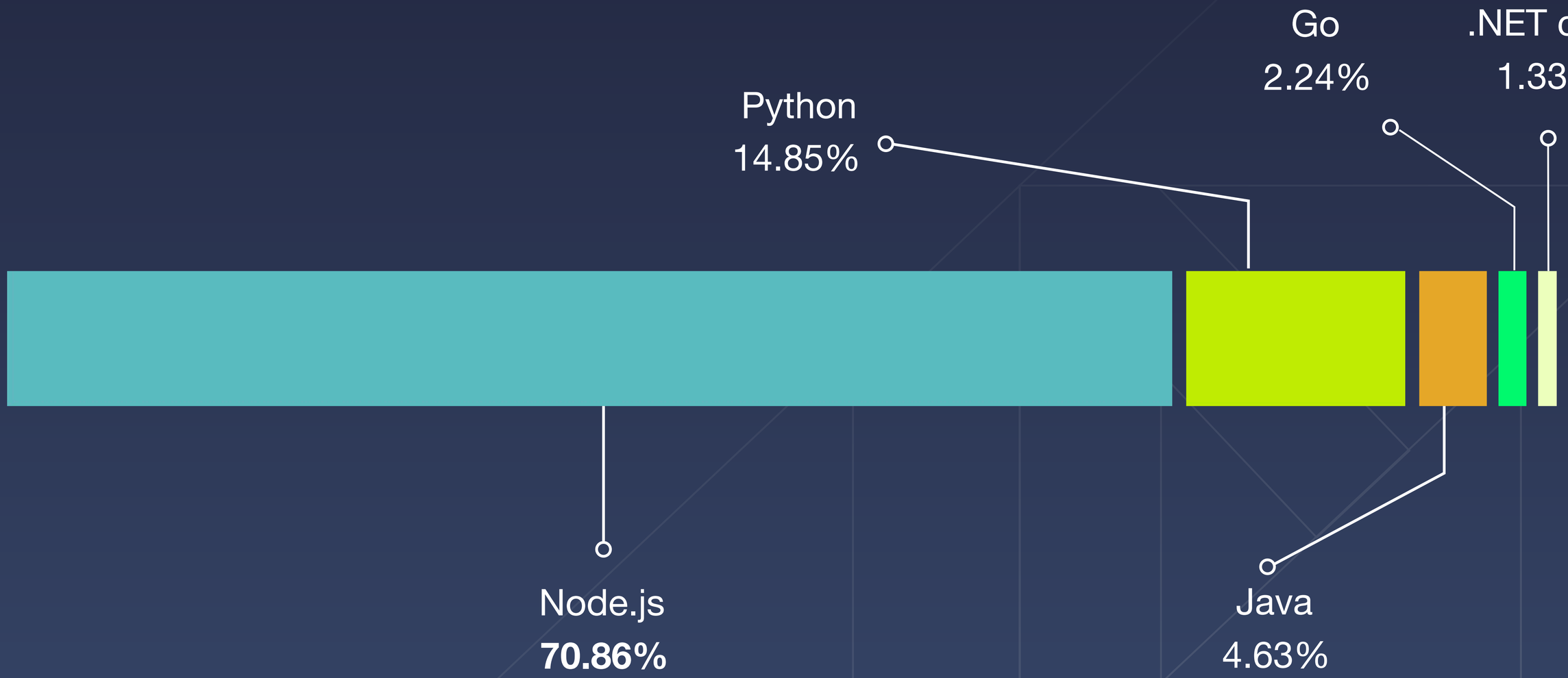
OpenFaaS，以容器技术为核心的开源 Serverless 框架。

Fn，来自 Oracle 的开源 Serverless 框架，由原 Iron Functions 团队开发。

语言支持

	AWS	Google	Azure	阿里云	腾讯云	华为云
NodeJs	√	√	√	√	√	√
Java	√		√	√		√
C#	√		√			
Python	√			√	√	√
Go	√*					√

语言支持

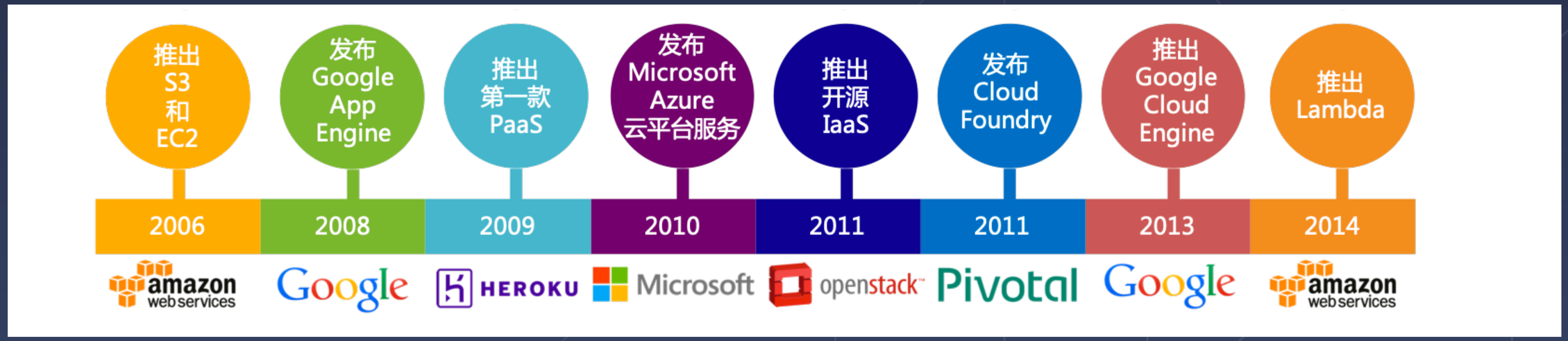


能力支持

	AWS	Google	Azure	阿里云	腾讯云	华为云
HTTP	API Gateway	√	√	API Gateway	API Gateway	API Gateway
定时	CloudWatch 定时		√	√	√	
对象存储	S3	Cloud storage	Blob storage	OSS	COS	OBS
消息队列	SNS	Pub/Sub topic	Queue storage	MQ	CMQ	SMN/ DMS
日志服务	CloudWatch Logs			√	√	
数据库	DynamoDB	Firestore	Cosmos DB	√	√	
流式计算	Kinesis	Firebase Realtime DB		Datahub		DIS
IoT	Alexa/IoT 按钮					
边缘计算	CloudFront			√		
其他	Email/Lex/Cognito	Crashlytics/ Analytics				

云计算发展

IaaS　PaaS　开源 PaaS　FaaS



我们举个例子


本地部署
(On-Premises)
You Manage
餐桌
苏打水
煤气/电炉
烤箱
火
披萨面团
番茄酱
配料
奶酪
在家自己做

方案一：IaaS


基础设施即服务
（IaaS）
You Manage
餐桌
苏打水
煤气/电炉
烤箱
火
披萨面团
番茄酱
配料
奶酪
Managed by Vendor
买成品回家做

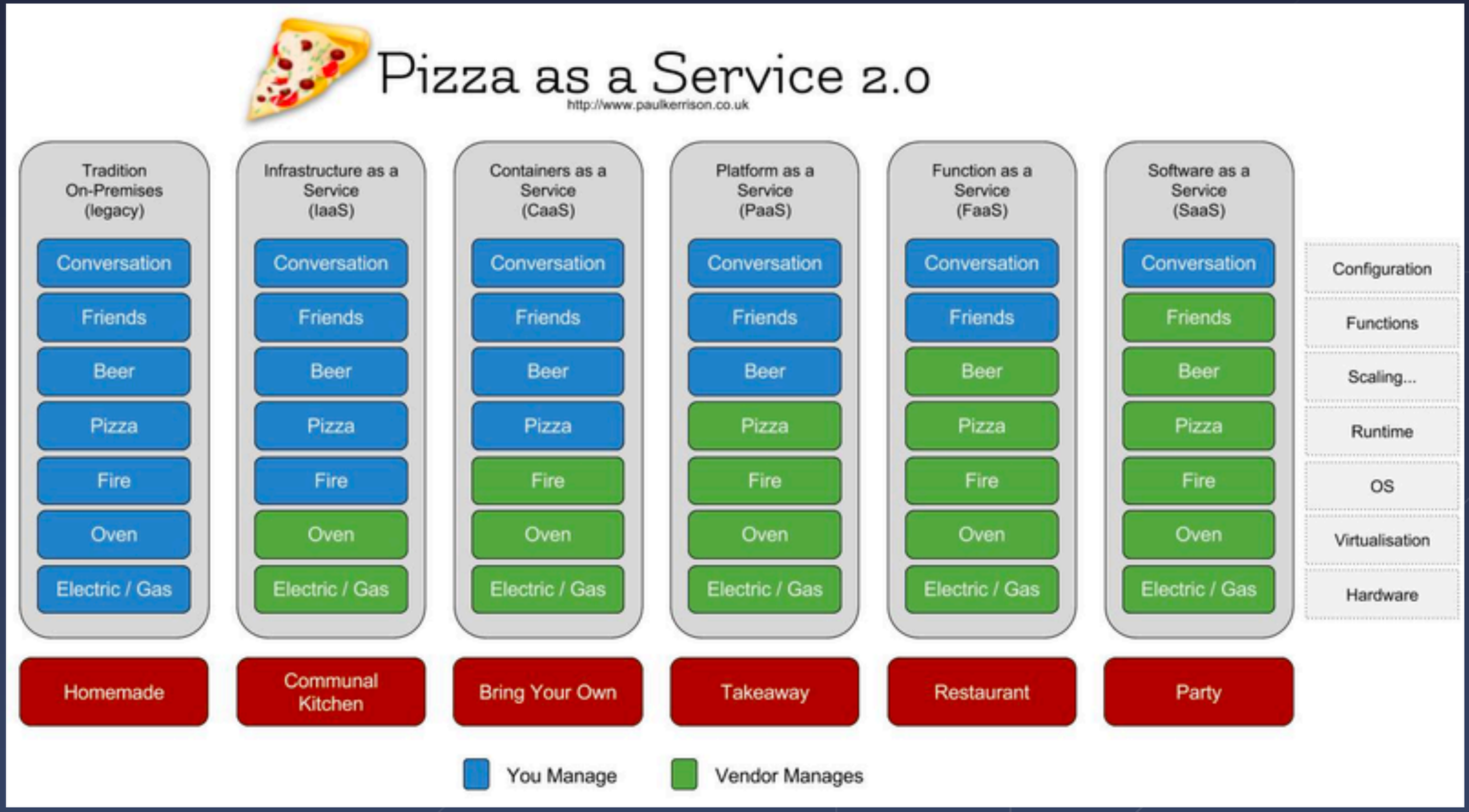
方案二： PaaS

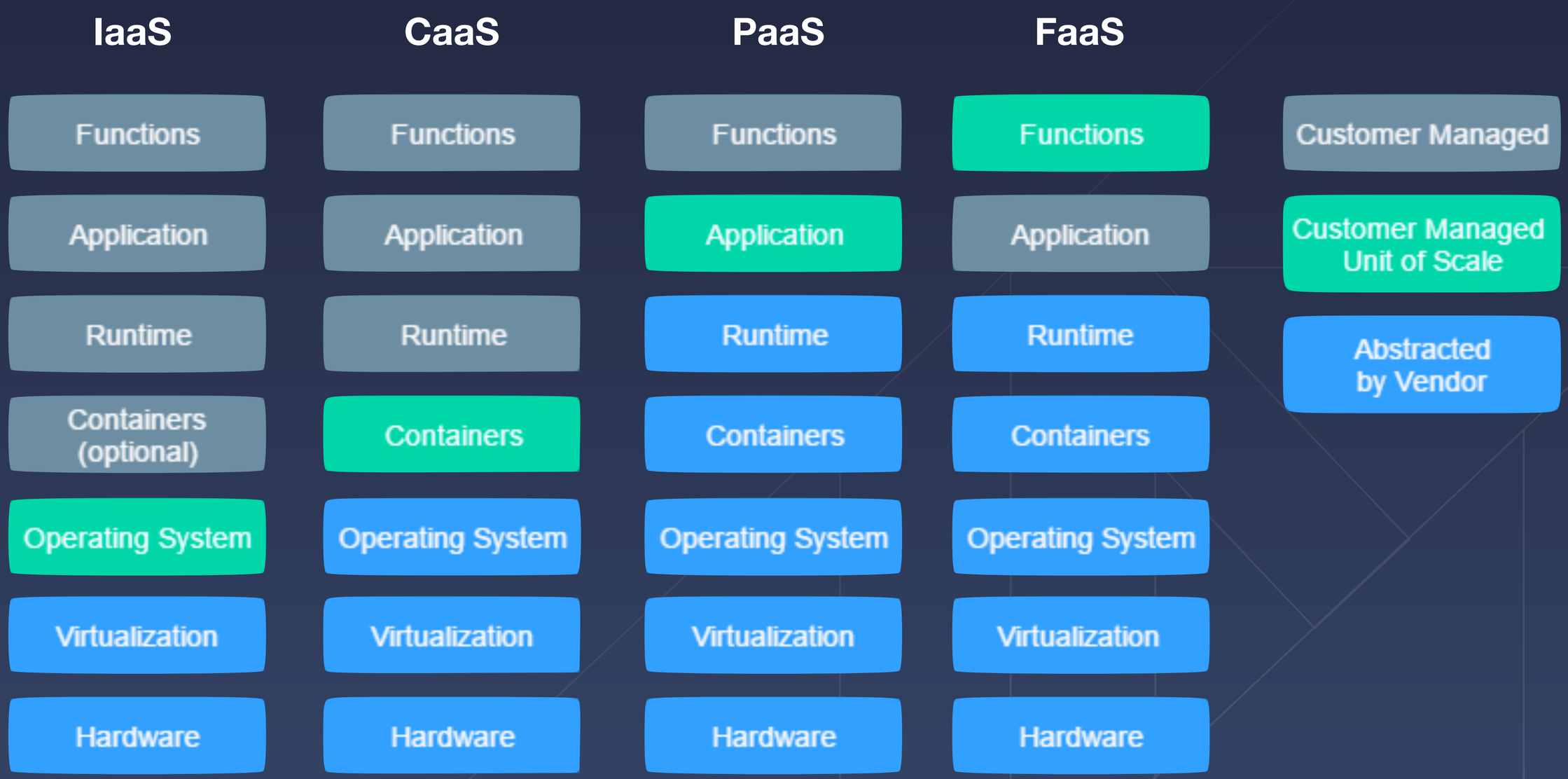

平台即服务
（PaaS）
You Manage
Managed by Vendor
餐桌
苏打水
煤气/电炉
烤箱
火
披萨面团
番茄酱
配料
奶酪
叫外卖
微信号: hr_opt

方案三： SaaS


Giulietta
Pizzeria e dolceria


软件即服务
（SaaS）
餐桌
苏打水
煤气/电炉
烤箱
火
披萨面团
番茄酱
配料
奶酪
Managed by Vendor
去披萨店吃

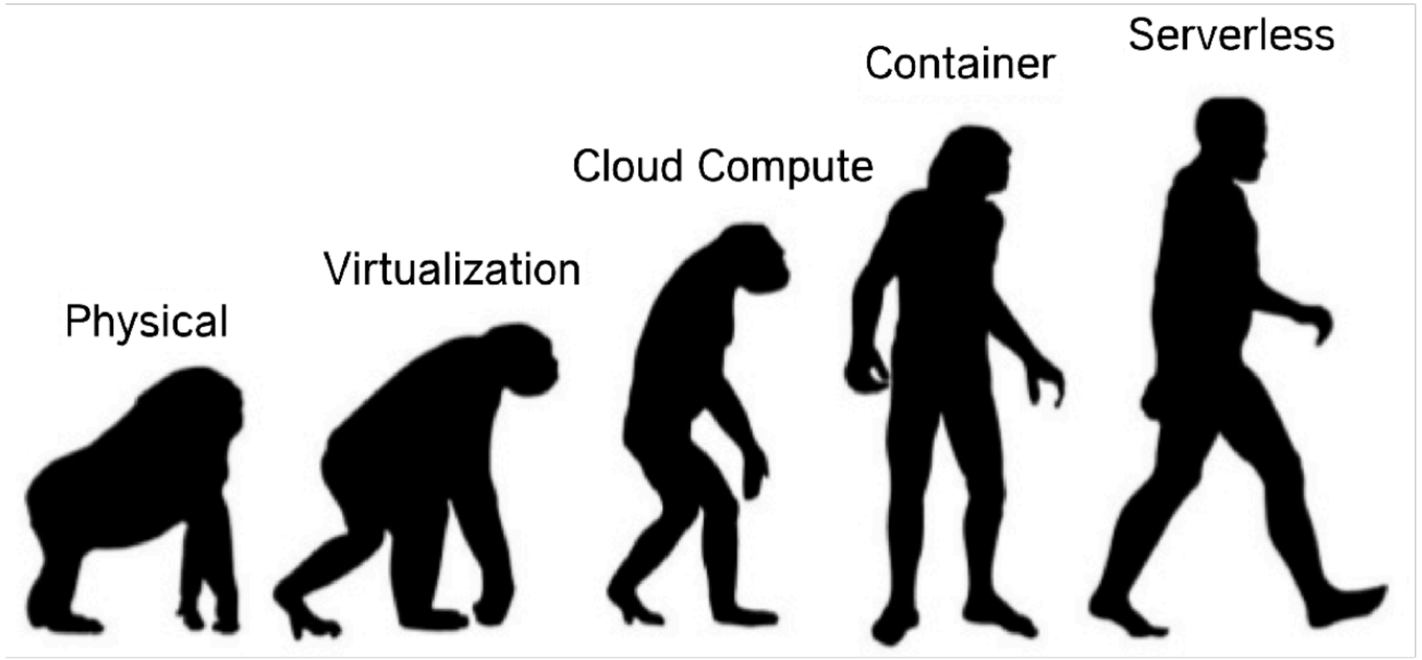



IaaS
CaaS
PaaS
FaaS
Functions
Application
Runtime
Containers (optional)
Operating System
Virtualization
Hardware
Functions
Application
Runtime
Containers
Operating System
Virtualization
Hardware
Functions
Application
Runtime
Containers
Operating System
Virtualization
Hardware
Functions
Application
Runtime
Containers
Operating System
Virtualization
Hardware
Customer Managed
Customer Managed Unit of Scale
Abstracted by Vendor

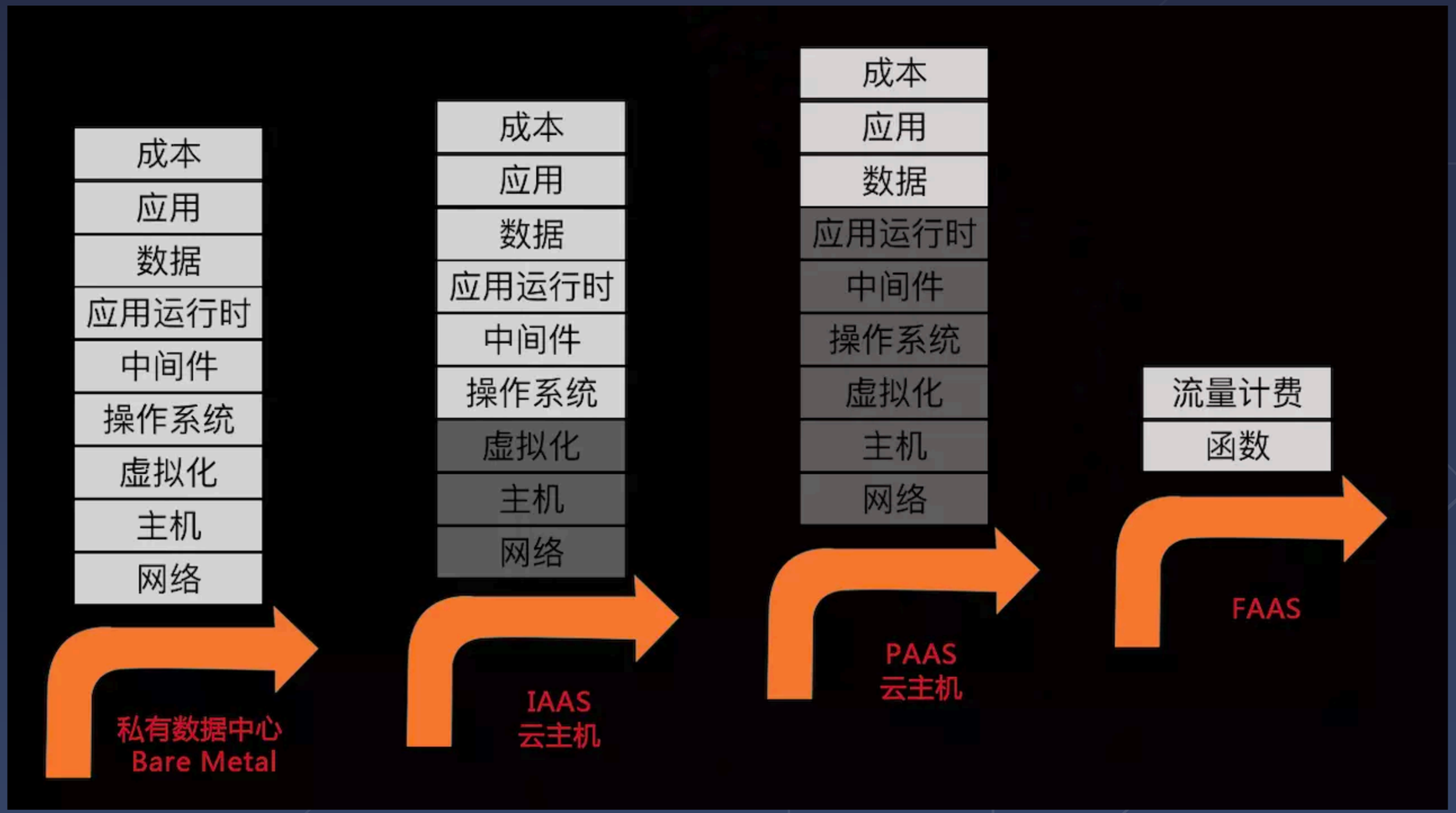

CLOUD
Pos 1

云计算发展

为了保证开发环境的正确(Bug不由环境引起)，想出一系列的隔离方式，从最早的物理服务器，一直在不断的抽象或者虚拟化服务器



产品维度的变革


成本
应用
数据
应用运行时
中间件
操作系统
虚拟化
主机
网络
私有数据中心
Bare Metal
成本
应用
数据
应用运行时
中间件
操作系统
虚拟化
主机
网络
IAAS
云主机
成本
应用
数据
应用运行时
中间件
操作系统
虚拟化
主机
网络
PAAS
云主机
流量计费
函数
FAAS

计费模型的变化

从固定的服务计费模型逐步转变为按需的资源计费模型

2019 前端四大方向

搭建服务

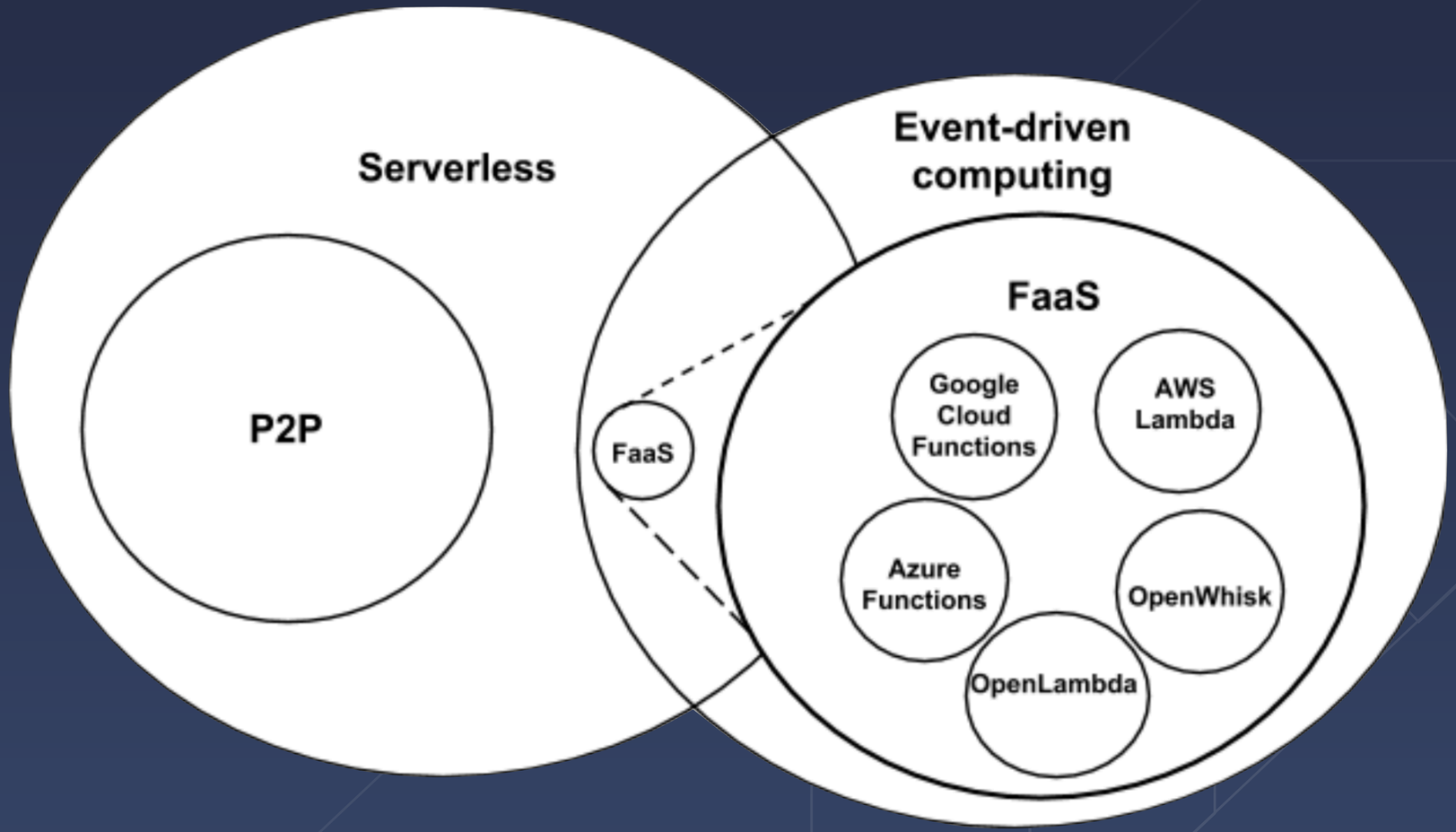
Serverless

智能化

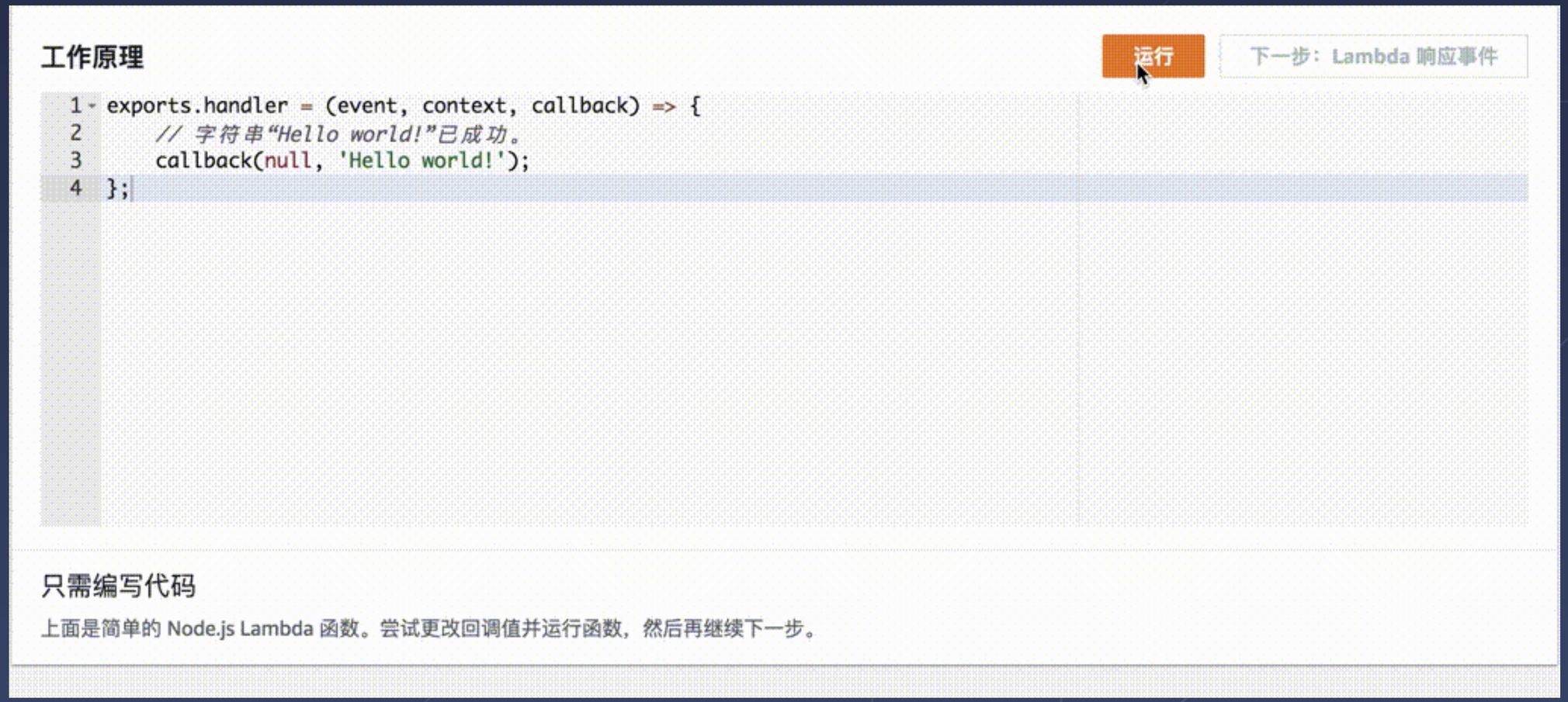
IDE

阿里经济体前端委员会四大技术方向

什么是 Serverless/FaaS



工作原理


工作原理
运行
下一步：Lambda 响应事件
1 exports.handler = (event, context, callback) => {
2 // 字符串“Hello world!”已成功。
3 callback(null, 'Hello world!');
4 };
只需编写代码
上面是简单的 Node.js Lambda 函数。尝试更改回调值并运行函数，然后再继续下一步。

