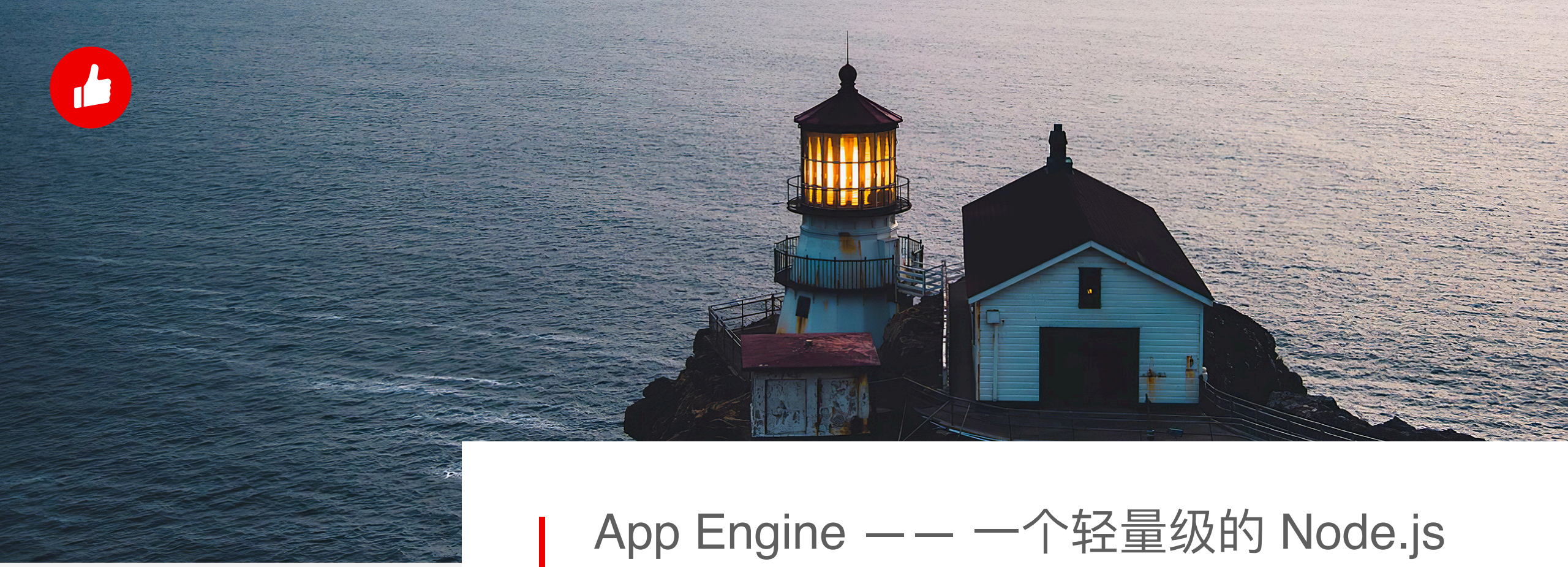


App Engine —— 一个轻量级的 Node.js 服务托管和调度平台

王干 / KK @有赞前端 基础业务 / 服务引擎小组负责人

分享内容

1

应用引擎背景介绍

介绍有赞应用引擎出现的背景、前端服务的特点和现状

以及应用引擎的目标和价值

2

应用引擎设计及实践总结

分阶段逐步设计构建应用引擎生态体系，介绍应用引擎

总体架构及核心模块实现原理

3

应用引擎产品设计

4

FAQ

Part 1：应用引擎背景介绍

前端 Node 服务特点

- 轻量级，模型简单
- 轻 CPU 重 I/O
- 二方依赖少
- 改动频繁，需随产品快速迭代

前端 Node 服务现状与问题

- 缺少应用框架/库版本控制和治理
- 缺少运行时动态定制能力
- 缺少各个层面的 apm 能力
- 缺少应用诊断能力
- 长尾应用多，极低的资源利用率
- 代码直接部署在容器上，启停流程包含不必要的动作
- ……