基础准备

> The phrase “serverless” doesn’t mean servers are no longer involved. It simply means that developers no longer have to think that much about them.
>
> — *Ken fromm, 2012*

Serverless = FaaS + BaaS

Cloud Function　　Cloud Service

定义

云平台

- 各家云厂商，商业化部署 FaaS 函数的平台，函数最后承载的平台，现在主要指阿里云 FC 与腾讯云 SCF 两大平台

社区平台


阿里云
中国站
全部 域名 商标 公司
云服务器 ECS
搜索
购物车 文档
云服务器 ECS | 云数据库 RDS MySQL 版 | Web应用防火墙 | CDN
最新活动 产品分类 企业应用中心 解决方案 定价 云市场 支持与服务 合作伙伴与生态 开发者 了解阿里云
精选爆款
弹性计算 >
云服务器 ECS | 轻量应用服务器
数据库 >
MySQL | SQL Server | Redis
域名与网站 >
域名注册 | 网站建设 | 云虚拟主机
存储与CDN >
对象存储 OSS | CDN
物联网与云通信 >
物联网服务 | 短信服务
全部产品
云服务器 ECS
轻量应用服务器
GPU 云服务器
超级计算集群
批量计算
容器服务 Kubernetes 版
容器镜像服务
资源编排
图形工作站（公测中）
弹性裸金属服务器（神龙）
FPGA 云服务器
专有宿主机
弹性高性能计算 E-HPC
容器服务
弹性容器实例 ECI
弹性伸缩
函数计算
快速入口
高配云服务器免费试用
了解详情
企业级云服务器低至1折
了解详情


菜单
腾讯云
最新活动 HOT 产品 解决方案 定价 文档 云市场 开发者
热门
基础
安全
大数据
人工智能
企业应用
行业应用
开发者服务
容器服务
容器实例服务
弹性伸缩
批量计算
边缘计算机器 NEW
Serverless
云函数
Serverless Framework NEW
中间件

触发器

触发器，也叫 Event（事件），Trigger 等，特指触发函数的方式。
与传统的开发理念不同，函数不需要自己启动一个服务去监听数据，而是通过绑定一个（或者多个）触发器，数据是通过类似事件触发的机制来调用到函数。

目前云厂商最常见的触发器就是 http 和 timer、云存储等。

触发器

事件触发

云函数支持设置多种触发器来决定代码何时运行，在满足触发器条件（Event）时，代码自动开始运行，并根据请求自动调度基础设施资源实现自动伸缩和回收，提高计算效率。
目前支持以下触发器：

- 对象存储 COS：支持在特定的 COS Bucket 操作文件上传或文件删除等事件时触发云函数，可以对文件进行更多操作。例如：在图片上传到特定 Bucket 时，对其进行压缩或裁剪以适应不同分辨率的移动终端。
- 定时器：支持定时触发函数，助力用户构造更加灵活的自动化控制系统。
- 手动触发：支持通过 云 API /控制台 手动触发函数，帮助用户更便捷、更清晰地调试和使用云函数。
- CMQ 主题队列触发：由 CMQ Topic 主题队列内的消息触发，利用 CMQ 消息队列解耦事件，可以帮助用户和更多应用完成联动。
- Ckafka 消息队列触发：由 Ckafka Topic 主题队列内的消息触发，对消息进行处理，可以帮助用户实现日志聚合、消息存储等。
- API 网关触发：支持 API 网关中的 API 配置后端为云函数，在 API 接收到客户端请求时，触发云函数，并将处理结果作为 API 响应返回给客户端。

函数

逻辑意义上的一段代码片段，通过常见的入口文件包裹起来执行。函数是单一链路，并且无状态的，现在很多人认为，Serverless = FaaS + BaaS ，而 FaaS 则是无状态的函数，BaaS 解决带状态的服务。

什么叫带状态(stateful)

- 应用状态就是应用组件完成他们的工作（即执行任务）时所需的数据。从软件的架构、编码的范式到编程语言本身都离不开应用状态的参与，应用状态实质上说明了着怎样去管理一个应用的行为（任务，操作等）和状态（数据）。

什么叫带状态

- 全局变量
- 本地文件存储
- 长链接
- 。。。

函数组（服务）

多个函数聚合到一起的逻辑分组名，对应原有的应用概念。

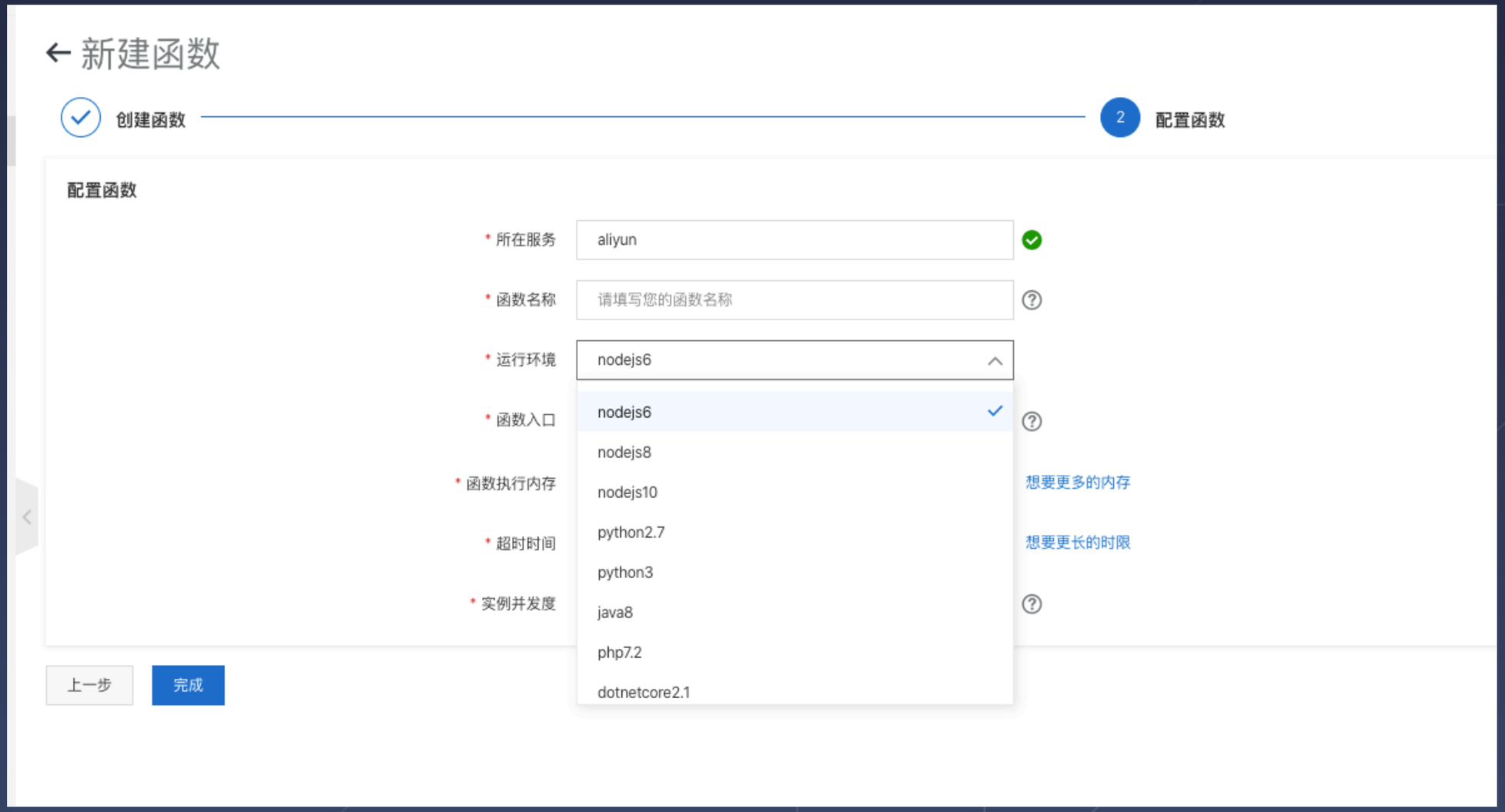
函数组


创建函数
2
配置函数
配置函数
* 所在服务
aliyun
* 函数名称
请填写您的函数名称
* 运行环境
nodejs6
* 函数入口
index.handler
* 函数执行内存
512MB
想要更多的内存
* 超时时间
60
秒
想要更长的时限
* 实例并发度
1
上一步
完成

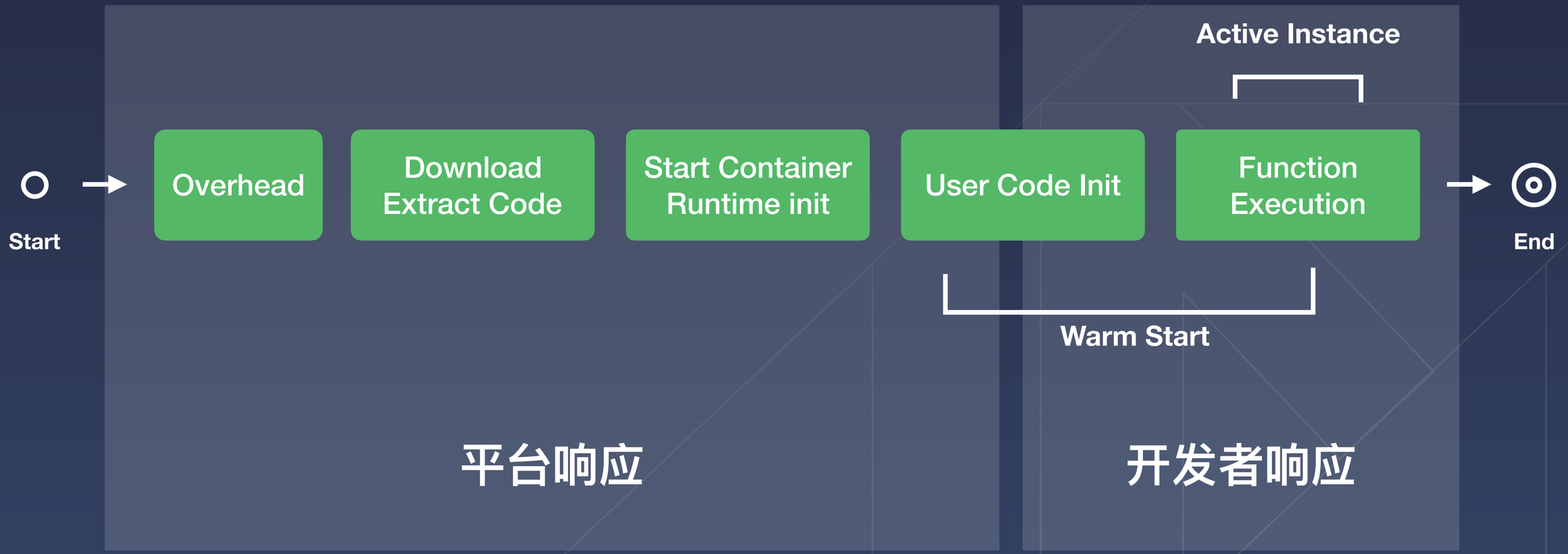
函数运行时

英文叫 Runtime，具体指执行函数的环境，其中包含了 Node.js 和一个对接平台的代码包，该代码包会实现对接平台的各种接口，处理异常，转发日志等能力。

函数运行时

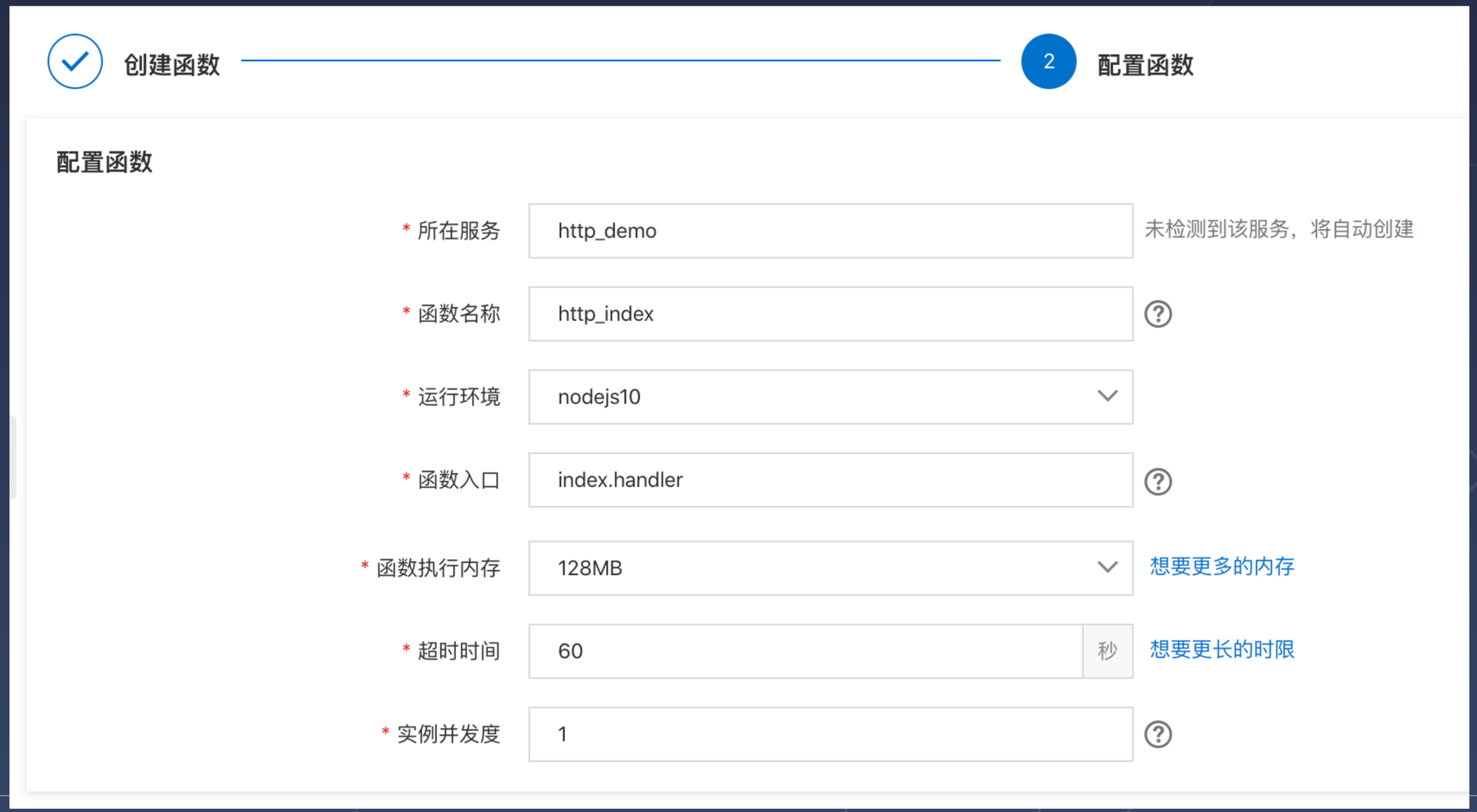

新建函数
创建函数
2
配置函数
配置函数
* 所在服务
aliyun
* 函数名称
请填写您的函数名称
* 运行环境
nodejs6
* 函数入口
* 函数执行内存
想要更多的内存
* 超时时间
想要更长的时限
* 实例并发度
nodejs6
nodejs8
nodejs10
python2.7
python3
java8
php7.2
dotnetcore2.1
上一步
完成

Code Start


Active Instance
Start
Overhead
Download
Extract Code
Start Container
Runtime init
User Code Init
Function
Execution
End
Warm Start
平台响应
开发者响应

快速上手函数

在阿里云写一个 http 接口

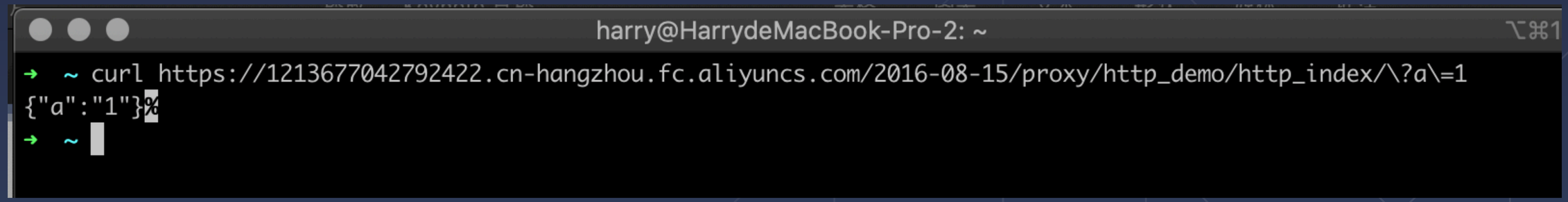


在阿里云写一个 http 接口

在阿里云写一个 http 接口

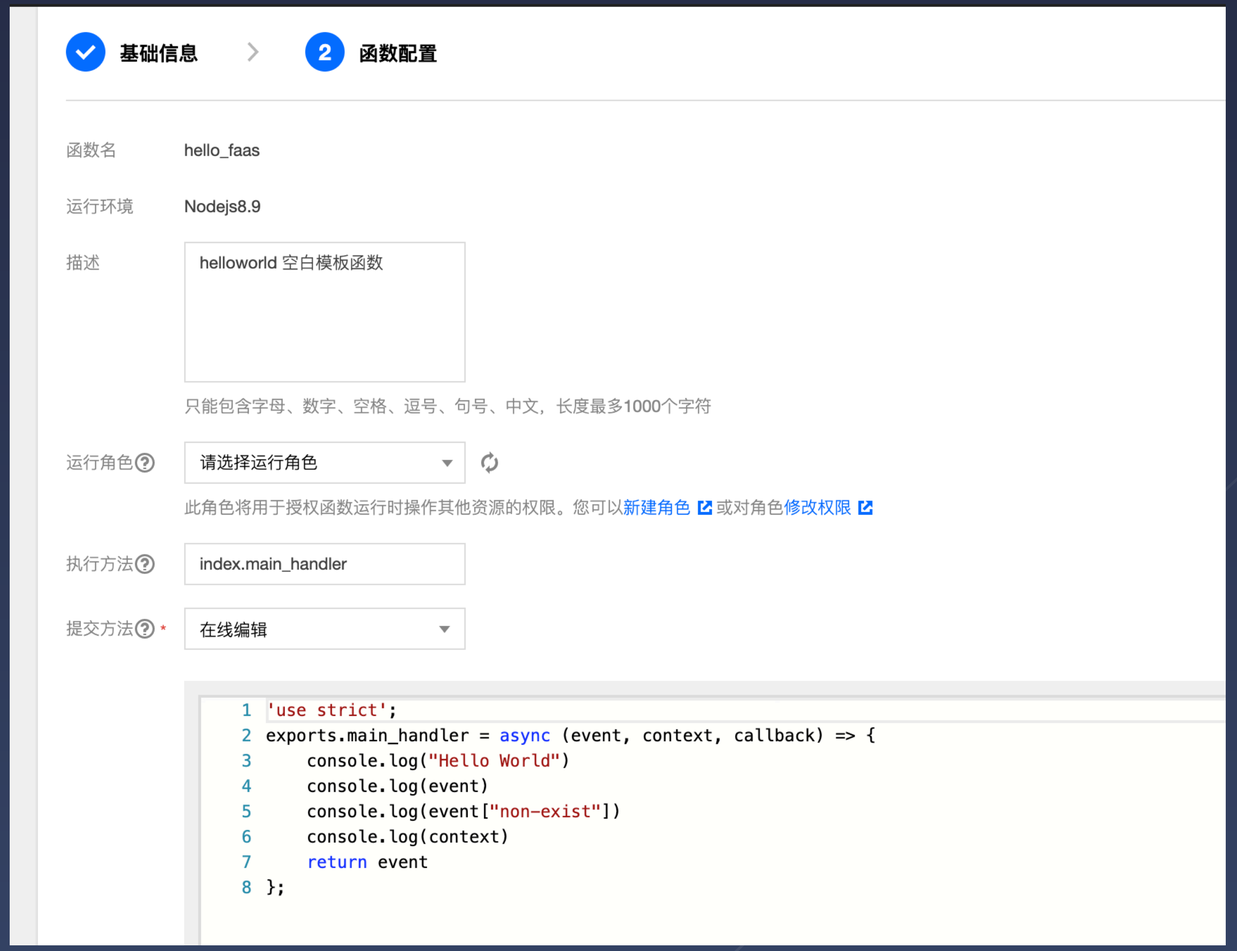
```
var getRawBody = require('raw-body');
var getFormBody = require('body/form');
var body = require('body');

exports.handler = function(req, resp, context) {
    var params = {
        path: req.path,
        queries: req.queries,
        headers: req.headers,
        method : req.method,
        requestURI : req.url,
        clientIP : req.clientIP,
    }
   resp.send(JSON.stringify(params.queries));
}
```

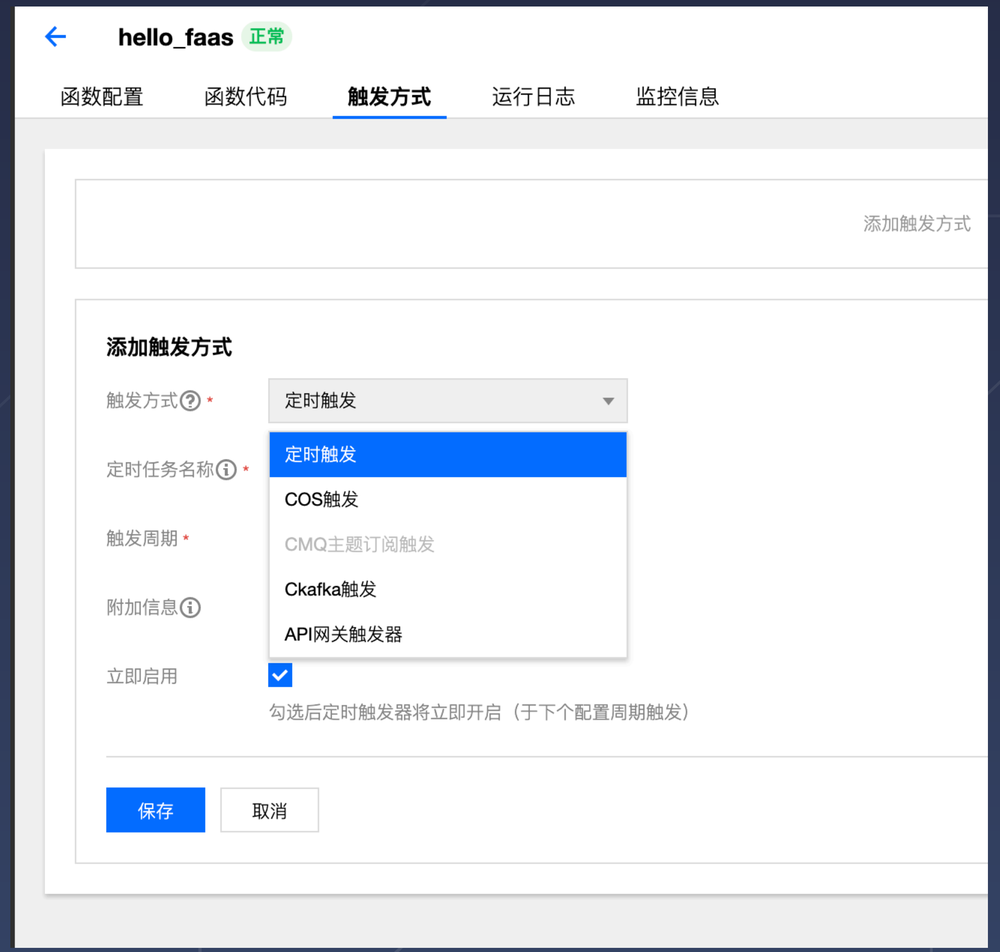
在阿里云写一个 http 接口



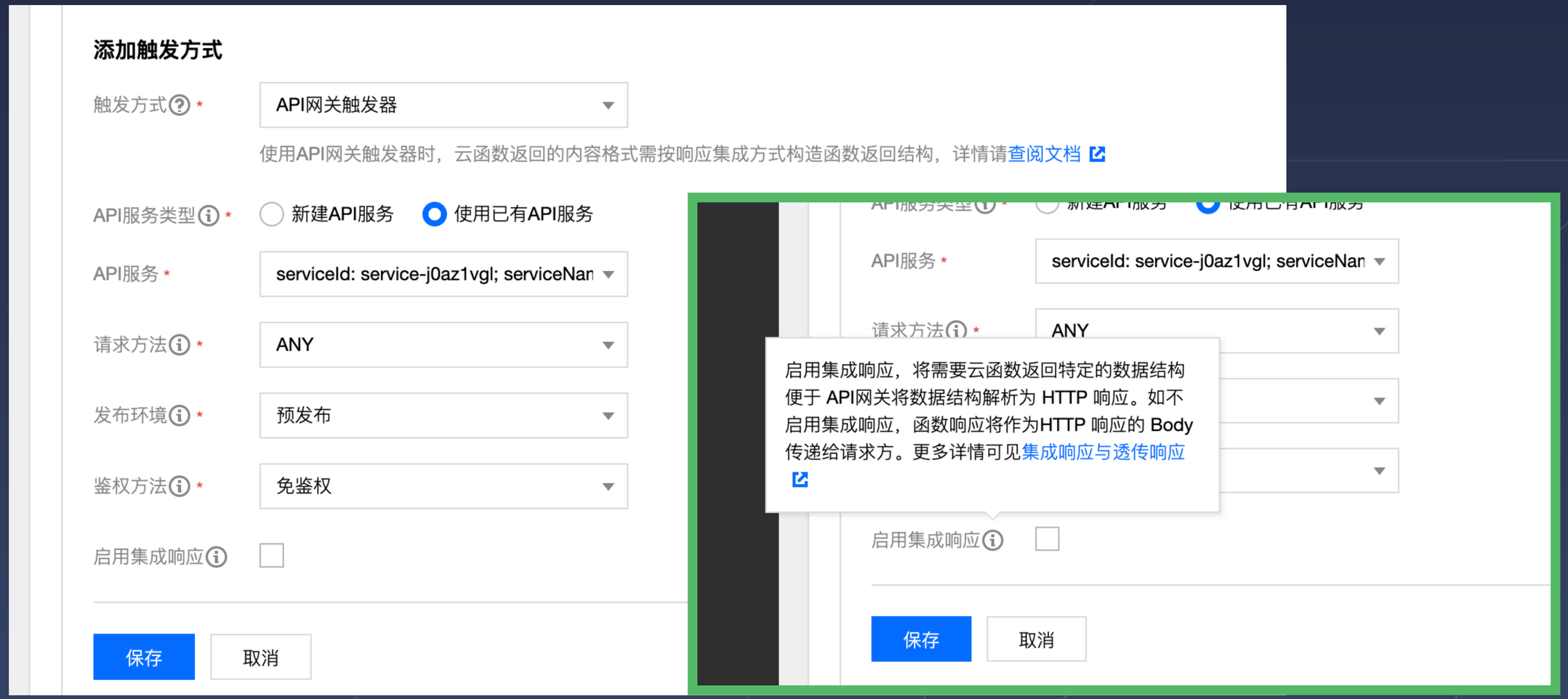
在腾讯云写一个 http 接口

在腾讯云写一个 http 接口

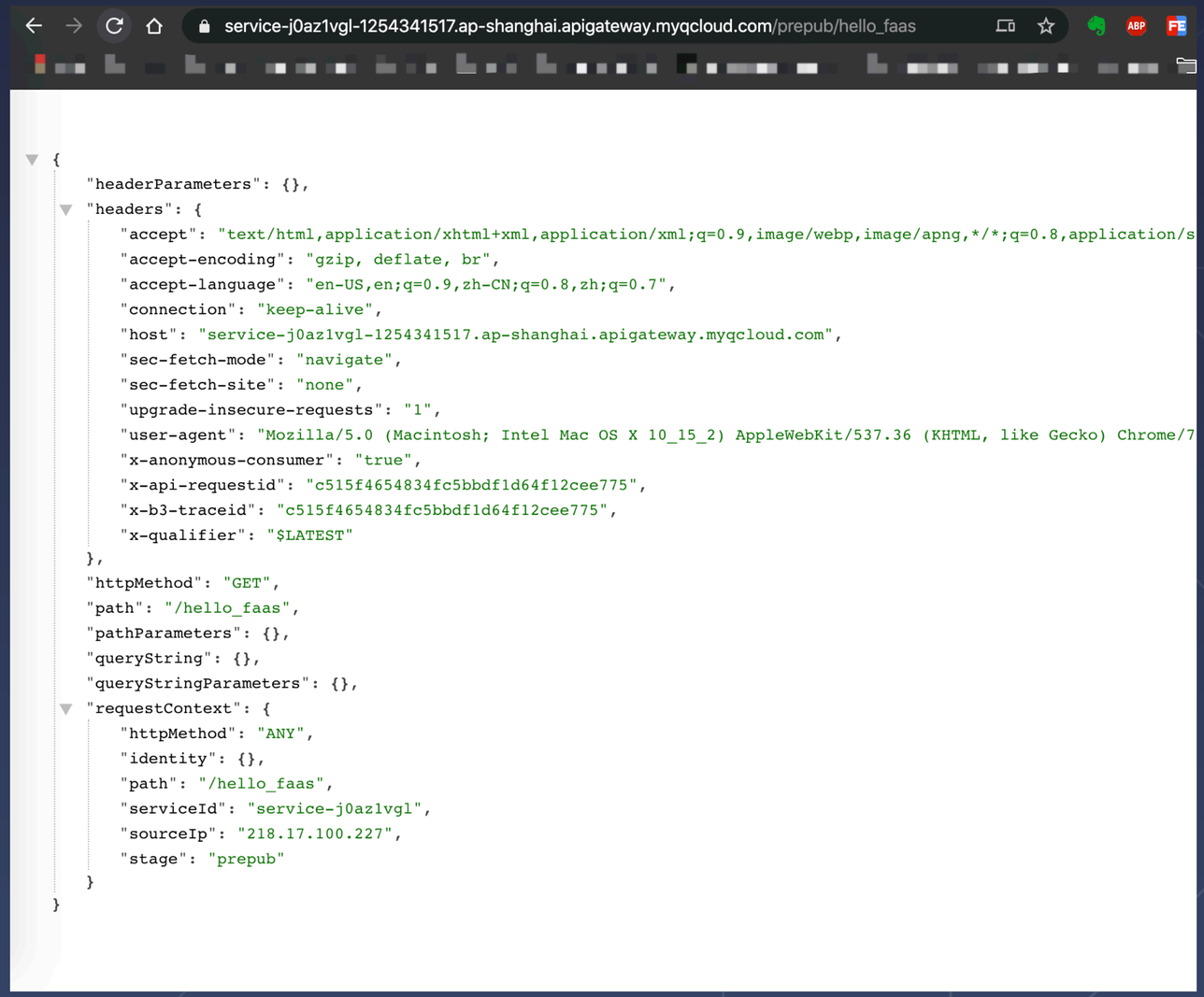




在腾讯云写一个 http 接口



在腾讯云写一个 http 接口



区别和问题

- 多平台出入参不同
- 参数没有定义，不知道结构，需要看文档
- 代码简单，依赖需要自己装

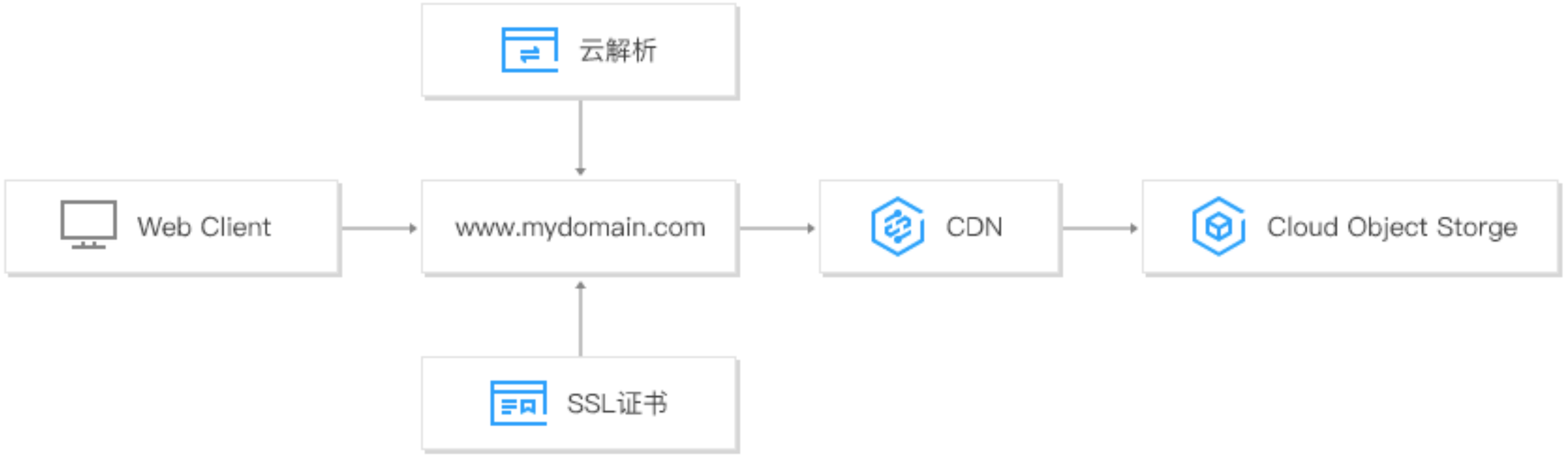
更多区别

AWS	GCF	FC	SCF	Ginkgo runtime
Env 隔离	x	x	√	x
两段式加载	?	x	√	√
hot reload	x	x	√	√
initialize	x	x	√	x
code 预热	x	x	√	x
stream res	x	x	√	x
timeout	x	x	x	x
Throttle	x	x	√	x
错误处理	√	x	√	√
错误栈美化	x	屏蔽错误栈	√	x
错误分类	√	x	√	√
同步返回	√	√ (非HTTP)	√	√
异步返回	√	√	x	√
健康检查	x	√	x	x

场景

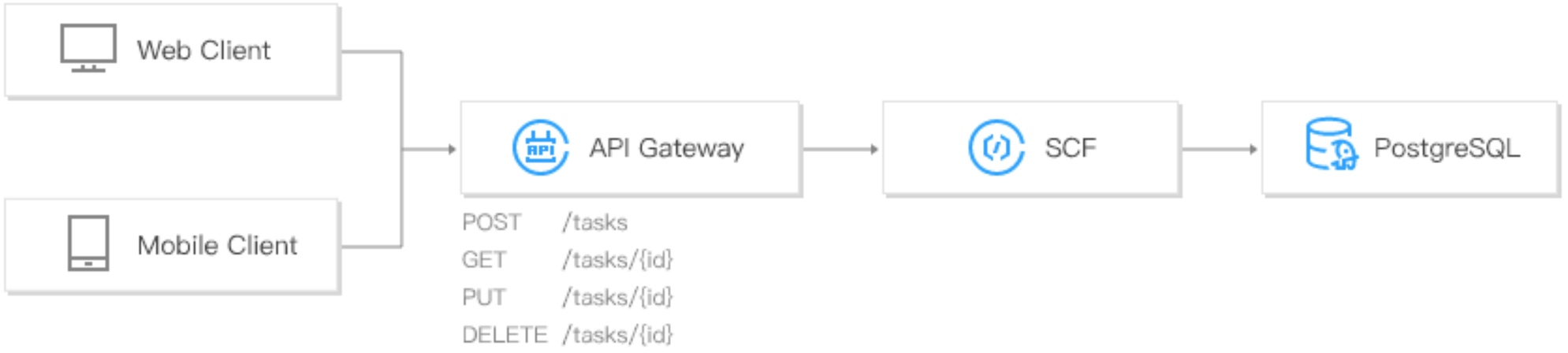
静态网站托管

通过结合云解析、SSL证书、CDN 和 COS 等组件，快速支持静态网站托管的场景。无需在多个产品控制台进行繁琐配置，您可以一键部署一个css/js/html的静态网站，支持各种框架（Hexo、Vuepress、Lavas、docsify、thumbsup等）。了解更多 >



构建 RESTful API

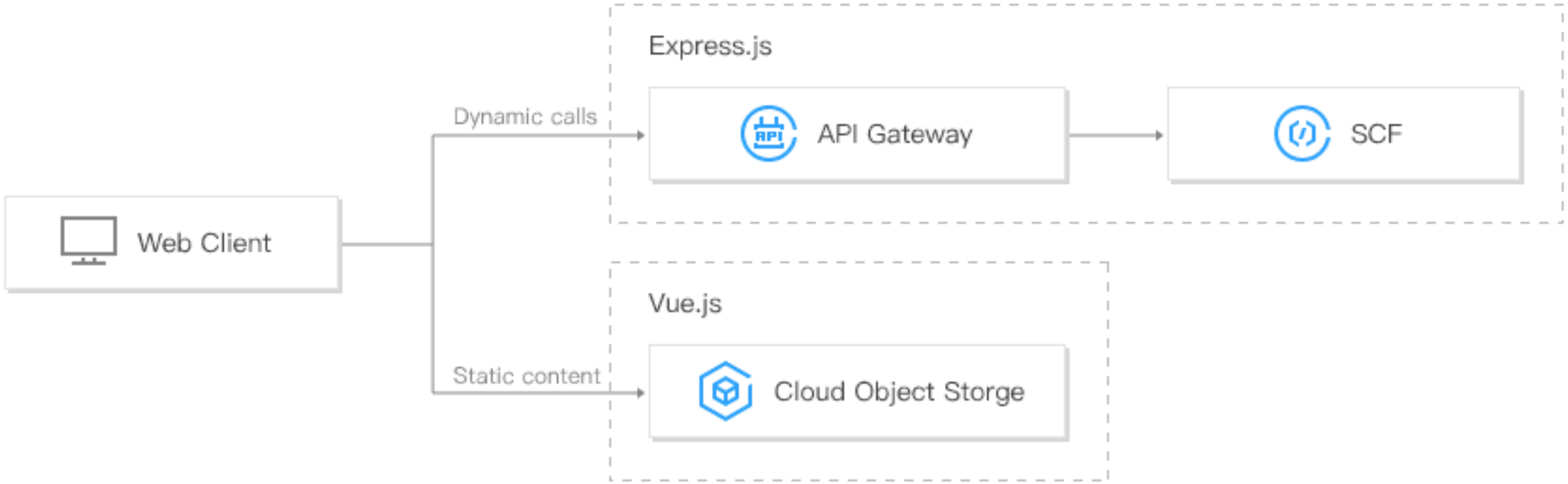
通过 SCF 云函数及 API 网关组件，轻松构建 RESTful API，极简配置，便捷部署，即可完成 API 的 CRUD 操作，适用于多种业务场景。了解更多 >



部署全栈应用

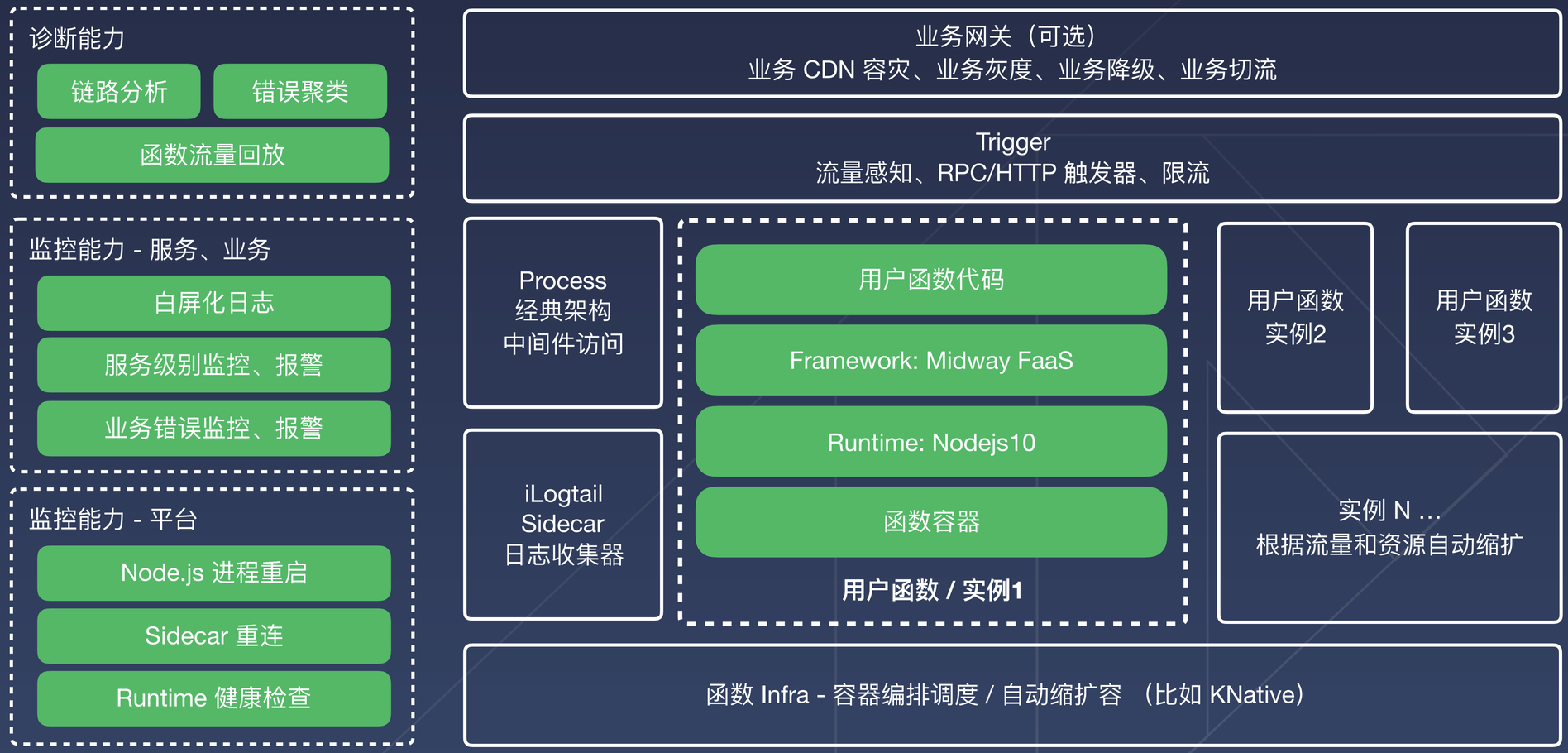
通过结合多个 Serverless Components，结合后端 API 与前端 Vue.js 结合等场景，帮助开发者更便捷地部署 Serverless 全栈 Web 应用。了解更多 >

- Serverless REST API：由无服务云函数 SCF 和 API Gateway 提供相关能力，帮助开发者架构自己的项目和路由。
- Serverless Vue.js 站点：由对象存储 COS 提供相关存储能力。通过后端 API 传递到前端，并使用 Vue.js 做相关渲染。



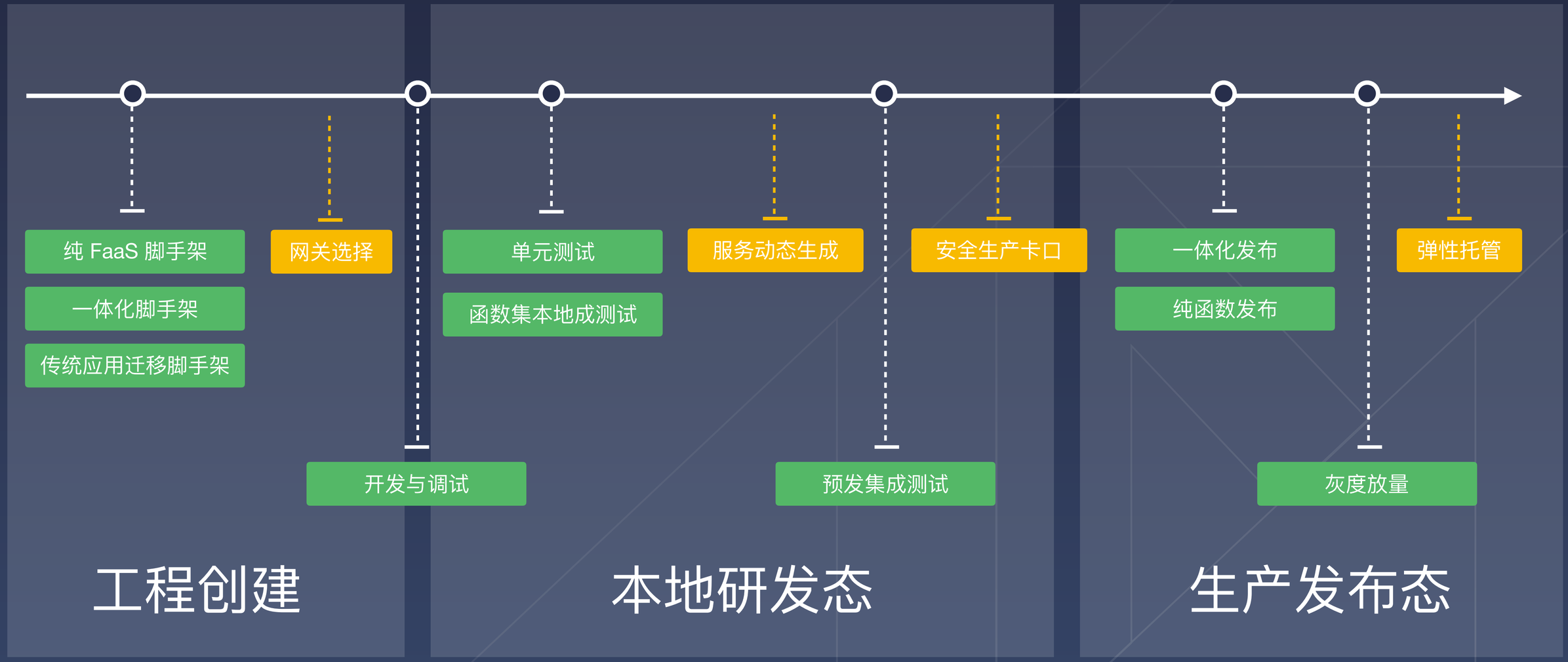
第二章节
企业级 Serverless 开发模式

运行态系统架构 / 函数架构



阿里 FaaS 技术栈运行时架构

阿里 FaaS 技术栈工程架构



工程创建和本地研发架构

研发模式升级

研发模式升级

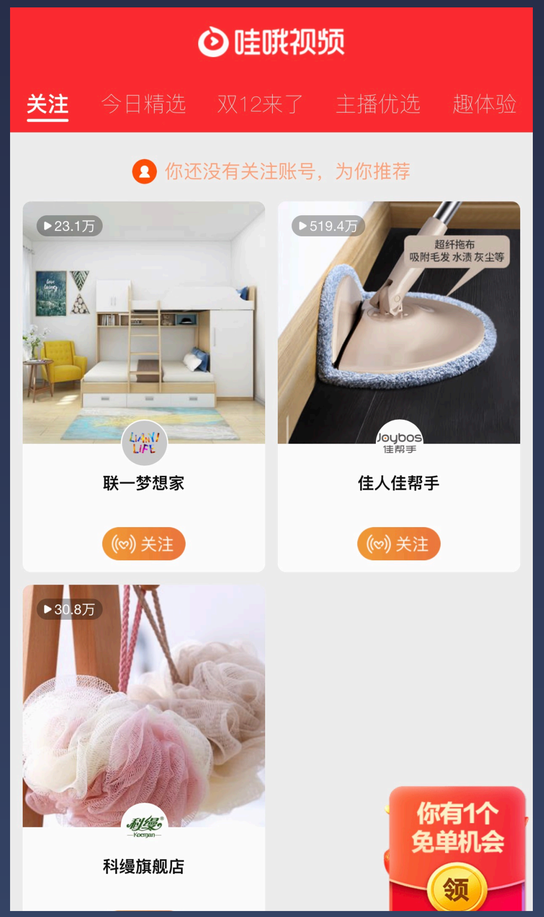
在整个升级过程中，针对研发过程中的各类**相似问题**，提炼出一些**共性的解决方案**，这就是我们的研发模式。