应用引擎目标和价值

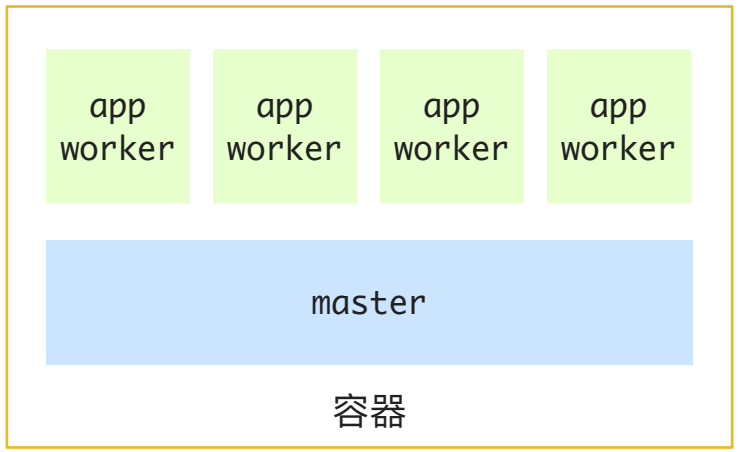
- 包版本治理
- 前端服务治理
- 构建前端一体化研发体系
- 更高的资源利用率，极速启停
- 开箱即用的二方能力
- 支持大客单元化部署
- ……

Part 2：应用引擎设计及实践总结

- 阶段一：自主运行模式
- 阶段二：平台托管模式
- 阶段三：FaaS 模式

阶段一：自主运行模式

原有的 Master-Worker 进程模型

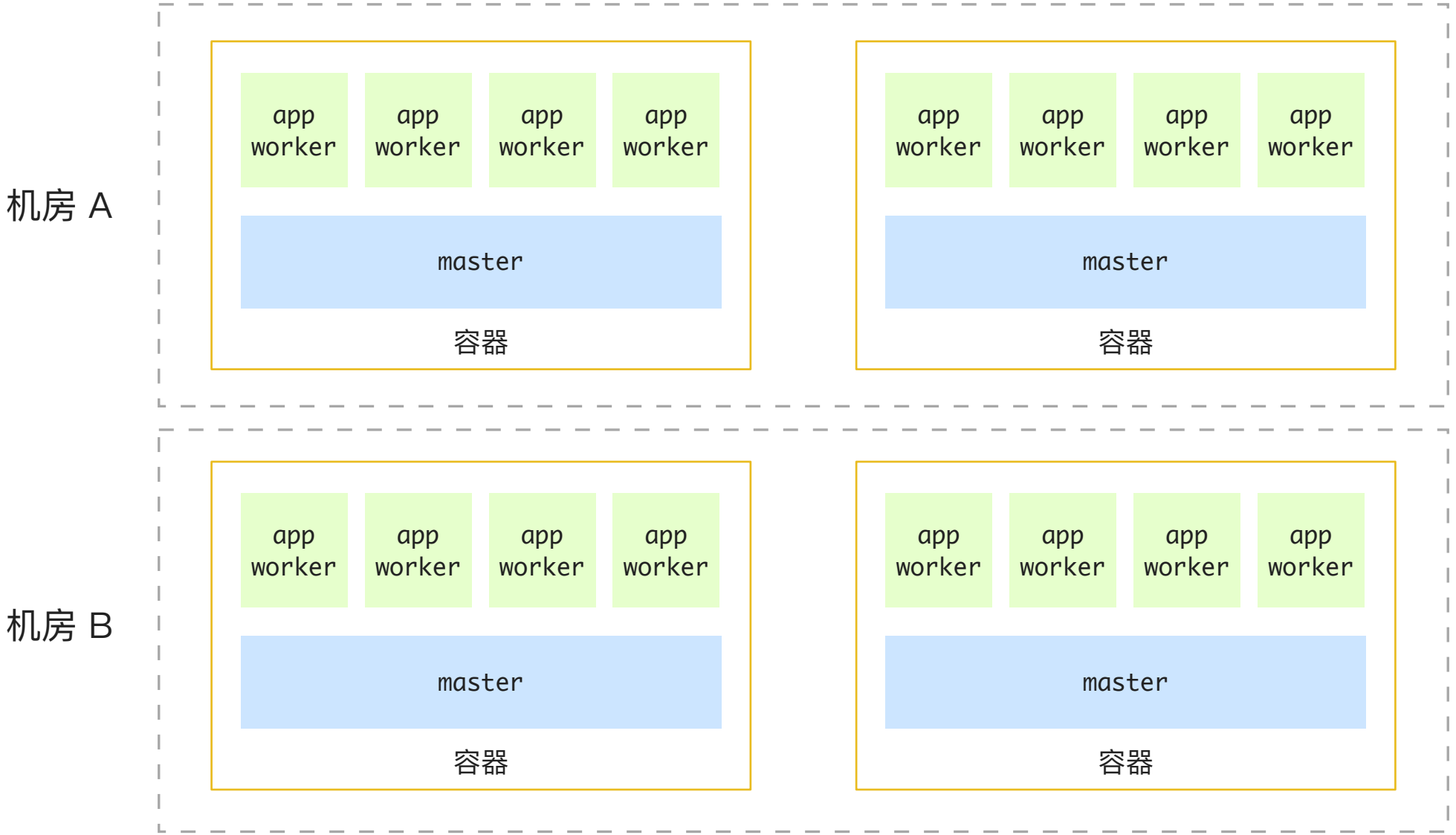


What is cluster?