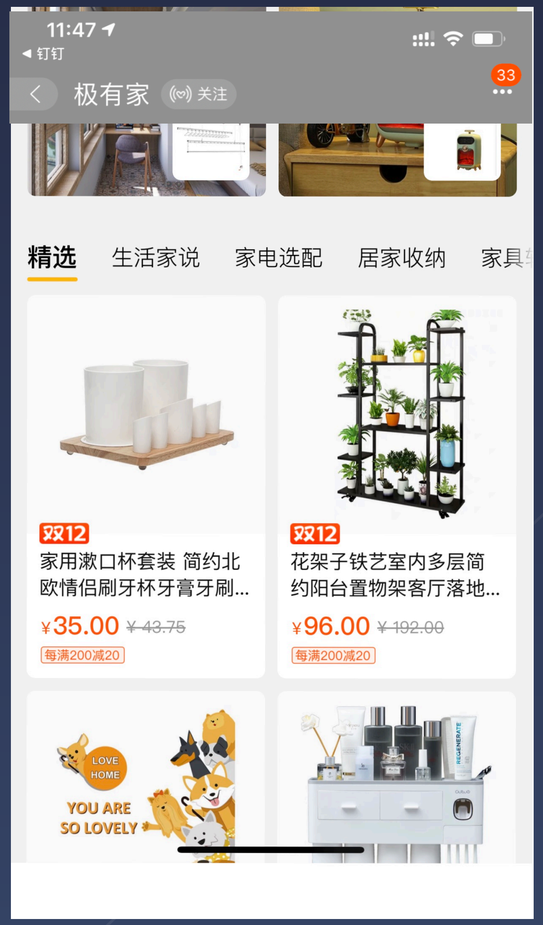
经历一年的研发模式升级

Now

完成了淘宝和飞猪两大 BU 导购链路，以及一些内部系统的升级。
经历了双十一和双十二大促的考验，承载了千万级的流量。

阿里的 Serverless



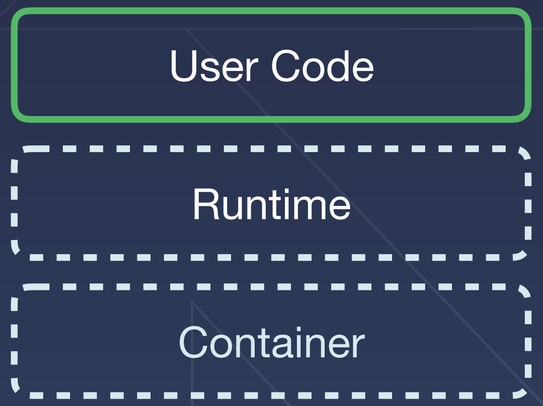


研发模式升级

主要是两部分，第一部分升级我们的**框架、工具链，技术栈**，
第二部分升级我们的**前端同学的应用开发意识**。

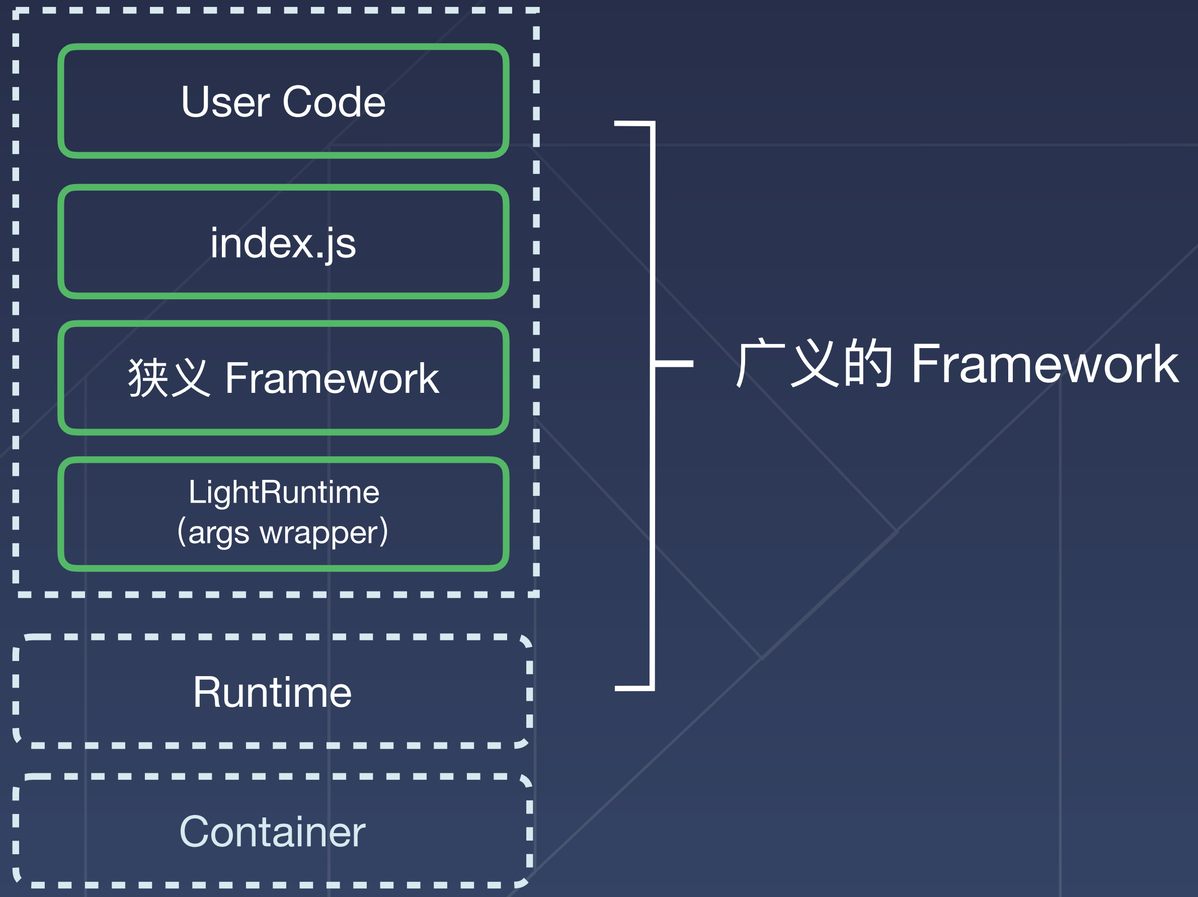
研发模式升级

升级我们的开发模型



研发模式升级

升级我们的开发模型



设计 Serverless Framework

1 防厂商锁定

Avoid Vendor Lock-in

现在的 FaaS 还没有一个固定的标准，使用时会担心固化在特定平台，后续无法迁移，我们希望思考和解决这个问题。

2 灵活性

Flexibility

函数框架支持灵活的部署模式，可以在垂直和水平两方面进行按需拆分和组合。

3 开发效率

Development Efficiency

提升开发效率，在快速迭代业务的同时，尽可能标准化，易解耦，可扩展和复用。

4 生命周期扩展

Lifecycle Extension

在平台运行时和用户代码之间，设计一层通用的运行时扩展能力，在统计埋点，提前加载模块等类似场景上提供支持。

设计 Serverless Framework

1 防厂商锁定

Avoid Vendor Lock-in

现在的 FaaS 还没有一个固定的标准，使用时会担心固化在特定平台，后续无法迁移，我们希望思考和解决这个问题。

2 灵活性

Flexibility

函数框架支持灵活的部署模式，可以在垂直和水平两方面进行按需拆分和组合。

3 开发效率

Development Efficiency

提升开发效率，在快速迭代业务的同时，尽可能标准化，易解耦，可扩展和复用。

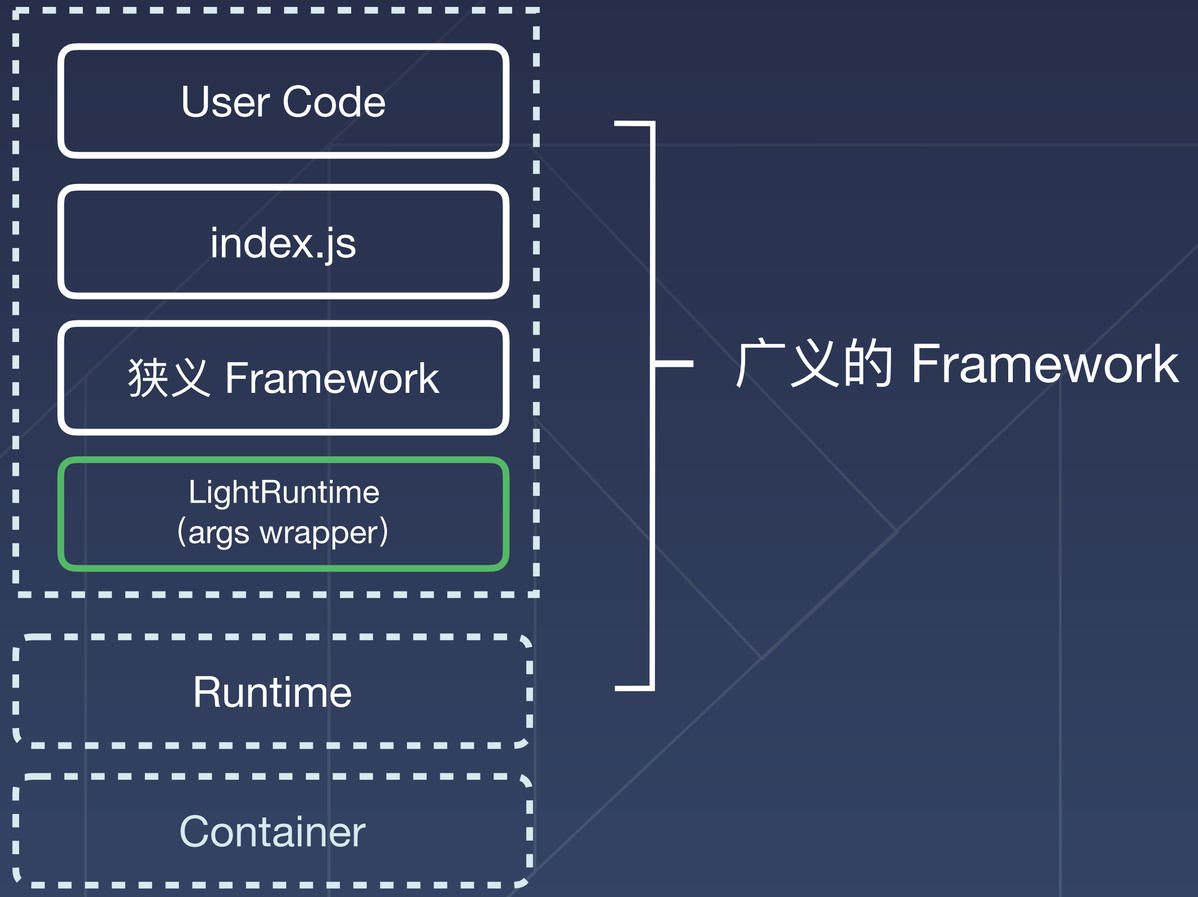
4 生命周期扩展

Lifecycle Extension

在平台运行时和用户代码之间，设计一层通用的运行时扩展能力，在统计埋点，提前加载模块等类似场景上提供支持。

研发模式升级

升级我们的开发模型

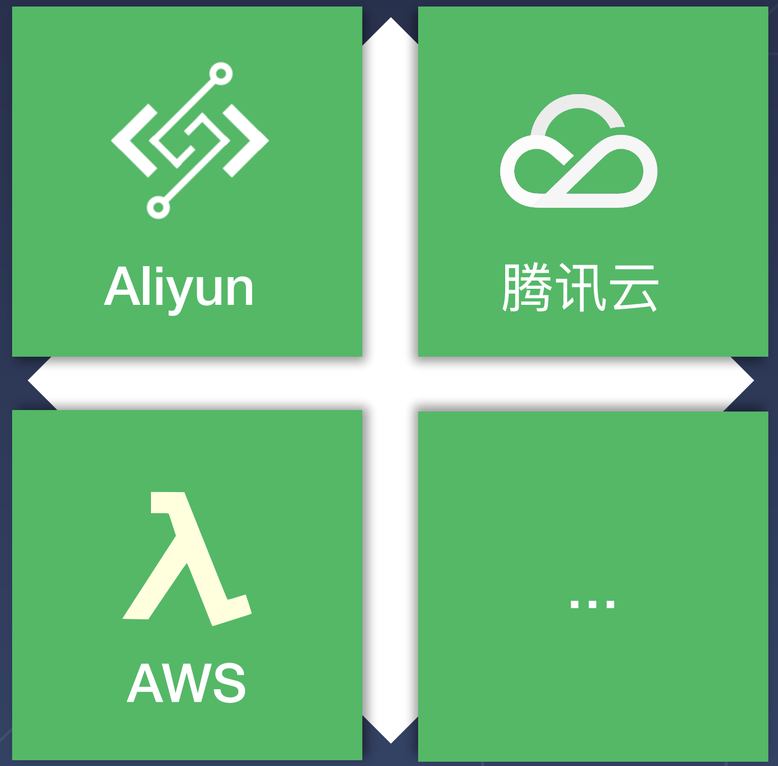


1 防厂商锁定 Avoid Vendor Lock-in

req, res, context

event, context, callback

event, context



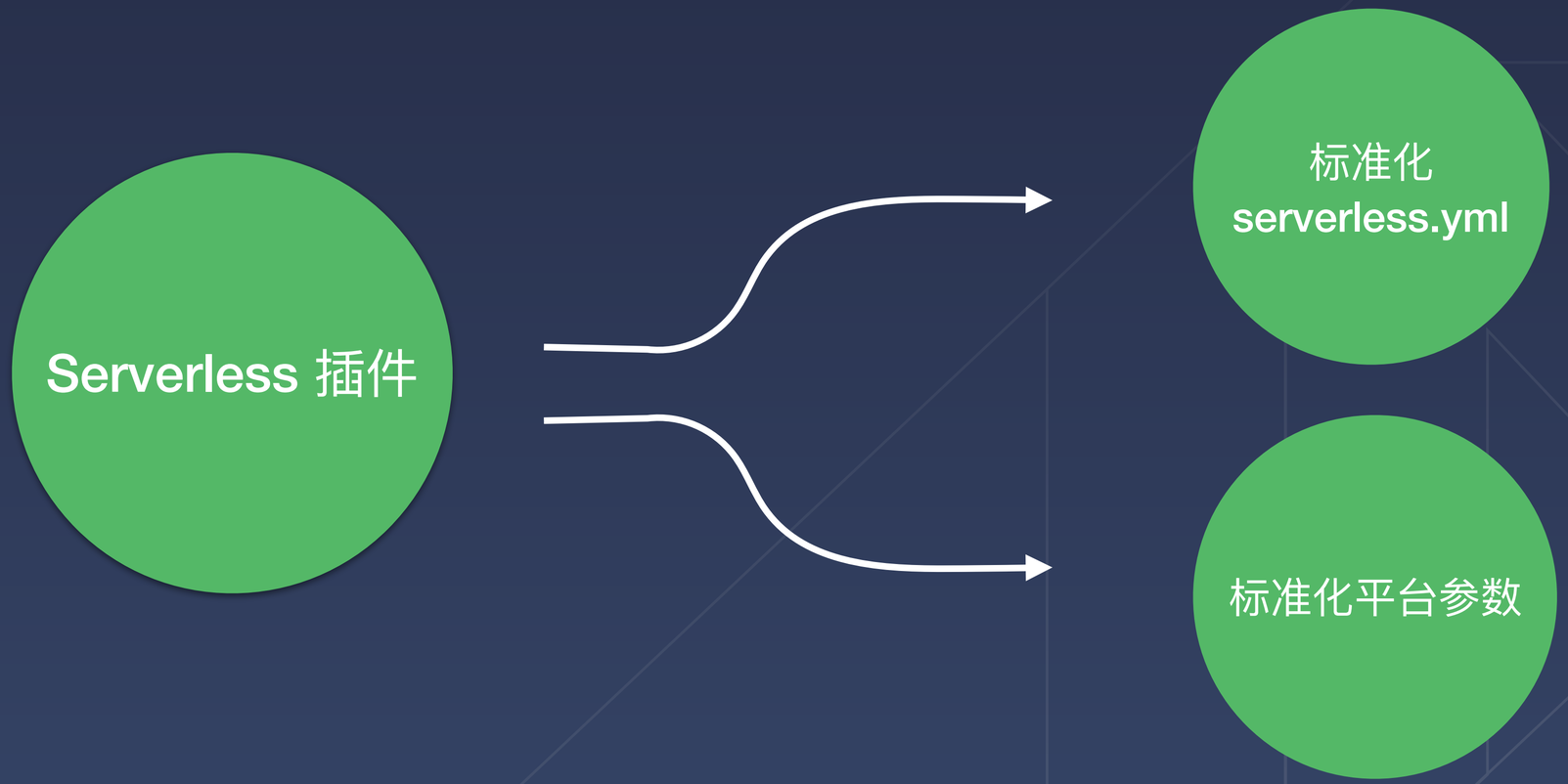
event, context, callback

不同的云厂商有着不一致的标准。

1. 通过定义，一定程度上解决了跨平台问题
2. 通过插件扩展，多平台的部署支持

```
# Step 1. Install serverless globally
$ npm install serverless -g

# Step 2. Login to your Serverless account
$ serverless login

# Step 3. Create a serverless function
$ serverless create --template hello-world

# Step 4. Deploy to cloud provider
$ serverless deploy

# Your Function is deployed!
$ http://xyz.amazonaws.com/hello-world
```

1 防厂商锁定 Avoid Vendor Lock-in



在 Serverless 的插件基础上，进一步标准化和扩展。

```yaml
service: serverless-midway-test

provider:
  name: aliyun

functions:
  index:
    handler: index.handler
    events:
      - http:
          path: '/'
          method: get
  hello:
    handler: hello.handler
    events:
      - http:
          path: '/hello'
          method: get

plugins:
  - test

aggregation:
  index:
    deployOrigin: false
    handler: aggregation.handler
    functions:
      - index
      - hello

package:
  include:
  exclude:
  excludeDevDependencies: false
  artifact: my-artifact.zip
```

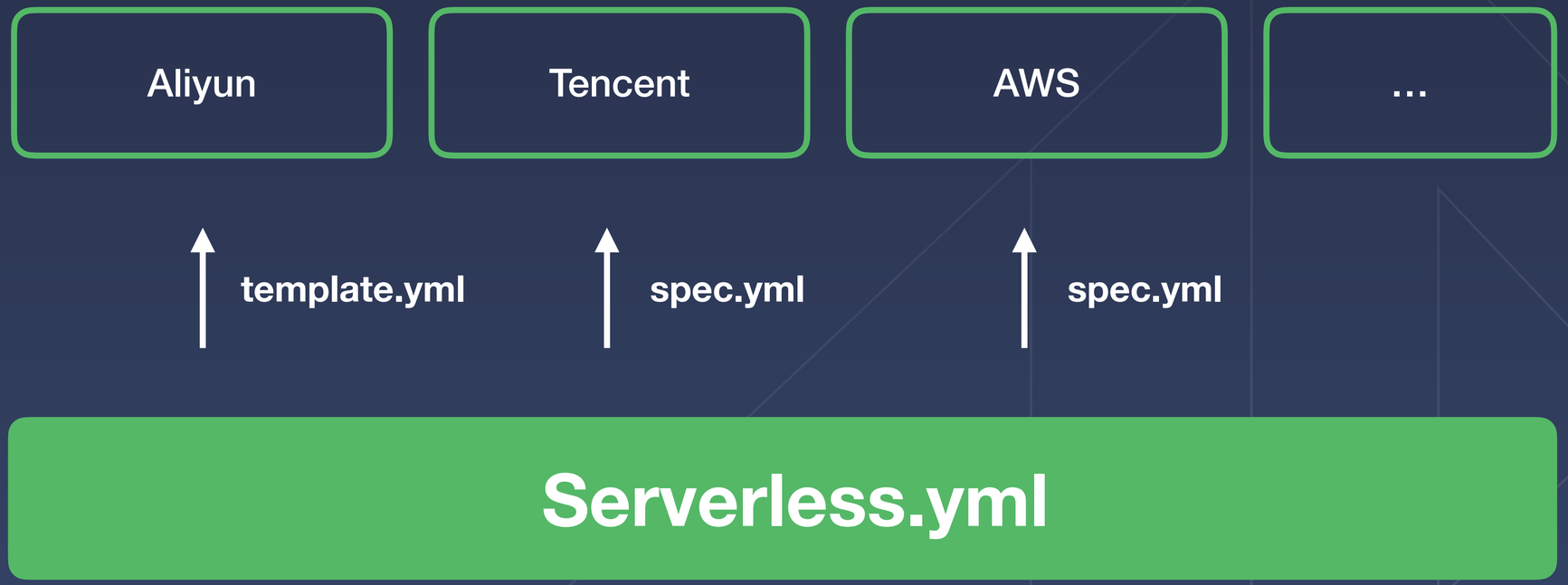
标准化 serverless.yml

以前，我们使用不同平台的 serverless.yml 文件来部署，里面的格式和内容是由云厂商插件来定义的。这些云厂商提供的能力有一定相似性，我们希望尽可能用同一份 yml 文件来定义这些内容。

用一份定义生成不同平台的 spec 的形式，并在其中尽可能固化相同的字段，并且扩展我们自己的字段。

标准化 serverless.yml

以前，我们使用不同平台的 serverless.yml 文件来部署，里面的格式和内容是由云厂商插件来定义的。这些云厂商提供的能力有一定相似性，我们希望尽可能用同一份 yml 文件来定义这些内容。



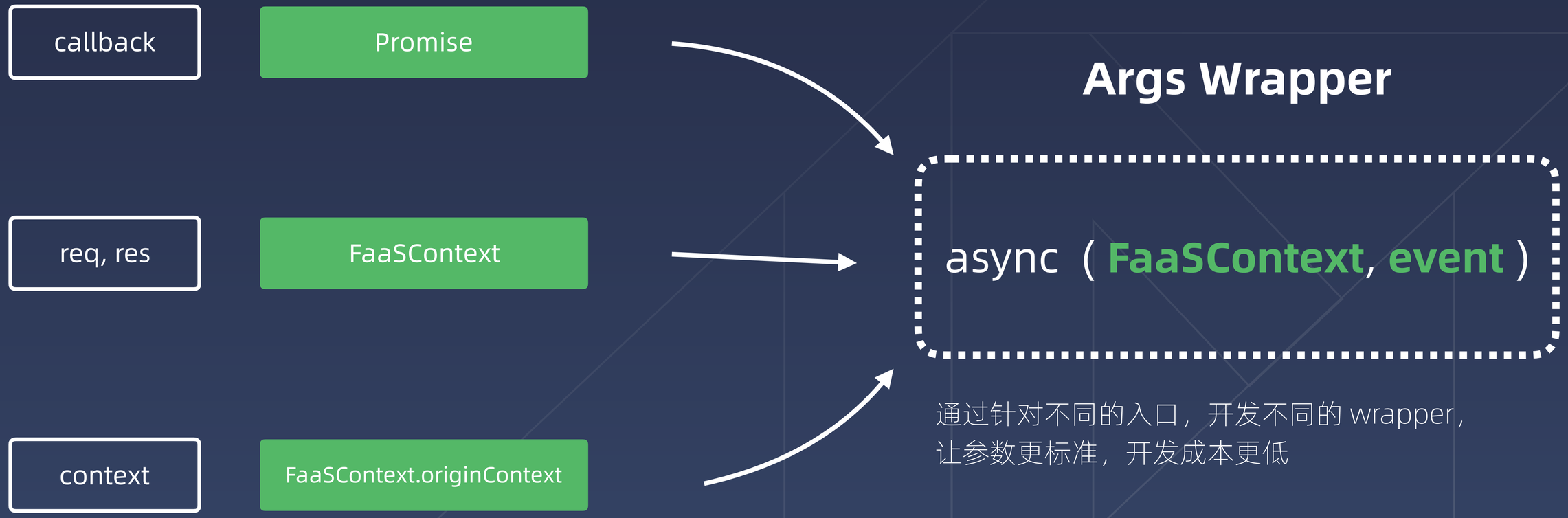
标准化平台出入参

callback

req, res

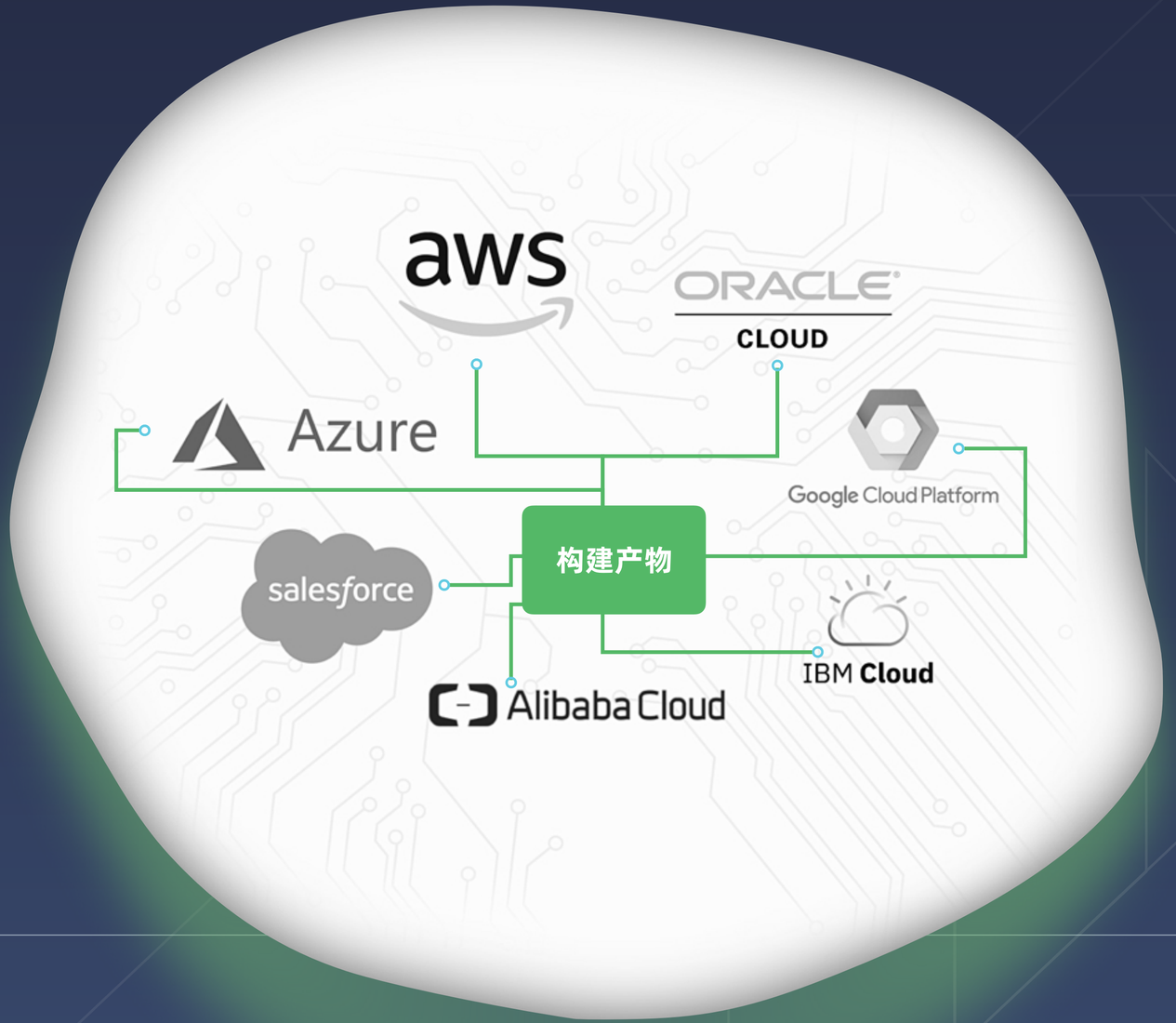
context

标准化平台出入参


callback
Promise
req, res
FaaSContext
context
FaaSContext.originContext
Args Wrapper
async（ FaaSContext, event ）
通过针对不同的入口，开发不同的 wrapper，
让参数更标准，开发成本更低

标准化平台出入参

标准化平台出入参


aws
ORACLE
CLOUD
Azure
Google Cloud Platform
构建产物
salesforce
IBM Cloud
Alibaba Cloud

产出标准定义

通过标准化不同平台触发器，我们产出了一套可沉淀，可校验的 interface，帮助我们在支持多平台时走的更快更远。

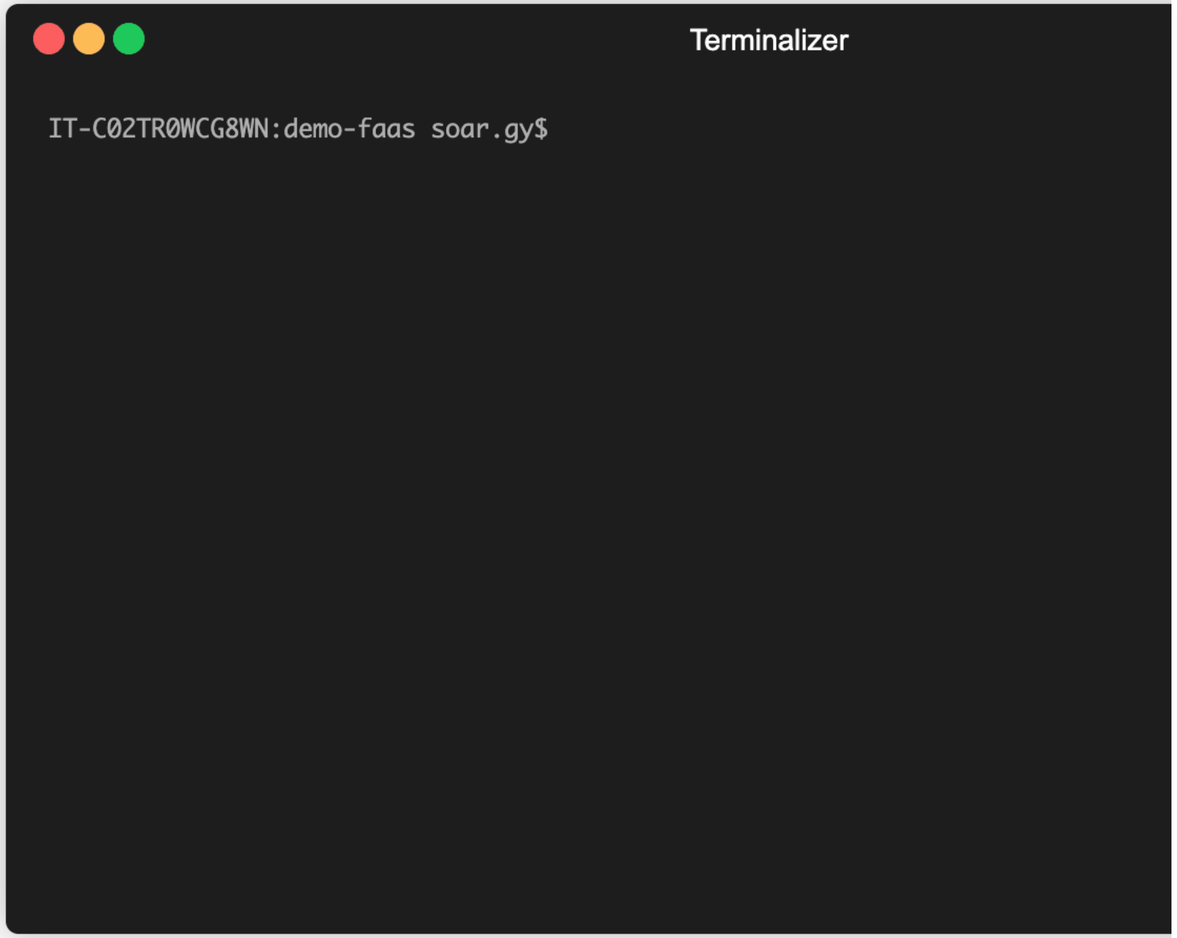
1. 基于开源的 Serverless 工具开发，复用现有 Serverless 插件能力和社区生态

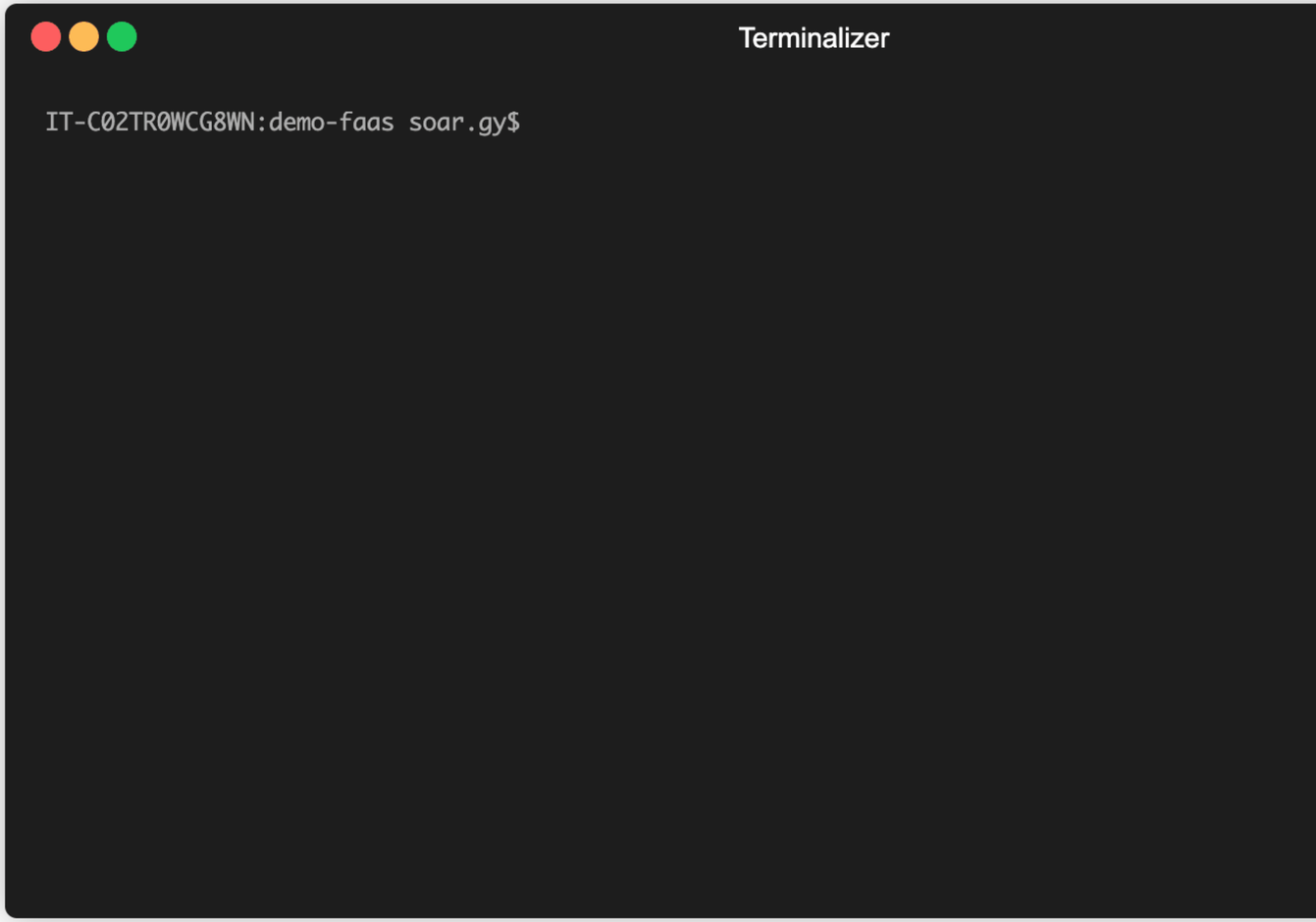
2. 每个平台，相同的地方保持一致，不同的触发器上独立扩展，有完整定义

3. 扩展开发、调试、部署的自有能力

```typescript
export interface FCServiceSpec {
  [propertyName: string]: FCServiceType | FCFunctionSpec | FCServiceProperties;
}

export interface MountPointSpec {
  ServerAddr?: string;
  MountDir?: string;
}

export interface FCFunctionSpec {
  Type: FCFunctionType;
  Properties: {
    Handler: string;
    Runtime: string;
    CodeUri: string;
    Initializer?: string;
    Description?: string;
    MemorySize?: number;
    Timeout?: number;
    InitializationTimeout?: number;
    EnvironmentVariables?: object;
  };
  Events?: {
    [eventName: string]: FCHTTPEvent | FCTimerEvent;
  };
}

export type HTTPEventType = 'GET' | 'POST' | 'PUT' | 'DELETE' | 'HEAD';

export interface FCHTTPEvent {
  Type: 'HTTP';
  Properties: {
    AuthType?: 'ANONYMOUS' | 'FUNCTION';
    Methods?: HTTPEventType[];
  };
}
```

发布到多个平台


Terminalizer
IT-C02TR0WCG8WN:demo-faas soar.gy$


Terminalizer
IT-C02TR0WCG8WN:demo-faas soar.gy$

设计 Serverless Framework

1 防厂商锁定

Avoid Vendor Lock-in

现在的 FaaS 还没有一个固定的标准，使用时会担心固化在特定平台，后续无法迁移，我们希望思考和解决这个问题。

2 灵活性

Flexibility

函数框架支持灵活的部署模式，可以在垂直和水平两方面进行按需拆分和组合。

3 开发效率

Development Efficiency

提升开发效率，在快速迭代业务的同时，尽可能标准化，易解耦，可扩展和复用。

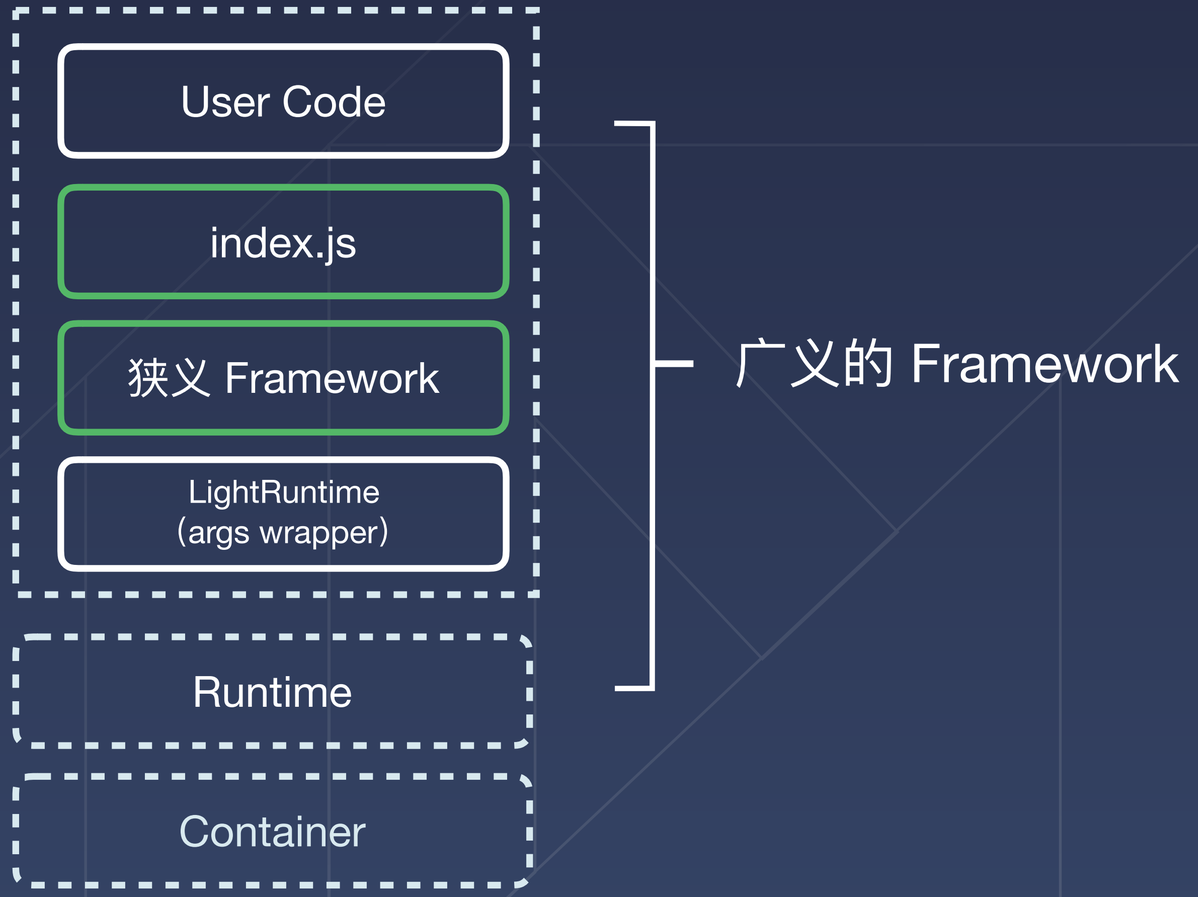
4 生命周期扩展

Lifecycle Extension

在平台运行时和用户代码之间，设计一层通用的运行时扩展能力，在统计埋点，提前加载模块等类似场景上提供支持。

研发模式升级

升级我们的开发模型



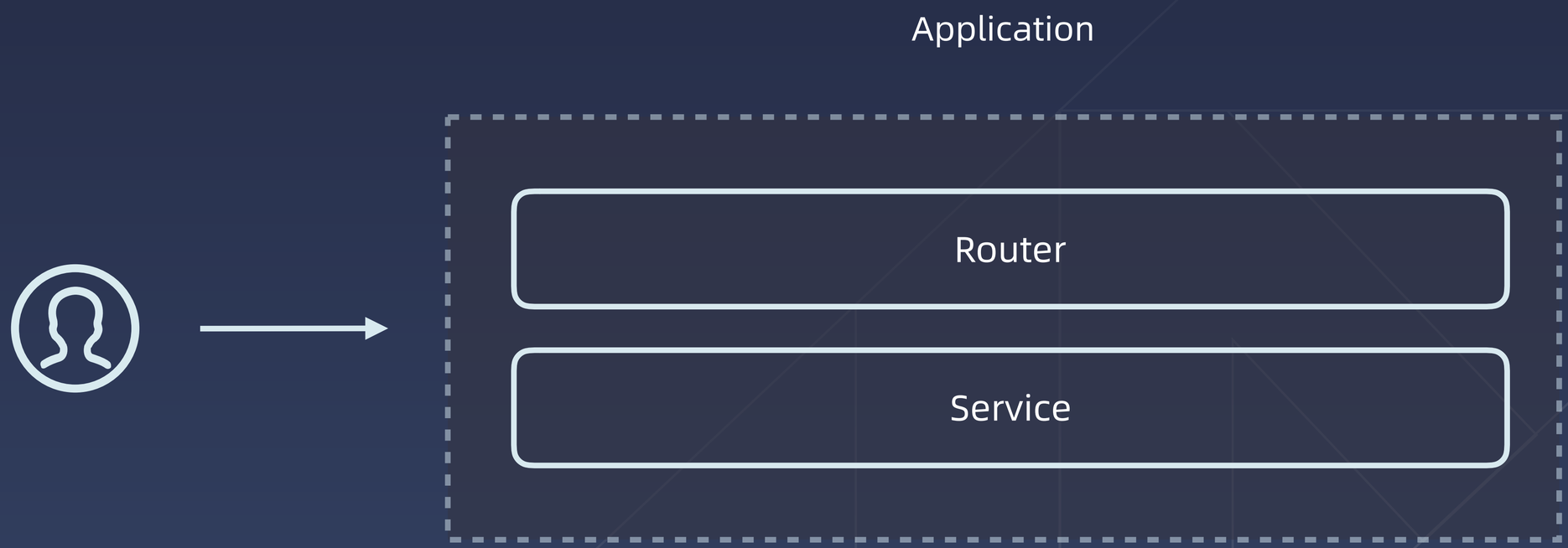
2 灵活性 Flexibility

所谓的 Flexibility，是我们对未来的一种美好愿景，这不仅仅体现在开发上，也体现在部署上。

举例：传统 Web 应用场景

```
router.get('/', 'index.main');
router.post('/user', 'user.create');
router.get('/api/user', 'api.findUser');
router.get('/api/user/list', 'api.listUser');
router.post('/api/user', 'api.createUser');
```

从应用框架到函数

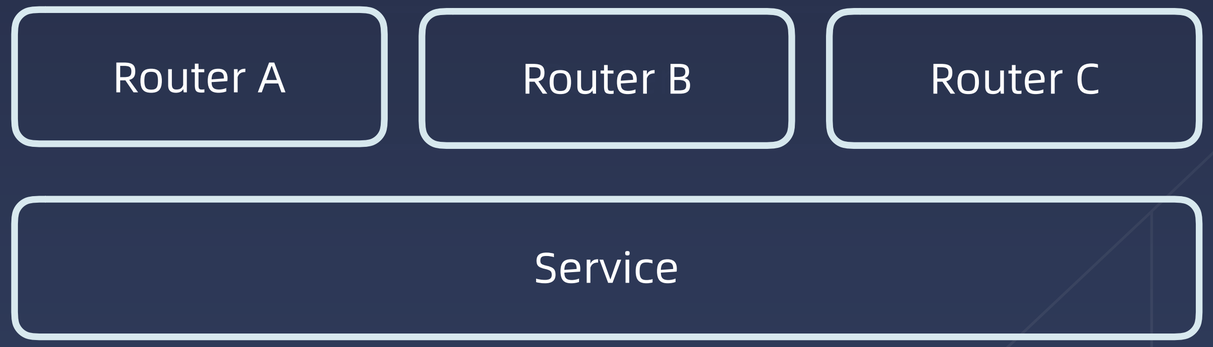


从应用框架到函数

Router

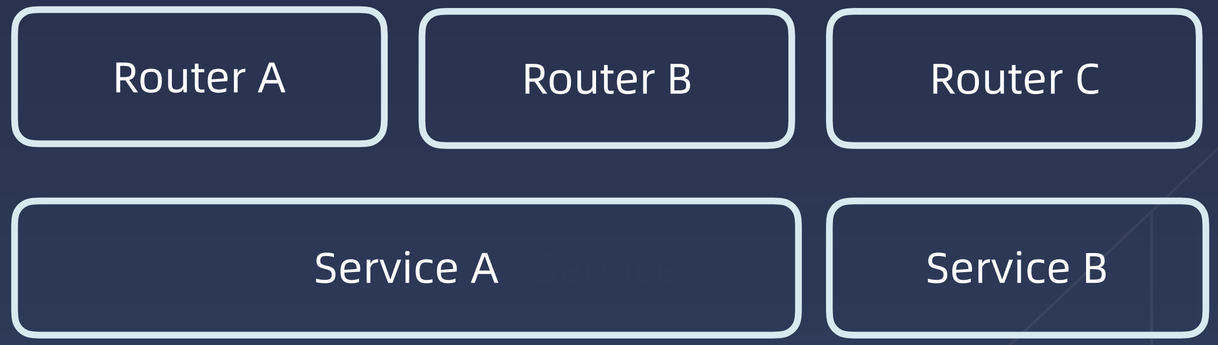
Service

从应用框架到函数



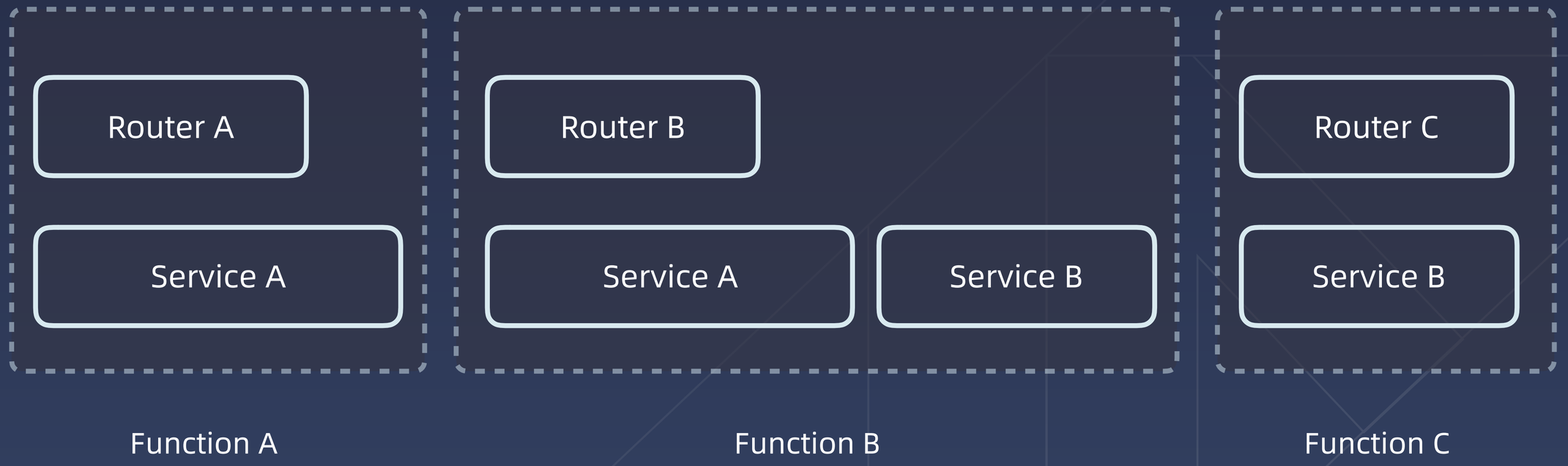
入口垂直拆分

从应用框架到函数

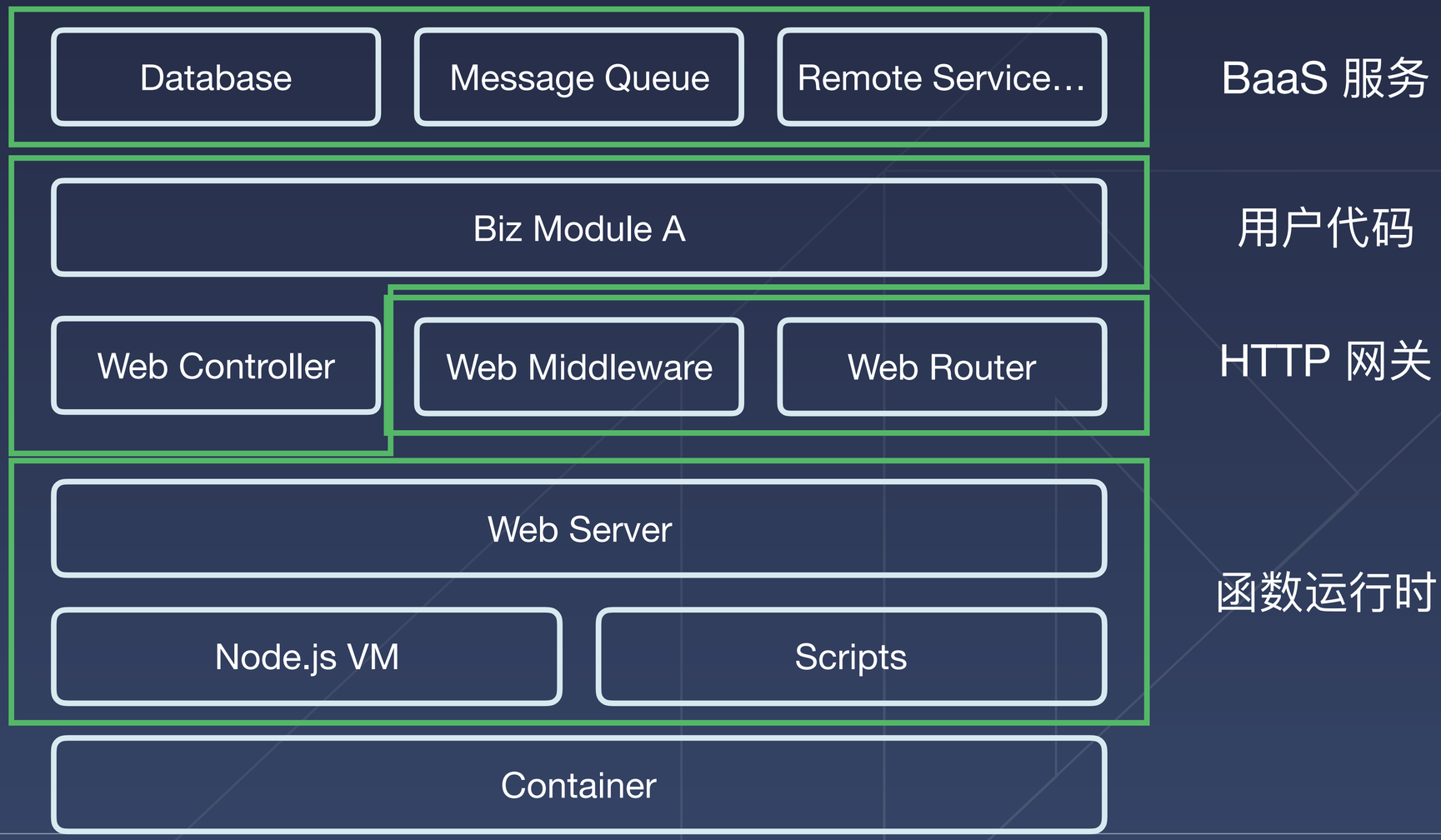


逻辑水平拆分

从应用框架到函数


Router A
Service A
Router B
Service A
Service B
Router C
Service B
Function A
Function B
Function C

研发模式升级


Database
Message Queue
Remote Service…
BaaS 服务
Biz Module A
用户代码
Web Controller
Web Middleware
Web Router
HTTP 网关
Web Server
Node.js VM
Scripts
函数运行时
Container

举例：传统 Web 应用场景

```
router.get('/', 'index.main');
router.post('/user', 'user.create');
router.get('/api/user', 'api.findUser');
router.get('/api/user/list', 'api.listUser');
router.post('/api/user', 'api.createUser');
```

1. 需要开发多个函数
2. 每个函数可以绑定到一个或者多个路由
3. 每个路由的调用频次不同

按传统模型，这样的多个函数代码将部署到多个容器中。

平台限制

```
Serverless: ErrorCode: LimitExceeded.Function RequestId: 5788c22f-3e1a-43da-b769-b9aae7d809f4

  Error ---------------------------------------------------

  Error: 函数个数已达限制
      at HttpConnection.doRequest (/Users/lxxyx/Desktop/ali/playground/demo-faas/node_modules/tencentcloud-sdk-nodejs/tencentcloud/common/abstr
act_client.js:98:45)
      at Request._callback (/Users/lxxyx/Desktop/ali/playground/demo-faas/node_modules/tencentcloud-sdk-nodejs/tencentcloud/common/http/http_co
nnection.js:28:13)
      at Request.self.callback (/Users/lxxyx/Desktop/ali/playground/demo-faas/node_modules/request/request.js:185:22)
      at Request.emit (events.js:198:13)
      at Request.EventEmitter.emit (domain.js:448:20)
      at Request.<anonymous> (/Users/lxxyx/Desktop/ali/playground/demo-faas/node_modules/request/request.js:1161:10)
      at Request.emit (events.js:198:13)
      at Request.EventEmitter.emit (domain.js:448:20)
      at IncomingMessage.<anonymous> (/Users/lxxyx/Desktop/ali/playground/demo-faas/node_modules/request/request.js:1083:12)
      at Object.onceWrapper (events.js:286:20)
      at IncomingMessage.emit (events.js:203:15)
      at IncomingMessage.EventEmitter.emit (domain.js:448:20)
      at endReadableNT (_stream_readable.js:1143:12)
      at process._tickCallback (internal/process/next_tick.js:63:19)

    For debugging logs, run again after setting the "SLS_DEBUG=*" environment variable.

  Get Support --------------------------------------------
     Docs:          docs.serverless.com
     Bugs:          github.com/serverless/serverless/issues
     Issues:        forum.serverless.com
```

函数的计费模型

总价 = 调用次数 + 资源消耗（CU)