A single instance of Node.js runs in a single thread. To take advantage of multi-core systems, the user will sometimes want to launch a cluster of Node.js processes to handle the load.

The cluster module allows easy creation of child processes that all share server ports.

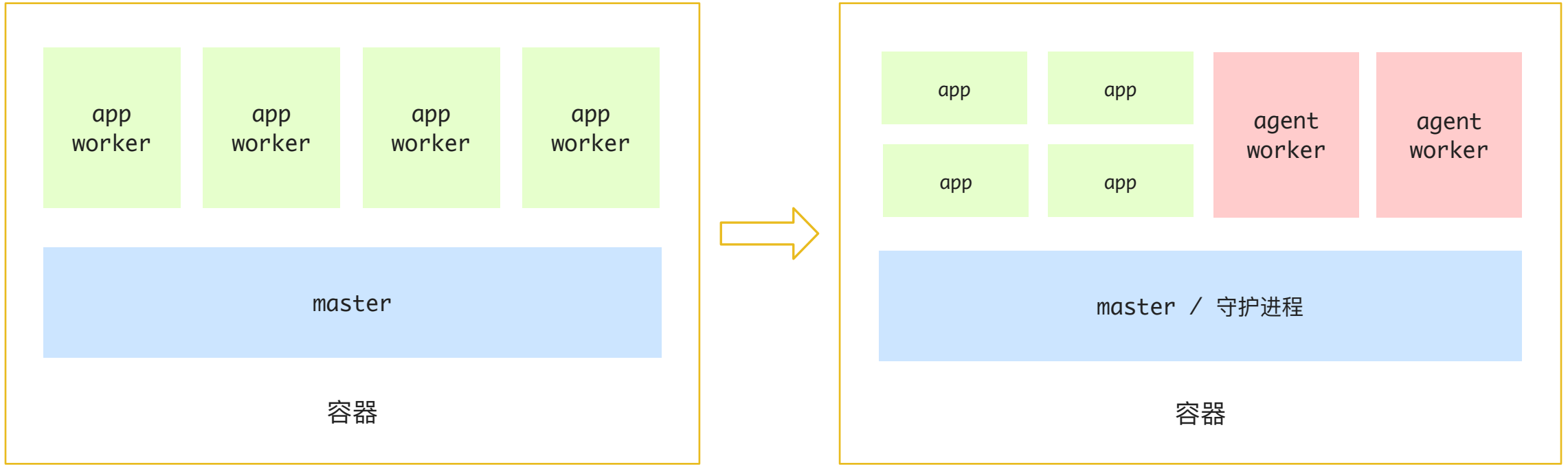
Master–Worker 进程模型



Master-Worker 进程模型存在的问题?

- 多个 worker 进程之间如何共享资源?
- worker 进程异常退出以后如何处理?
- 多个 worker 进程之间如何通信?