资源的消耗跟 CPU，内存密切相关，很难控制。

核心诉求：在一定程度上减少总成本。

灵活性的诉求

提升资源利用率

每个函数的资源利用率不高，每次请求都会触发冷启动，影响体验。

降低部署相对成本

部署到云平台，云平台减少内部调度、发布的成本，进而影响总成本。

未来的扩展性

增加函数内部互调、热点函数迁移，代码逻辑复用的可能性。

2 灵活性 Flexibility

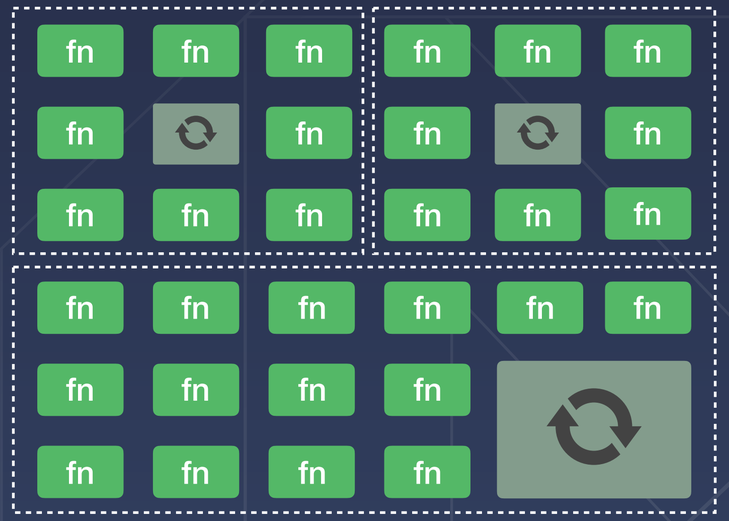
传统 FaaS 部署模型

所谓的 Flexibility，是我们对未来的一种美好愿景，这不仅仅体现在开发上，也在部署上提现。

2 灵活性 Flexibility

传统 FaaS 部署模型

理想：资源复用型 FaaS 部署模型

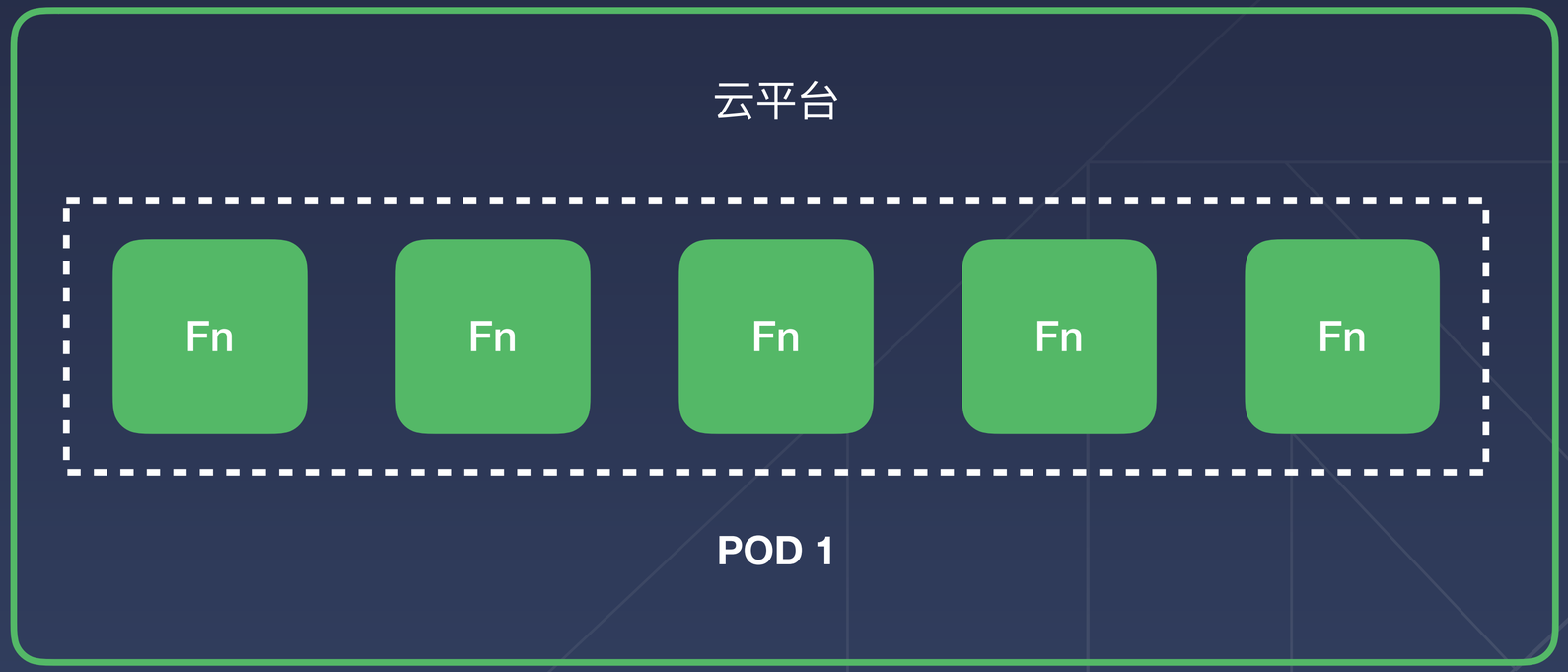


所谓的 Flexibility，是我们对未来的一种美好愿景，这不仅仅体现在开发上，也在部署上提现。

函数横向聚合能力

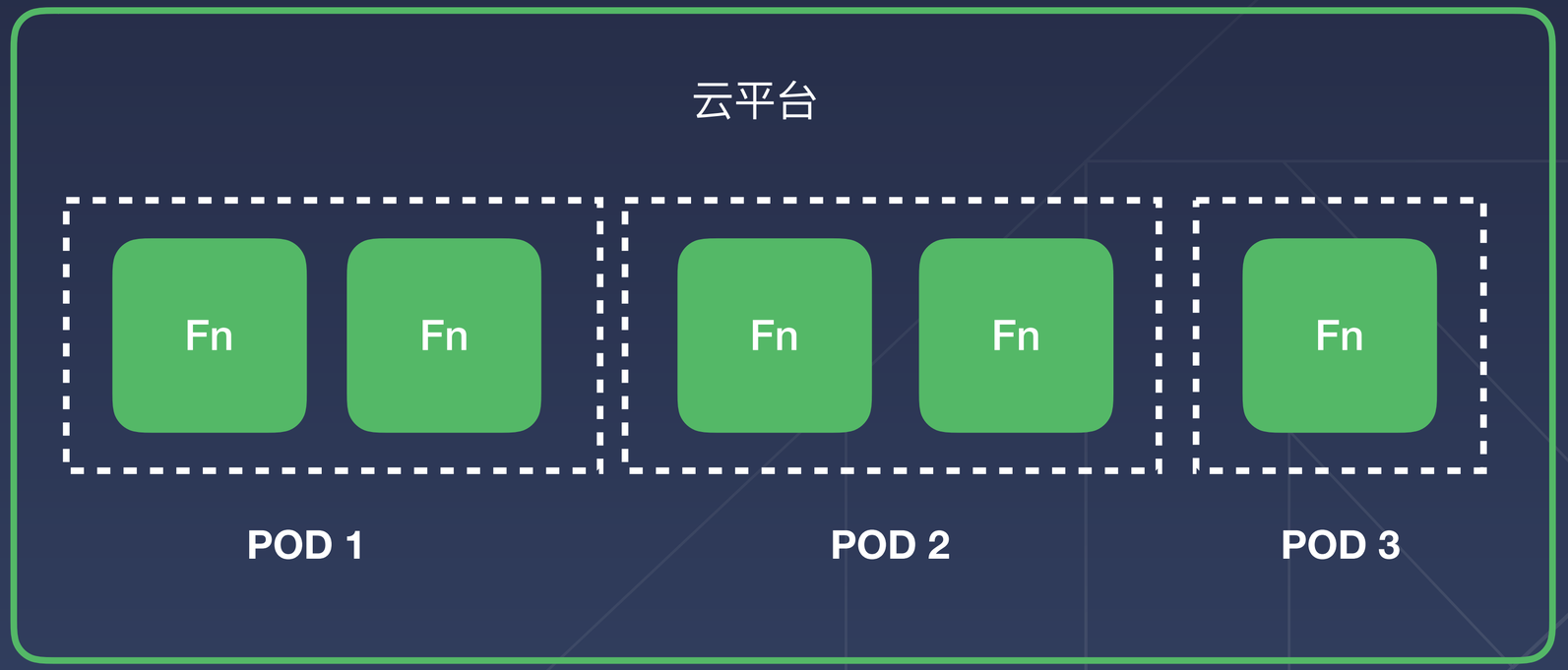
按传统模型，这样的多个函数代码将部署到多个容器中。

函数横向聚合能力



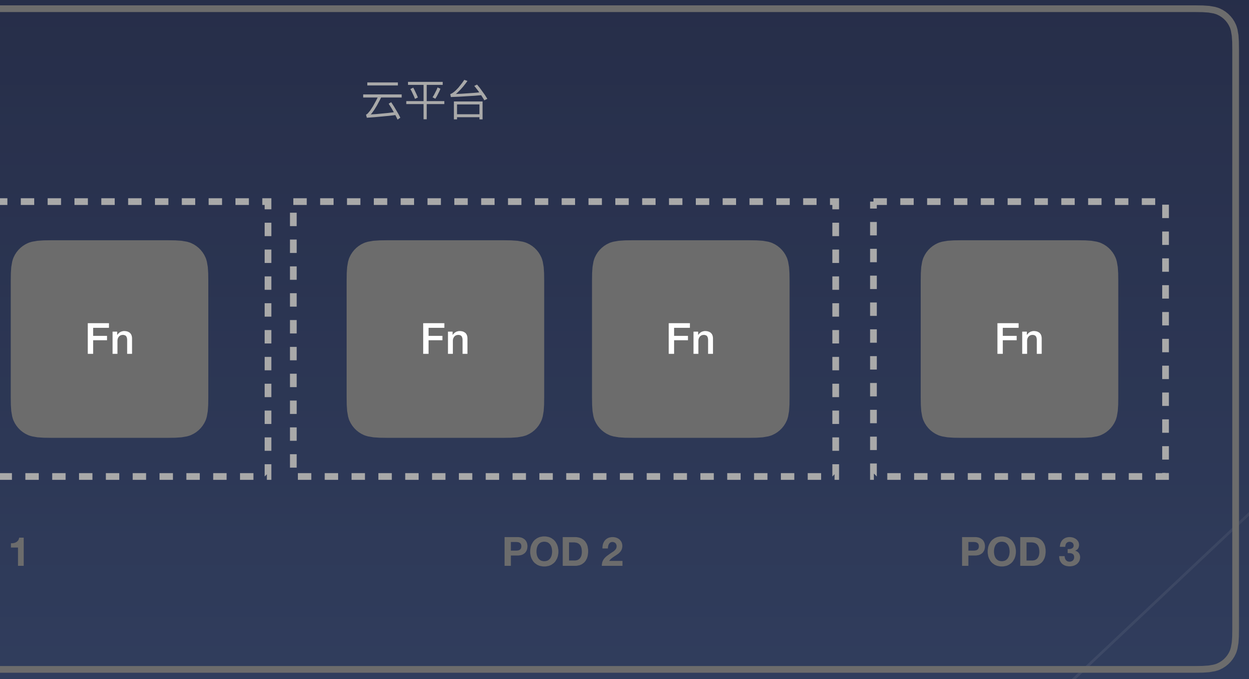
通过调整，我们的函数能够部署到同一个容器中。

函数横向聚合能力



通过调整，我们希望能随意拆分函数的部署结构，成本如何？？

函数横向聚合能力




```yaml
functions:
  index:
    handler: index1.handler
    trigger:
      - http:
        path: /
  help:
    handler: index2.handler
    trigger:
      - http:
        path: /help

## 额外的字段
aggregation:
  index:
    handler: index3.handler
    functions:
      - duizhao3
      - duizhao4
```

随意拆分函数的部署结构，成本如何？？

代码无需变化

设计 Serverless Framework

1 防厂商锁定

Avoid Vendor Lock-in

现在的 FaaS 还没有一个固定的标准，使用时会担心固化在特定平台，后续无法迁移，我们希望思考和解决这个问题。

2 灵活性

Flexibility

函数框架支持灵活的部署模式，可以在垂直和水平两方面进行按需拆分和组合。

3 开发效率

Development Efficiency

提升开发效率，在快速迭代业务的同时，尽可能标准化，易解耦，可扩展和复用。

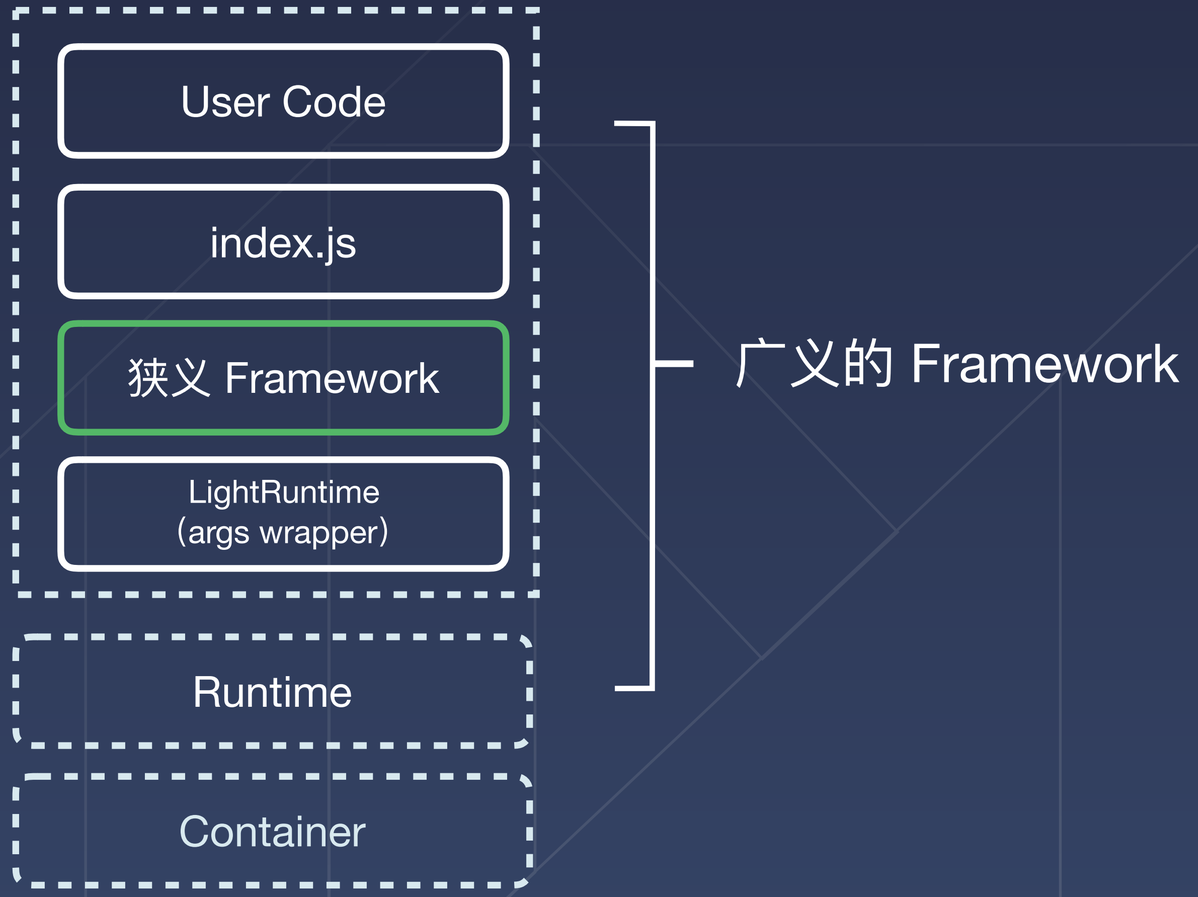
4 生命周期扩展

Lifecycle Extension

在平台运行时和用户代码之间，设计一层通用的运行时扩展能力，在统计埋点，提前加载模块等类似场景上提供支持。

研发模式升级

升级我们的开发模型



3 开发效率 Development Efficiency

函数入参

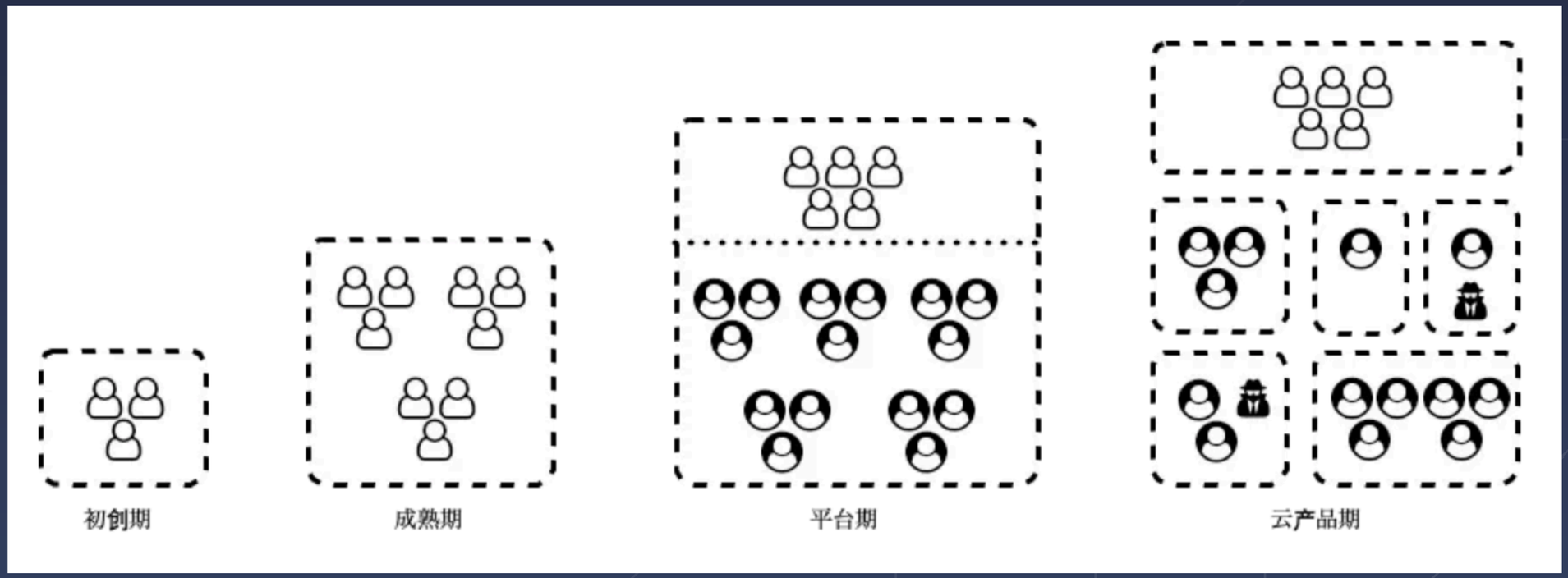
入口方法

```
exports.handler = function (event, context, callback) {
    console.info(null, 'hello world');
    callback(null, 'hello world');
};
```

函数逻辑

传统开发模型

开发效率和团队规模的对比



小 团队规模 大

低 协作复杂度 高

solution 1

IMPORT

TYPESCRIPT

context: **FaaSContext**

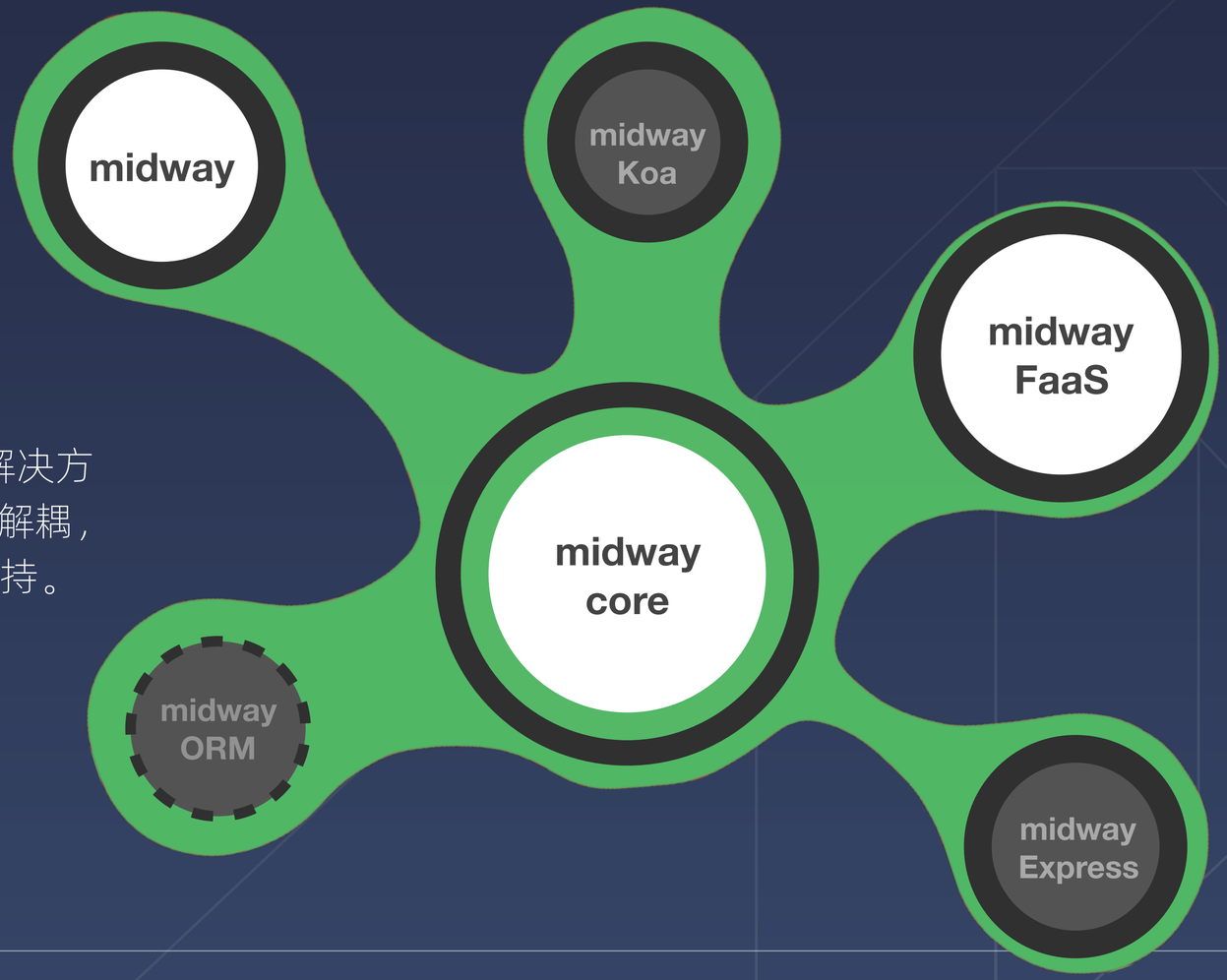
1. 兼容多触发器的入参
2. 定义 ctx 字段，降低开发成本

```typescript
export interface FaaSHTTPContext {
  req: FaaSHTTPRequest;
  res: FaaSHTTPResponse;
  request: FaaSHTTPRequest;
  response: FaaSHTTPResponse;
  headers: FaaSHTTPRequest['headers'];
  method: FaaSHTTPRequest['method'];
  path: FaaSHTTPRequest['path'];
  query: FaaSHTTPRequest['query'];

  get(key: string): string;

  set(key, value);

  type: string;
  status: FaaSHTTPResponse['statusCode'];
  body: FaaSHTTPResponse['body'];
}

export interface ServerlessInvokeOptipns {
  name: string;
  group: string;
  version?: string;
}

export interface ServerlessFunctionInvoker {
  invoke(invokeOpts: ServerlessInvokeOptipns, args: string);
}

export interface FaaSContext extends FaaSHTTPContext {
  logger: FaaSLogger;
  env: string;
  requestContext: RequestContainer;
  originContext: any;
}
```

多场景与传承

midway能力

提供基于 egg 的 TypeScript 解决方案，结合 IoC 将用户代码彻底解耦，更在之上提供了更多灵活性支持。



midway-faas 能力

基于 IoC，为 FaaS 场景提供业务逻辑复用等能力，并在跨平台，多终端集成等场景中提供标准化支持。

升级我们的开发模型

采用了和 midway6 一样架构的 midway-faas，支持大部分逻辑和代码

```ts
@provide()
@func('index.handler')
export class IndexHandler implements FunctionHandler {

  @inject()
  ctx: FaaSContext;

  async handler(event: FaaSEvent): any {
    return {
      data: 'hello world',
      event
    }
  }
}
```

```ts
@provide()
@controller('/')
export class IndexController implements Controller {

  @inject()
  ctx: Context;

  async handler(): any {
    ctx.body = 'test data';
  }
}
```

标准化代码结构

契合生态 和传统的 midway 开发一脉相承，有着同样的研发习惯和工具链。

可维护性 合理的采用 TypeScript 定义减少错误几率，使用依赖注入解耦代码。

高可靠性 使用原有的注入模块，以及统一的治理和监控工具，保持用户习惯。

可维护性，复用代码

{ path: [Module A, Module B, Module C]}

Class + decorator

solution 2

实现一个小而美的 FaaS 框架

Midway For Egg	Midway For Koa	Midway For Express	Midway For FaaS
600 line	220 line	240 line	160 line
符合全栈场景的框架，支持 egg 插件体系的完整版本。	支持 koa 作为基础框架，包含最简单单进程场景的 midway 适配版本。	支持 express 作为基础框架，包含最简单单进程场景的 midway 适配版本。	极简的 for FaaS 场景的支持 IoC 的最小框架版本。
			Selected

```
@provide()
@func('index.handler')
export class HelloService {

  @inject()
  ctx: FaaSContext;  // context

  async handler(event) {
      return 'hello world';
  }
}
```

USE DECORATOR

使用装饰器

设计 Serverless Framework

1 防厂商锁定

Avoid Vendor Lock-in

现在的 FaaS 还没有一个固定的标准，使用时会担心固化在特定平台，后续无法迁移，我们希望思考和解决这个问题。

2 灵活性

Flexibility

函数框架支持灵活的部署模式，可以在垂直和水平两方面进行按需拆分和组合。

3 开发效率

Development Efficiency

提升开发效率，在快速迭代业务的同时，尽可能标准化，易解耦，可扩展和复用。

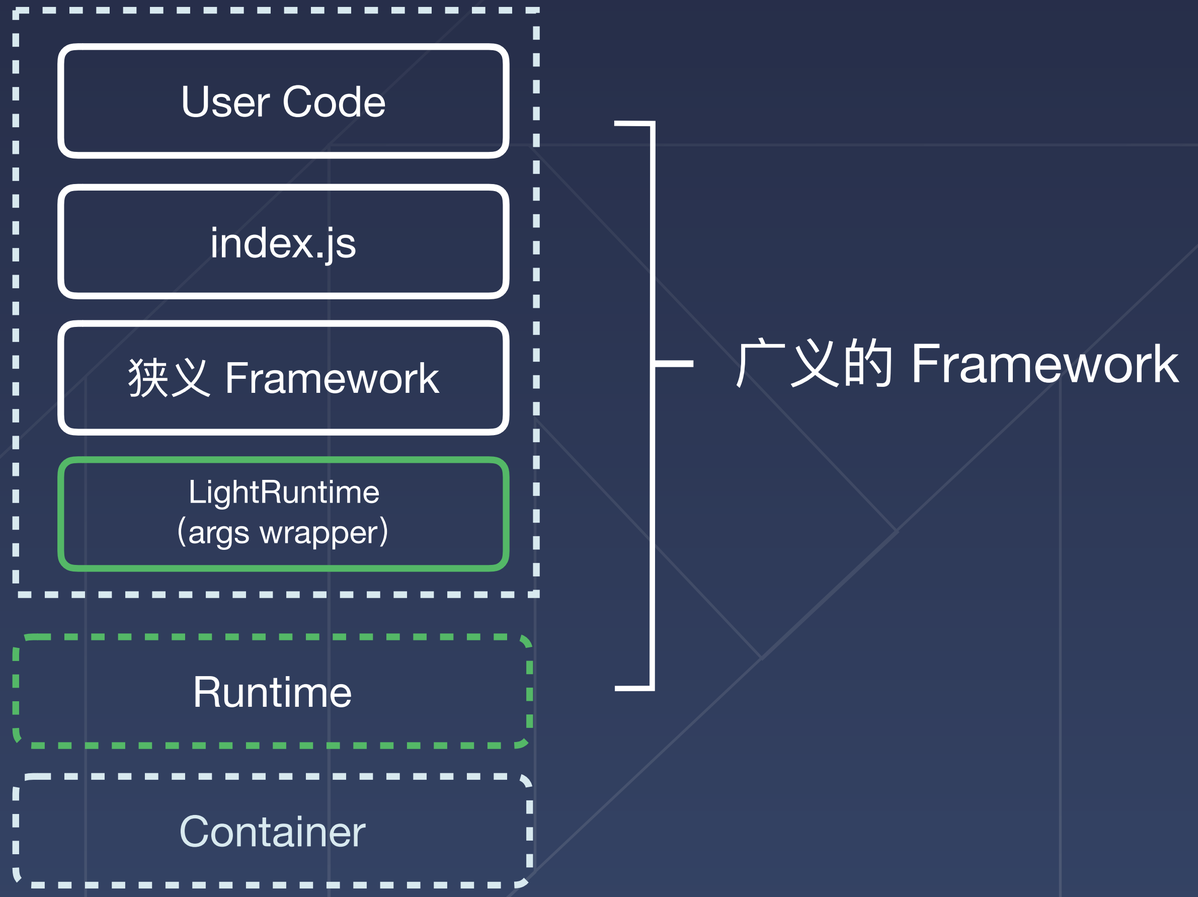
4 生命周期扩展

Lifecycle Extension

在平台运行时和用户代码之间，设计一层通用的运行时扩展能力，在统计埋点，提前加载模块等类似场景上提供支持。

研发模式升级

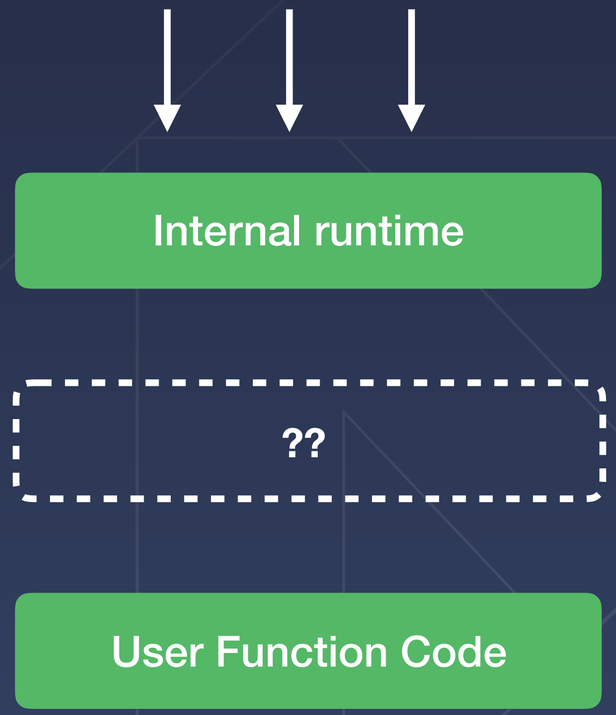
升级我们的开发模型

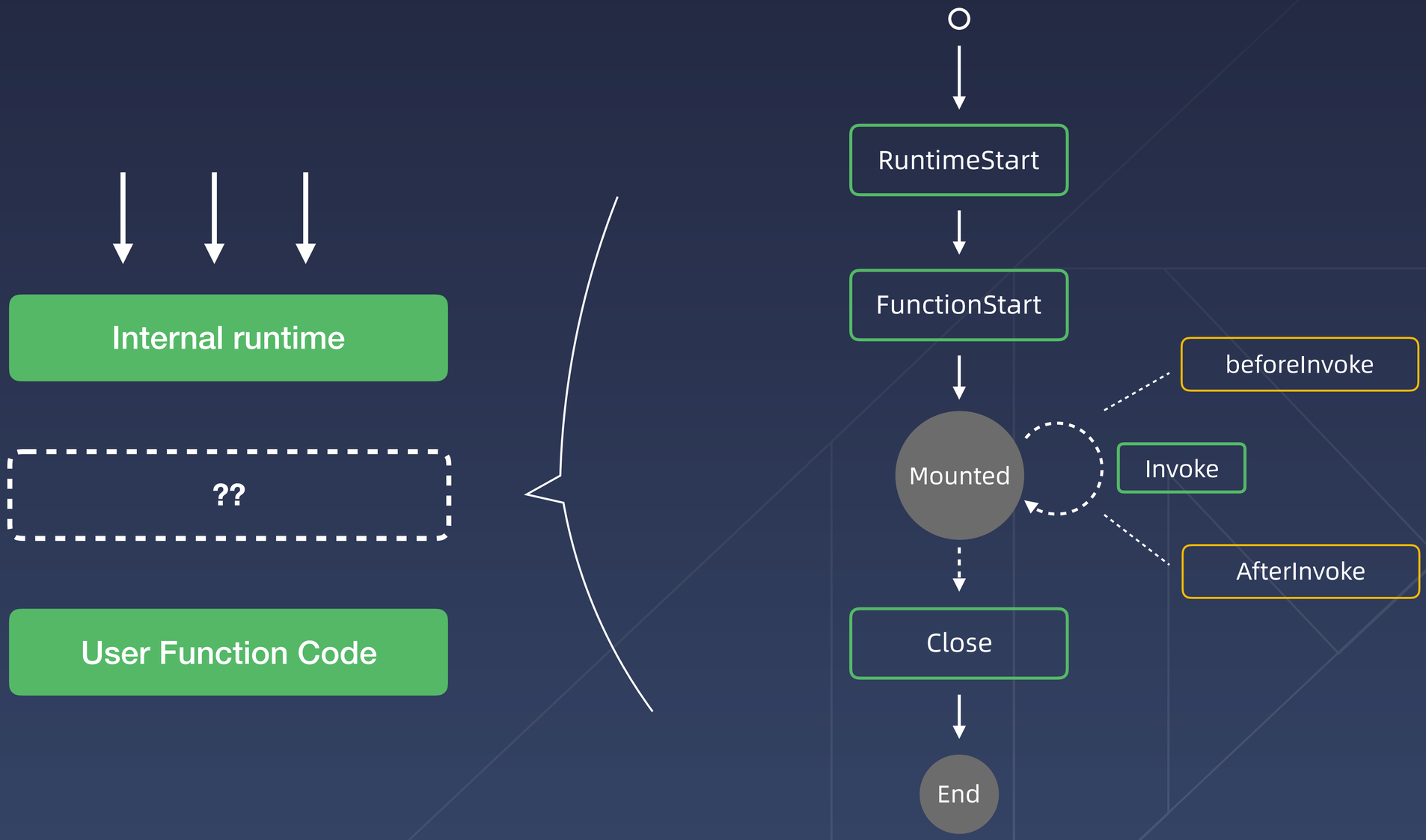


User claims

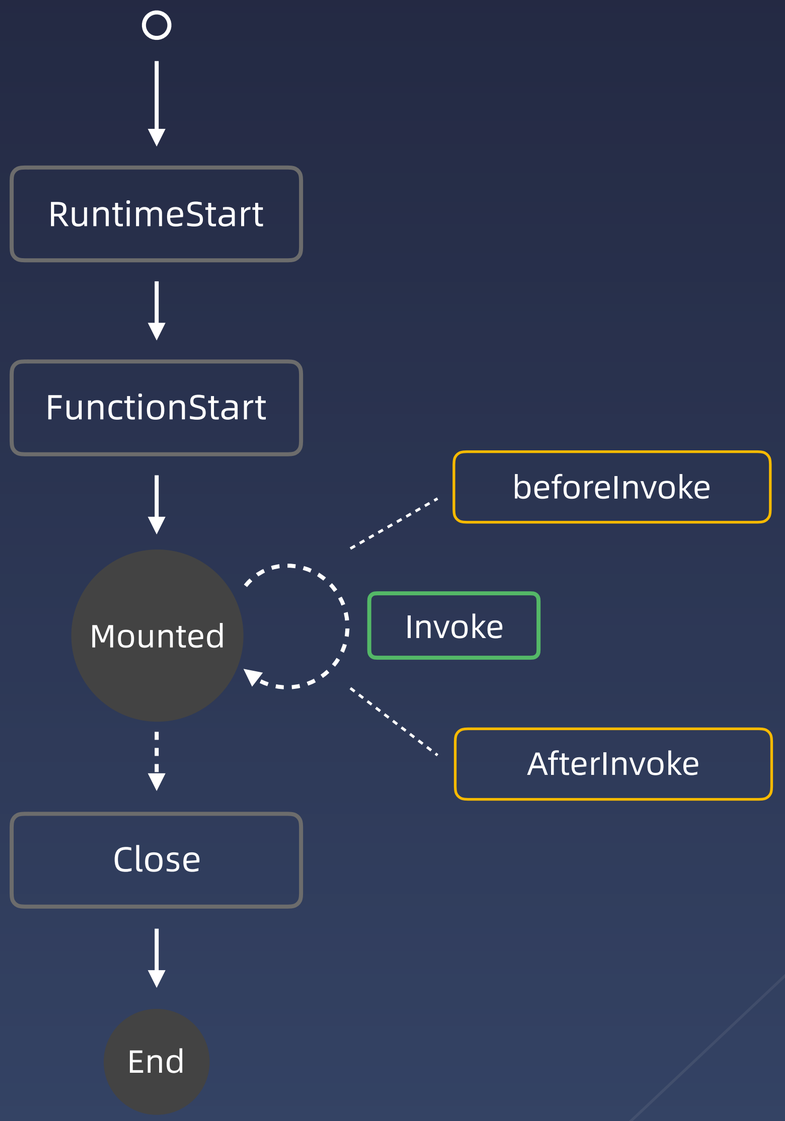
用户诉求

1. 在不同的函数调用前执行相同的逻辑
2. 抹平函数执行前的初始化差异
3. 其他开发者需求（统一治理、监控日志）



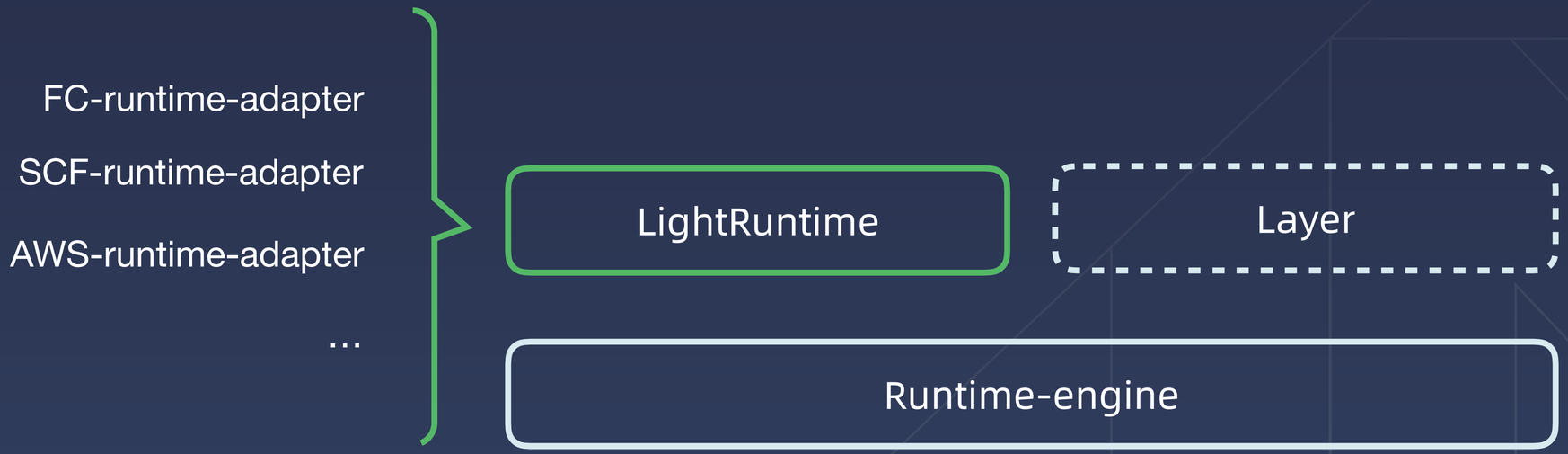

Internal runtime
??
User Function Code
RuntimeStart
FunctionStart
Mounted
beforeInvoke
Invoke
AfterInvoke
Close
End

运行时扩展


RuntimeStart
FunctionStart
beforeInvoke
Mounted
Invoke
AfterInvoke
Close
End

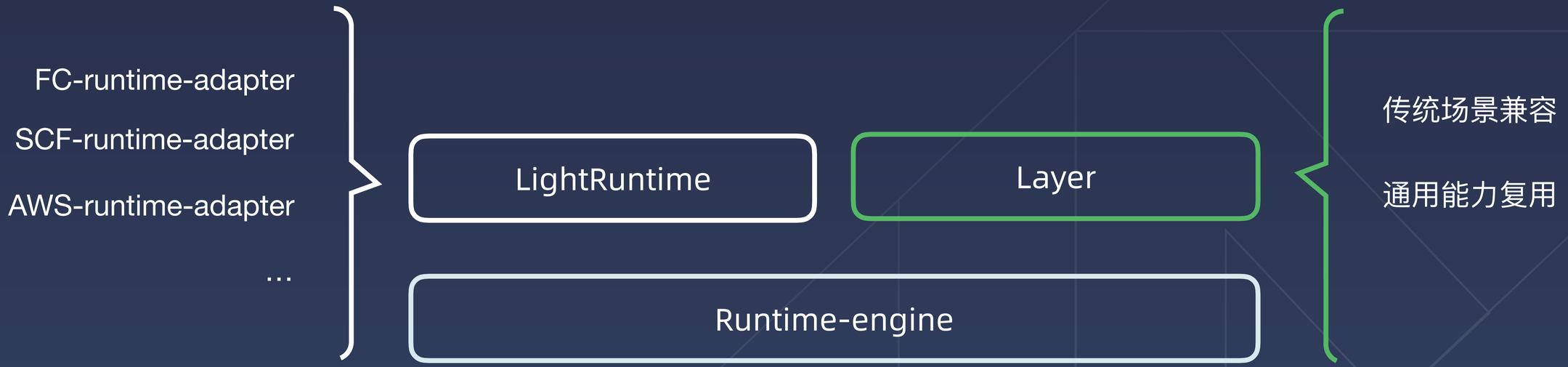

LightRuntime
Layer
Runtime-engine

运行时扩展



LightRuntime 的上层实现用于处理不同平台的兼容性。

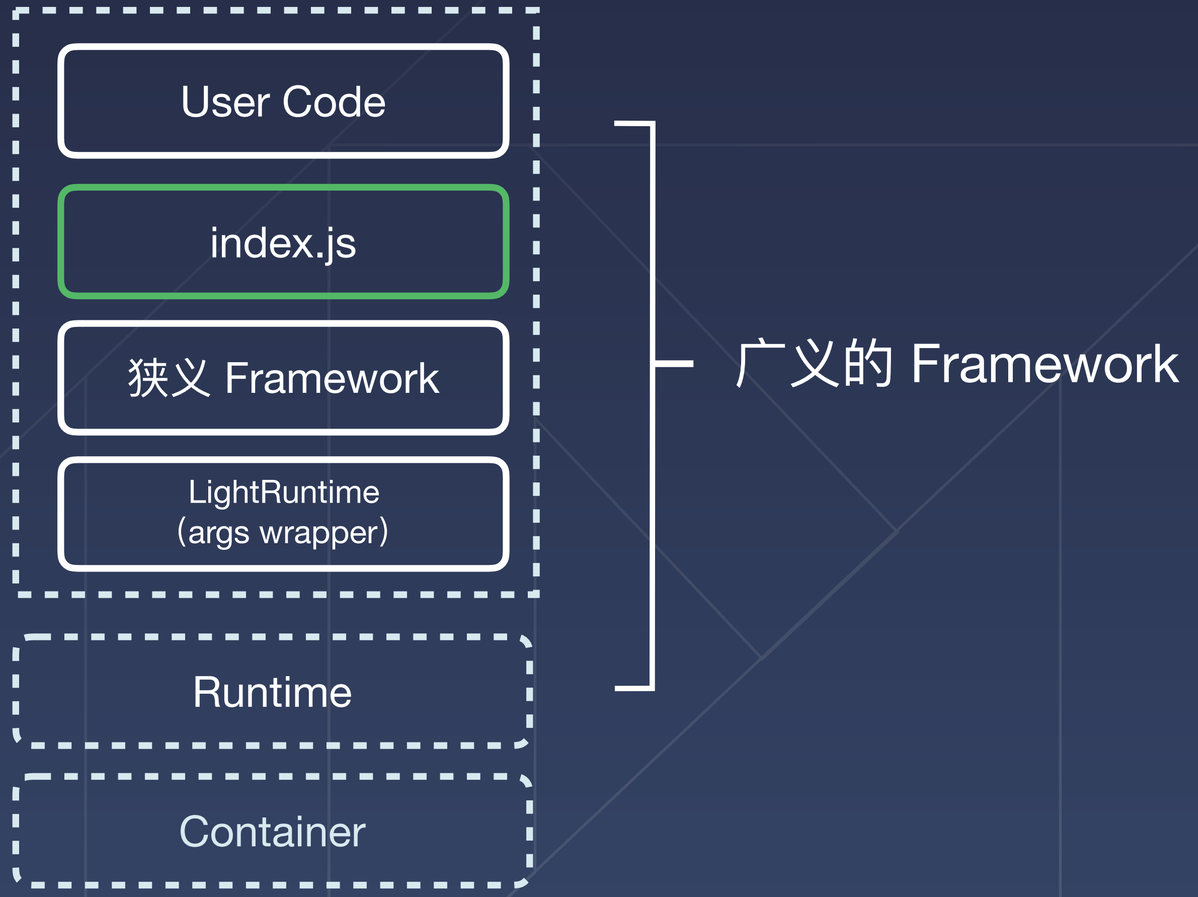
运行时扩展



Layer 用于针对运行时做扩展，方便在多个运行时中复用能力。

研发模式升级

升级我们的运行模型



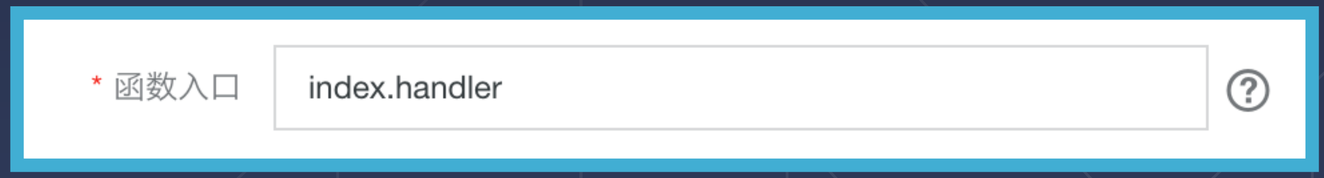
升级我们的运行模型

为了解决多个函数同仓库的问题

代码文件太多，仓库碎片化

同仓库多函数

index.handler



同仓库多函数

index.handler1

index.handler

index.handler2

同仓库多函数

index.handler3

index.handler1

index.handler

index.handler2

index.handler4

同仓库多函数

index.handler3	test.handler1	help.handler1
index.handler1	test.handler2	help.handler2
index.handler	test.handler3	help.handler3
index.handler2	test.handler4	help.handler4
index.handler4	test.handler5	help.handler5

同仓库多函数

index.handler

test.handler1

help.handler1

同仓库多函数

index.handler

test.handler1

help.handler1

index.js

test.js

help.js

同仓库多函数

- dist/
- src/

  index.js

  test.js

  help.js

  package.json

同仓库多函数

- dist/
- src/
  - index.ts
  - test.ts
  - help.ts
- index.js
- test.js
- help.js
- package.json

} 用户入口

} 真实入口

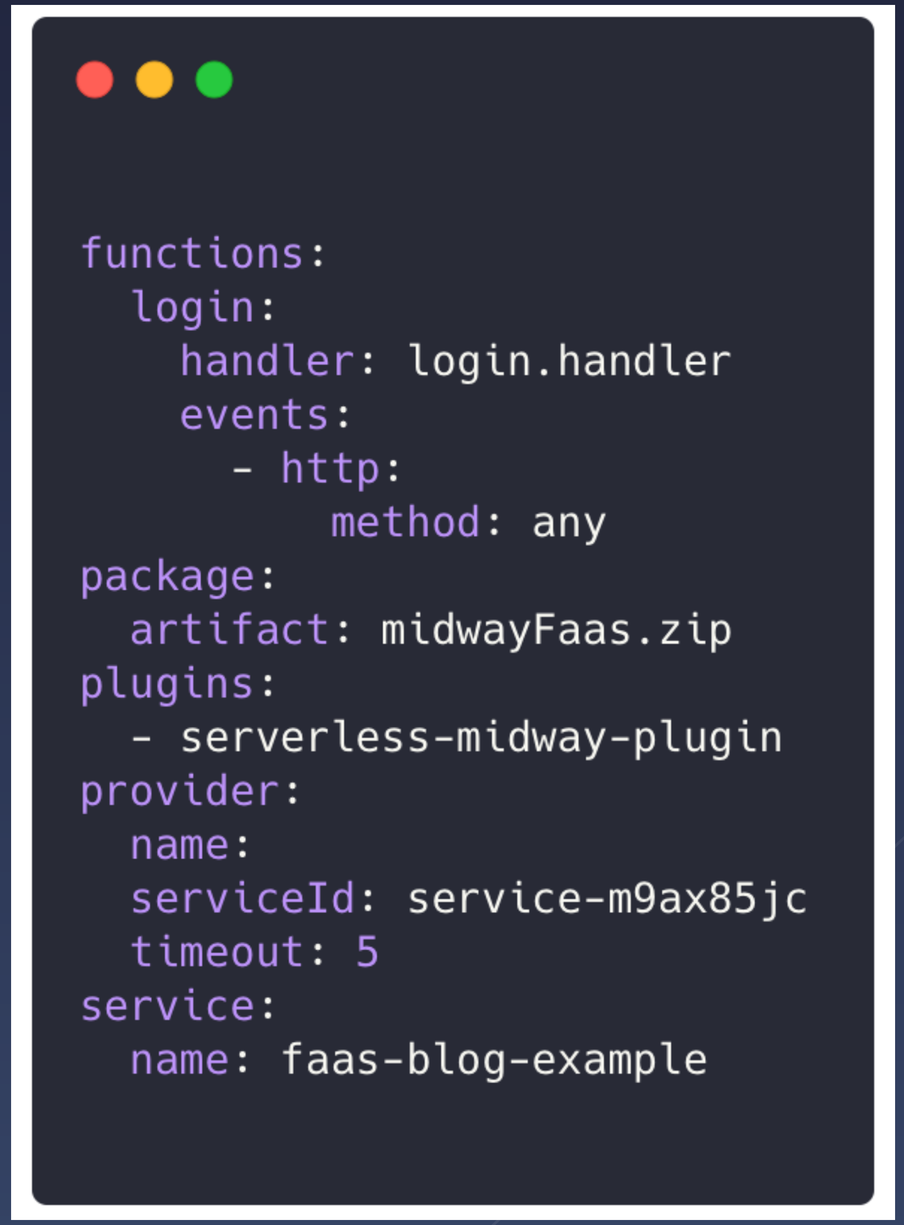
第三章节
企业级 Serverless 体验和实践

代码示例

- 一个 Blog 系统，包含传统 CRUD
- 部署到 **阿里云** 和 **腾讯云** 云函数
- 使用了 MongoDB
- 使用了 midway-faas 技术栈部署

代码结构


阿里云 FC
User API
Tags API
Article API
前端页面（Antd）
腾讯云 SCF
User API
Tags API
Article API
MongoDB

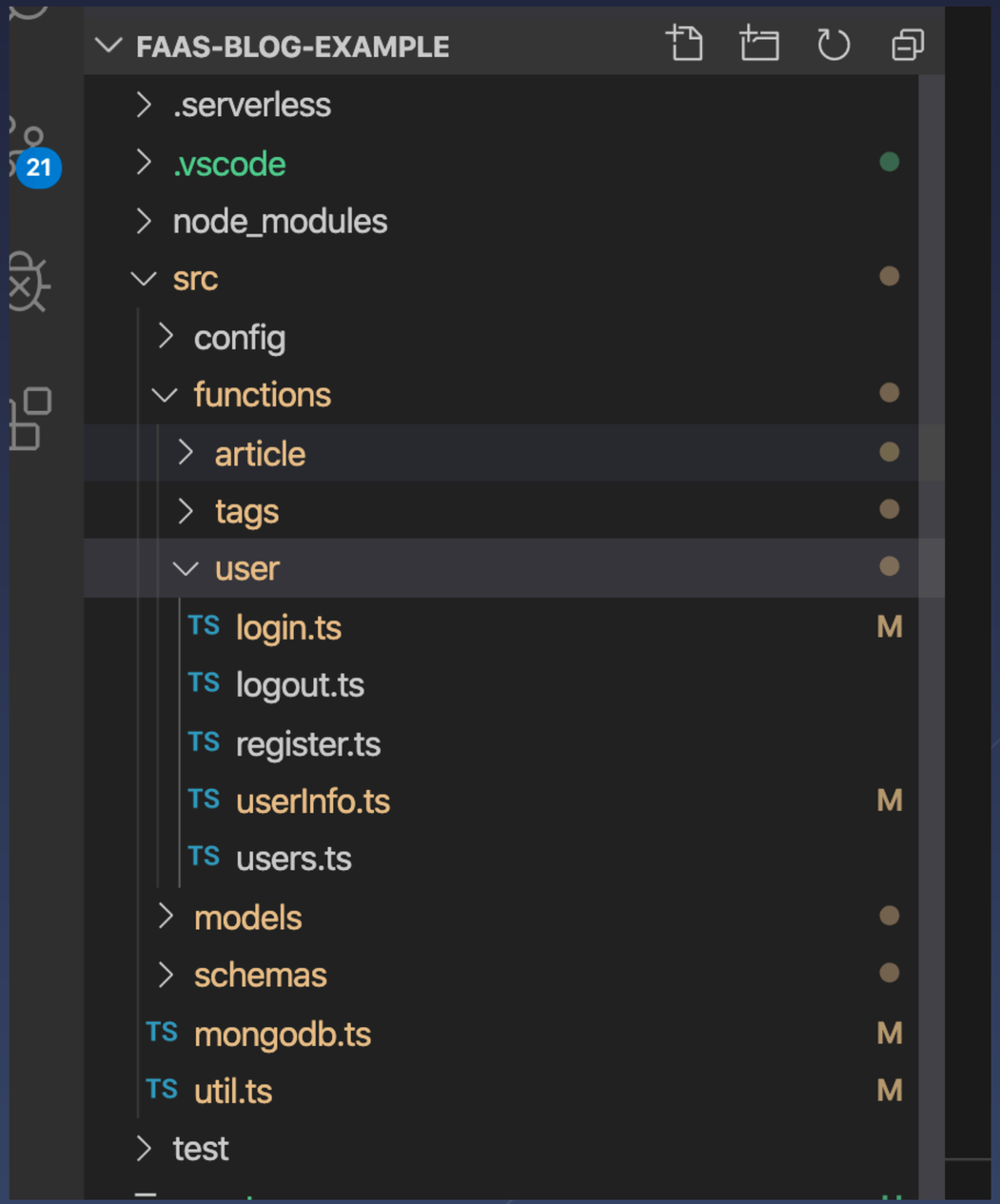


接口定义

打包信息

平台信息

服务名（应用）


FAAS-BLOG-EXAMPLE
.serverless
.vscode
node_modules
src
config
functions
article
tags
user
login.ts
M
logout.ts
register.ts
userInfo.ts
M
users.ts
models
schemas
mongodb.ts
M
util.ts
M
test

简单的目录约定

```
@provide()
@func('login.handler')
export class UserLoginService {
  @inject()
  ctx: FaaSContext; // context

  @init()
  async init() {
    await connectDB();
  }

  async handler() {
    try {
      const userInfo: any = await User.findOne({
        username,
        password: md5(password + MD5_SUFFIX),
      });

      if (userInfo) {
        //登录成功
        const data = {
          username: userInfo.username,
```

入口绑定

初始化逻辑

业务

```
      const userInfo: any = await User.findOne({
        username,
        password: md5(password + MD5_SUFFIX),
      });

      if (userInfo) {
        //登录成功
        const data = {
          username: userInfo.username,
          userType: userInfo.type,
          userId: userInfo._id,
        };
        setUserInfo(this.ctx, data);
        //登录成功后设置session
        // req.session.userInfo = data;
        responseClient(this.ctx, 200, 0, '登录成功', data);
        return;
      }
      responseClient(this.ctx, 400, 1, '用户名密码错误');
    } catch (err) {
      responseClient(this.ctx, null, null, null, err);
    }
  }
}
```

业务

响应

部署示例

调试和测试

调试和测试

```
serverless invok -f login
serverless invoke -f login --debug
```

Console | Sources | Memory | Adblock Plus | Profiler

Filesystem | Snippets

+ Add folder to workspace

Sync changes in DevTools with the local filesystem

Learn more

local.ts | debug.ts | login.ts? [sm] | internal/async_hooks.js

```
1  import { FaaSContext, func, inject, provide } from '@midwayjs/faas';
2  import { User } from '../../models/user';
3  import { md5, MD5_SUFFIX, responseClient, setUserInfo } from '../../util';
4  import { connectDB } from '../../mongodb';
5
6  @provide()
7  @func('login.handler')
8  export class UserLoginService {
9    @inject()
10   ctx: FaaSContext; // context
11
12   async handler() {
13     this.ctx.originContext.callbackWaitsForEmptyEventLoop = false;
14     await connectDB();
15     const { username, password } = this.ctx.req.body as any;
16
17     if (!username) {
18       responseClient(this.ctx, 400, 2, '用户名不可为空');
19       return;
20     }
21
22     if (!password) {
23       responseClient(this.ctx, 400, 2, '密码不可为空');
24       return;
25     }
26
27     try {
28       const userInfo: any = await User.findOne({
29         username,
30         password: md5(password + MD5_SUFFIX),
31       });
32
33       if (userInfo) {
34         //登录成功
35         const data = {
36           username: userInfo.username,
37           userType: userInfo.type,
38           userId: userInfo._id,
39         };
40         setUserInfo(this.ctx, data);
41         //登录成功后设置session
42         // req.session.userInfo = data;
43         responseClient(this.ctx, 200, 0, '登录成功', data);
44         return;
45       }
46       responseClient(this.ctx, 400, 1, '用户名密码错误');
47     } catch (err) {
48       responseClient(this.ctx, null, null, null, err);
```

Debugger paused

Watch

Call Stack

handler	login.ts? [sm]:13
exports.faasDebug	debug.ts:4
innerFun	local.ts:130

Scope

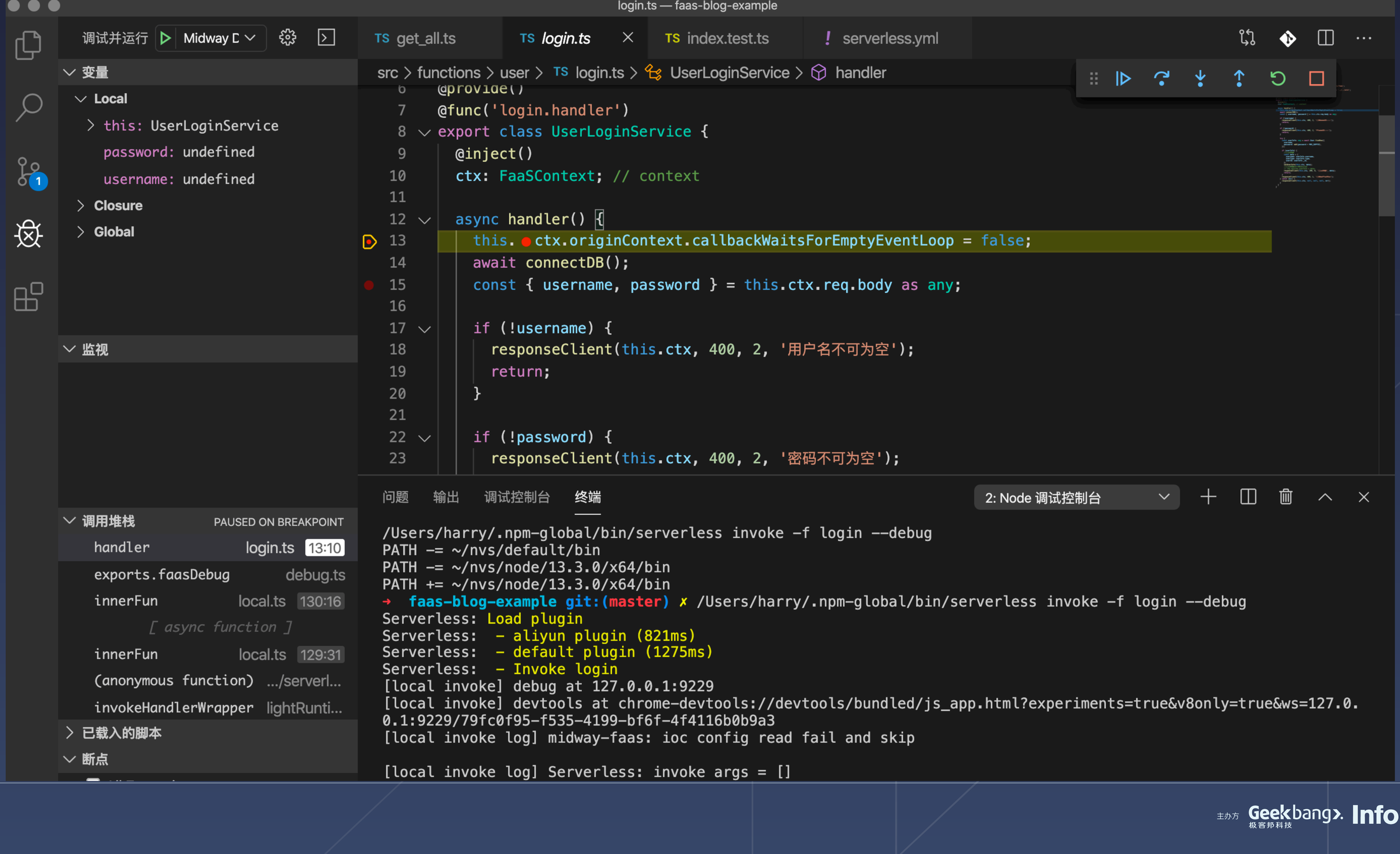
Local
- username: undefined
- password: undefined
- this: UserLoginService

Closure

Global global

Breakpoints

login.ts? [sm]:17
if (!username) {



第四章节
从传统应用迁移到 Serverless 体系

express

koa

next generation web framework for node.js

Born to build
better enterprise
frameworks and apps with
Node.js & Koa

为企业级框架和应用而生

User claims

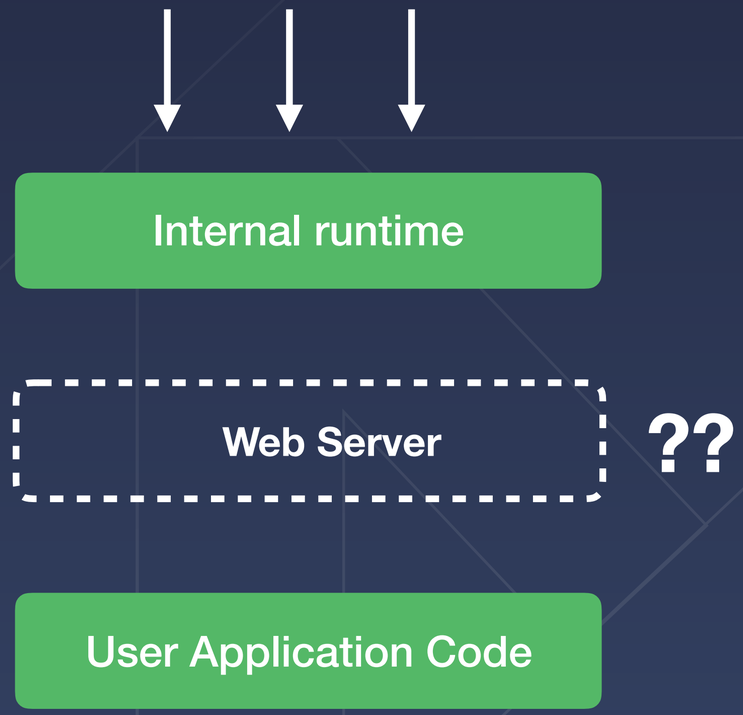
用户诉求

1. 传统应用迁移，不想增加成本
2. 需要跨不同的平台
3. 有逐步迁移到标准 FaaS 的可能性

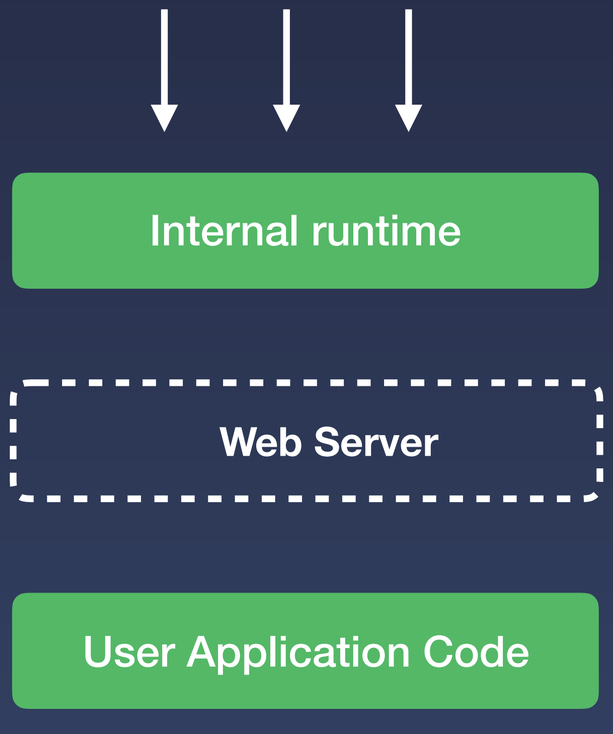
User claims

用户诉求

1. 传统应用迁移，不想增加成本
2. 需要跨不同的平台
3. 有逐步迁移到标准 FaaS 的可能性



运行时扩展



Implement

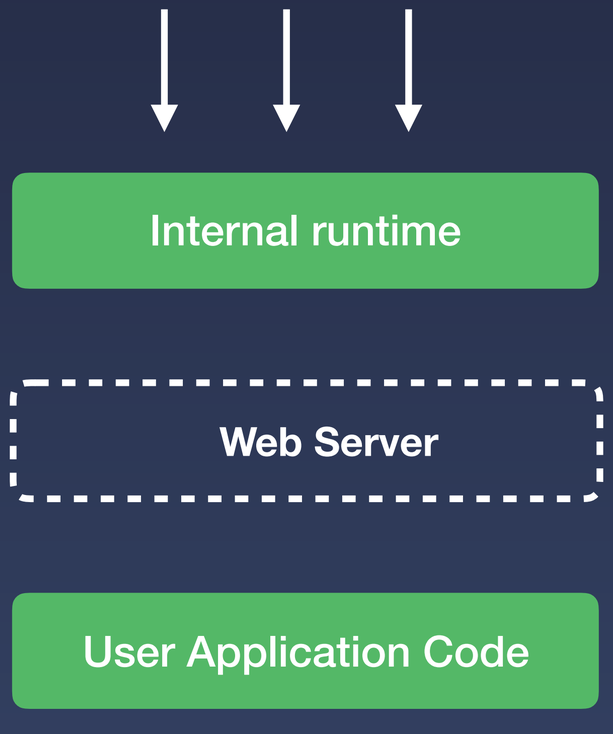
LightRuntime

Layer

Lifecycle

Runtime-Engine

运行时扩展



Implement

Lifecycle

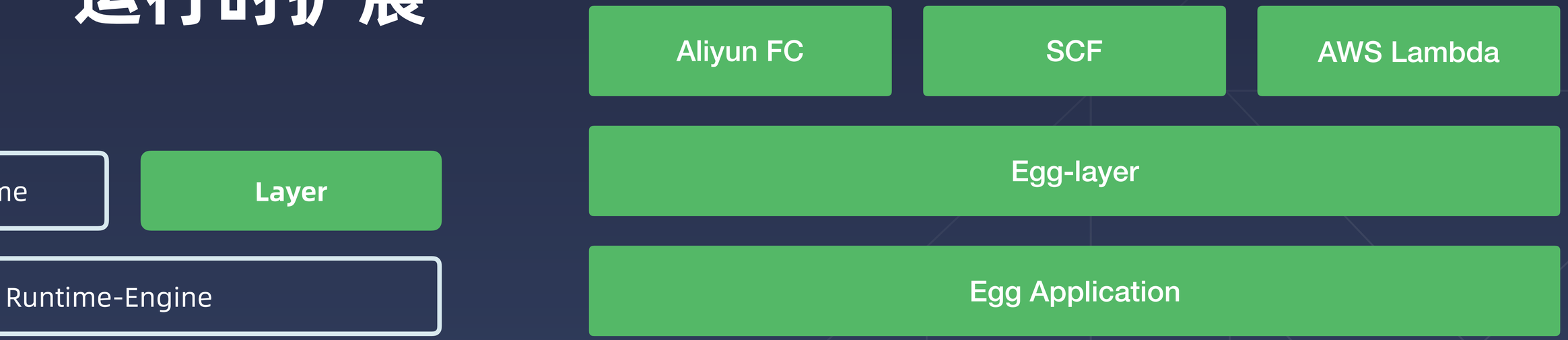
LightRuntime

Layer

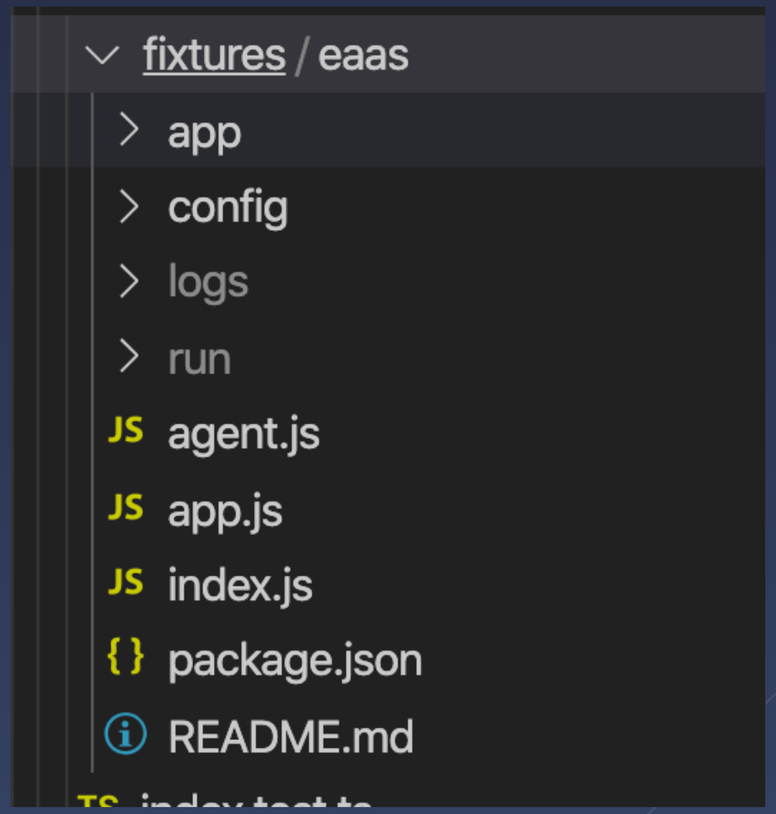
Runtime-Engine

运行时扩展

运行时扩展


ne
Layer
Runtime-Engine
Aliyun FC
SCF
AWS Lambda
Egg-layer
Egg Application

Egg-Layer




```
'use strict';

const { asyncWrapper } = require('@midwayjs/runtime-engine');
const { start } = require('@midwayjs/serverless-fc-starter');
const eggLayer = require('../../../dist');

let runtime;
let inited;

exports.handler = asyncWrapper(async (...args) => {
  if (!inited) {
    inited = true;
    runtime = await start({
      layers: [eggLayer]
    });
  }
  return runtime.asyncEvent()(...args);
});
```

第五章节

私有化 Node.js 运行时方案

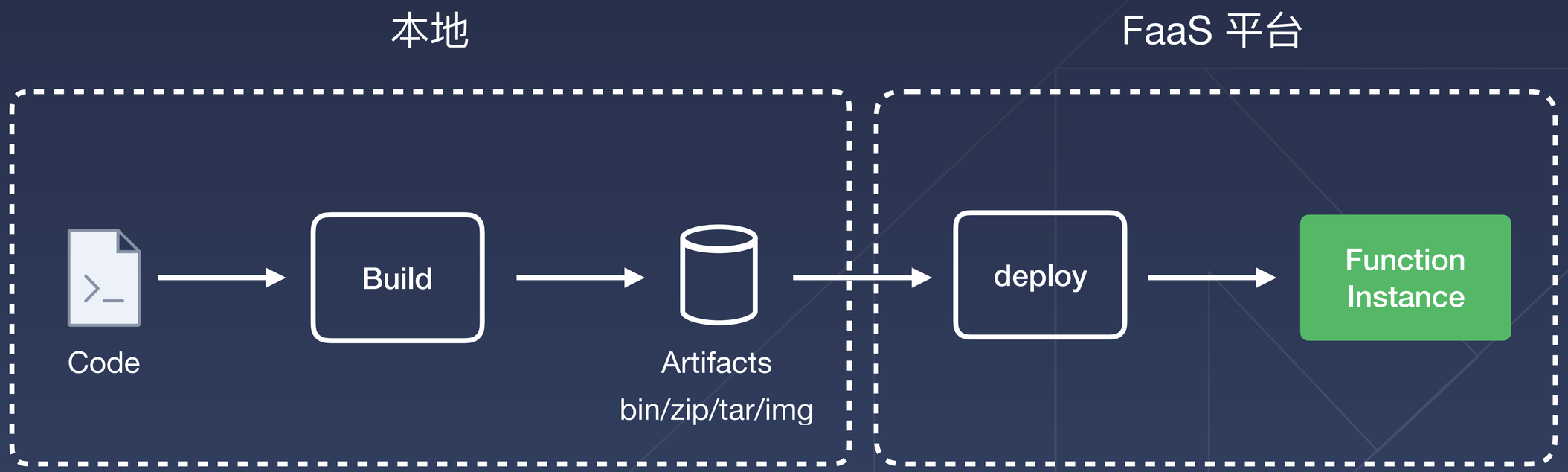
社区的函数方案


入口文件
User Code
Midway-faas
函数运行时
LightRuntime
Layer
函数调用执行
云平台网关
内部运行时

我们需要写一个自己的 Runtime

首先，在企业中，有一个自己的 Serverless 平台

Runtime 在 FaaS 中的位置



Function Instance

Code

```
(name) => {
  return `Hi, ${name}!`;
}
```

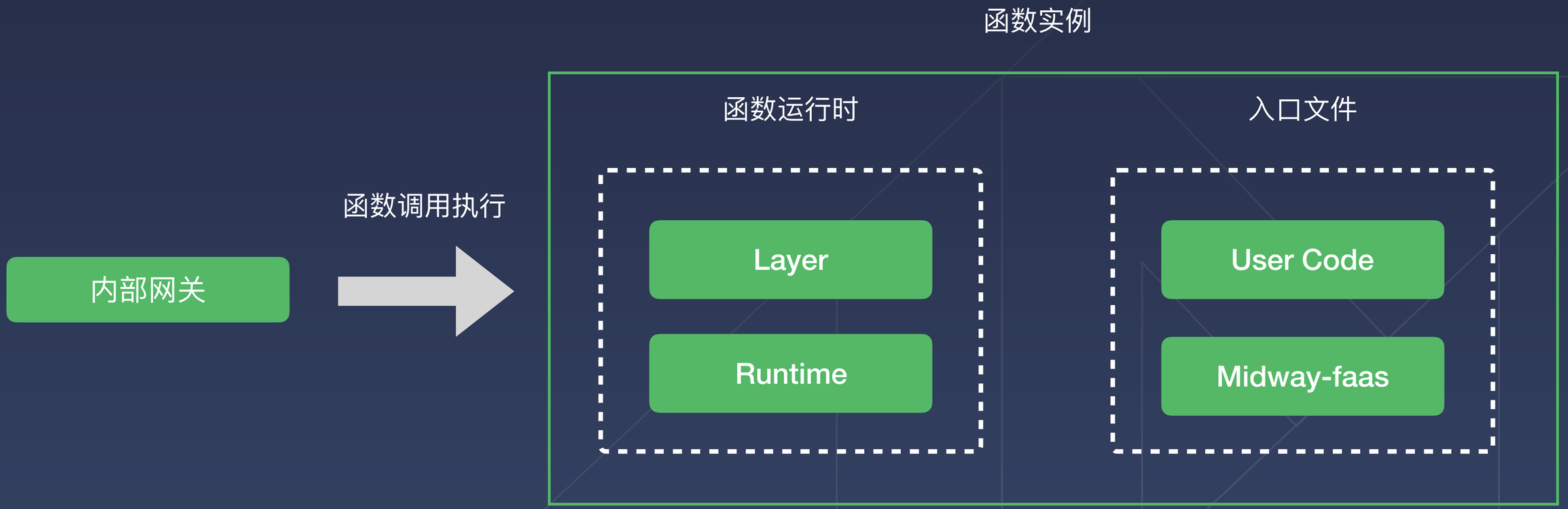
Container

Function Instance

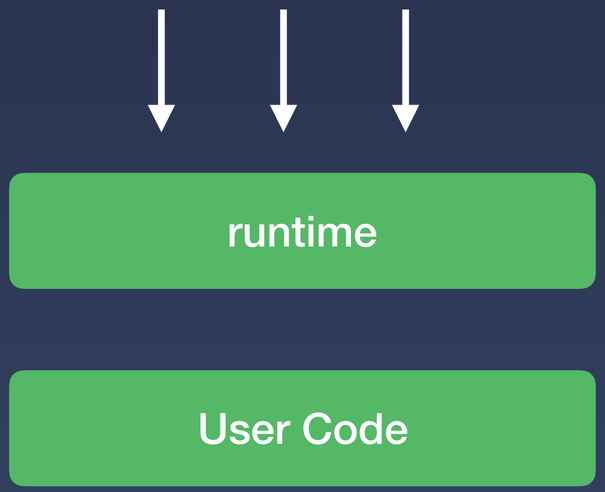
Runtime 的本质

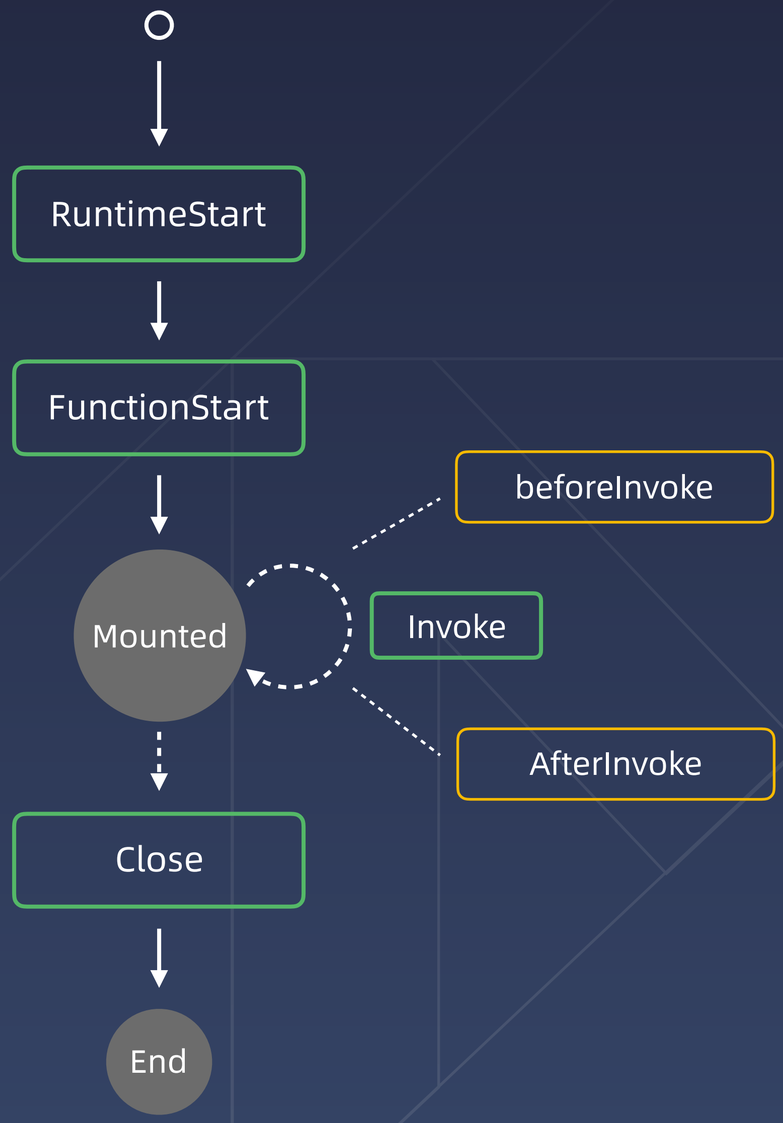
1. 对内：容纳代码的环境 (Node.js / v8)
2. 对外：对接外部事件（网关）的服务 （http）
3. 其他：更多的扩展能力（Layer）

我们的完整的函数方案


函数实例
函数运行时
入口文件
函数调用执行
内部网关
Layer
Runtime
User Code
Midway-faas

完整生命周期


runtime
User Code


RuntimeStart
FunctionStart
beforeInvoke
Mounted
Invoke
AfterInvoke
Close
End

示例

面向未来

1. Serverless 是一个非常适合前端去开拓和挖掘的一个新体系
2. 不断的寻找新场景（SSR/传统技术栈），性能优化（极速启动）
3. 携手生态共赢

THANKS

GMTC
全球大前端技术大会