进程模型优化

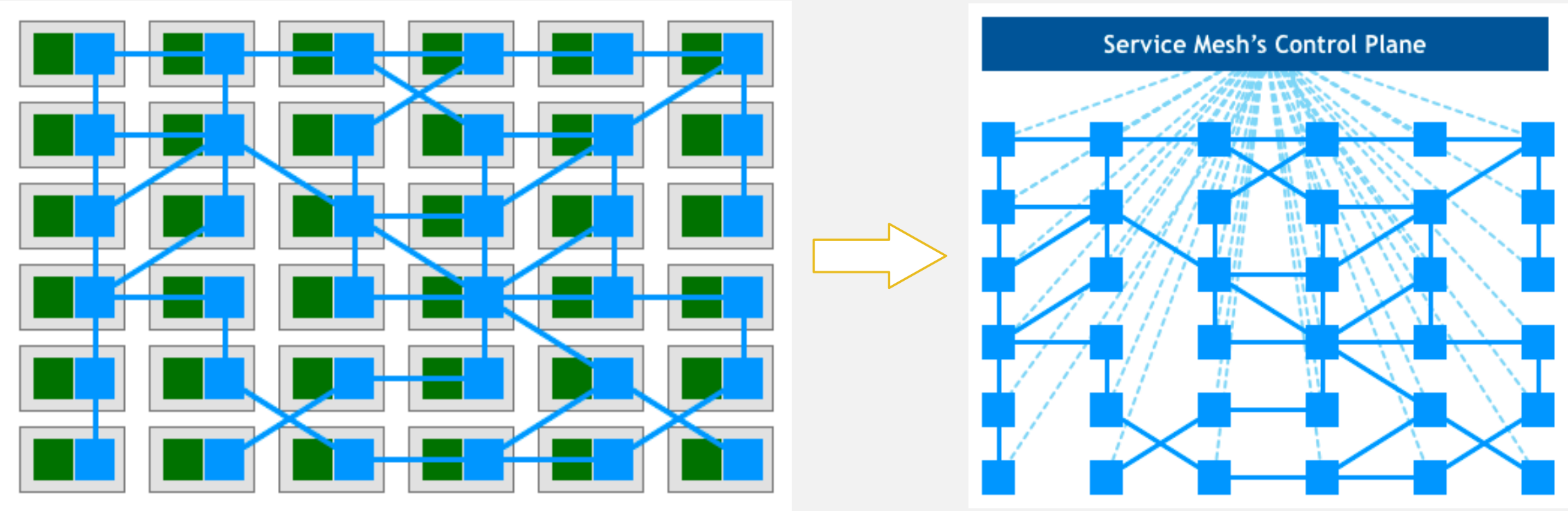


一主多 worker

一主多 worker、agent 模型

进程模型优化

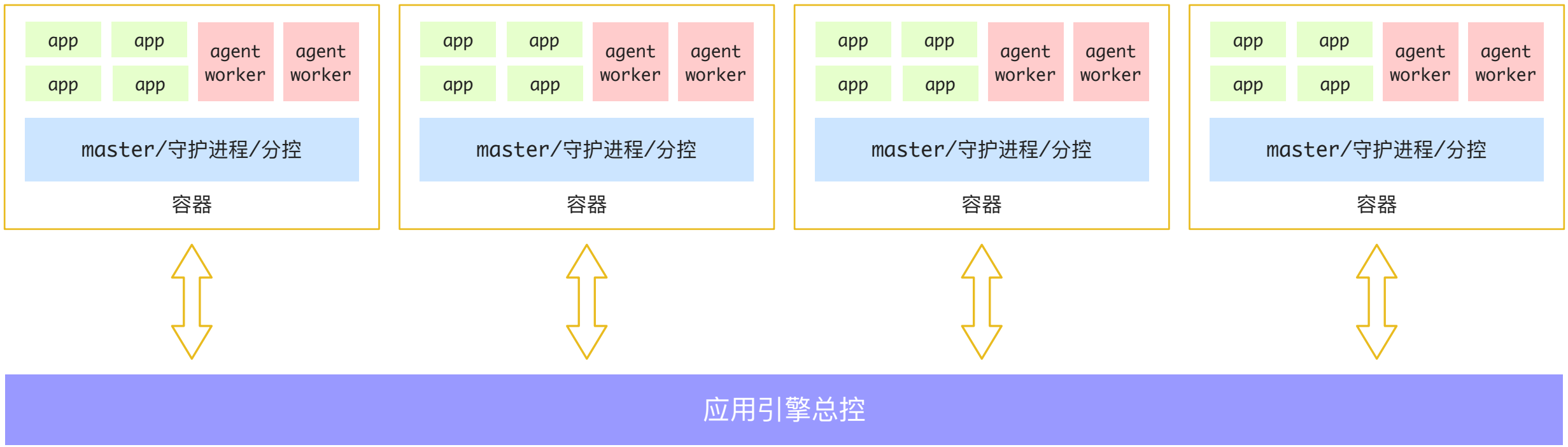
进程类型	进程数量	作用
master	1	守护进程：负责进程管理、进程间通信
app worker	一般设置为 CPU 核心数	业务应用进程：执行业务代码
agent worker	1	非业务进程：执行非业务代码，例如长连接客户端



当 Sidecar 在架构中越来越多时，需要对 Sidecar 进行统一管理。

于是，我们为 Sidecar 增加了一个全局控制器，用于统一管控 Sidecar。

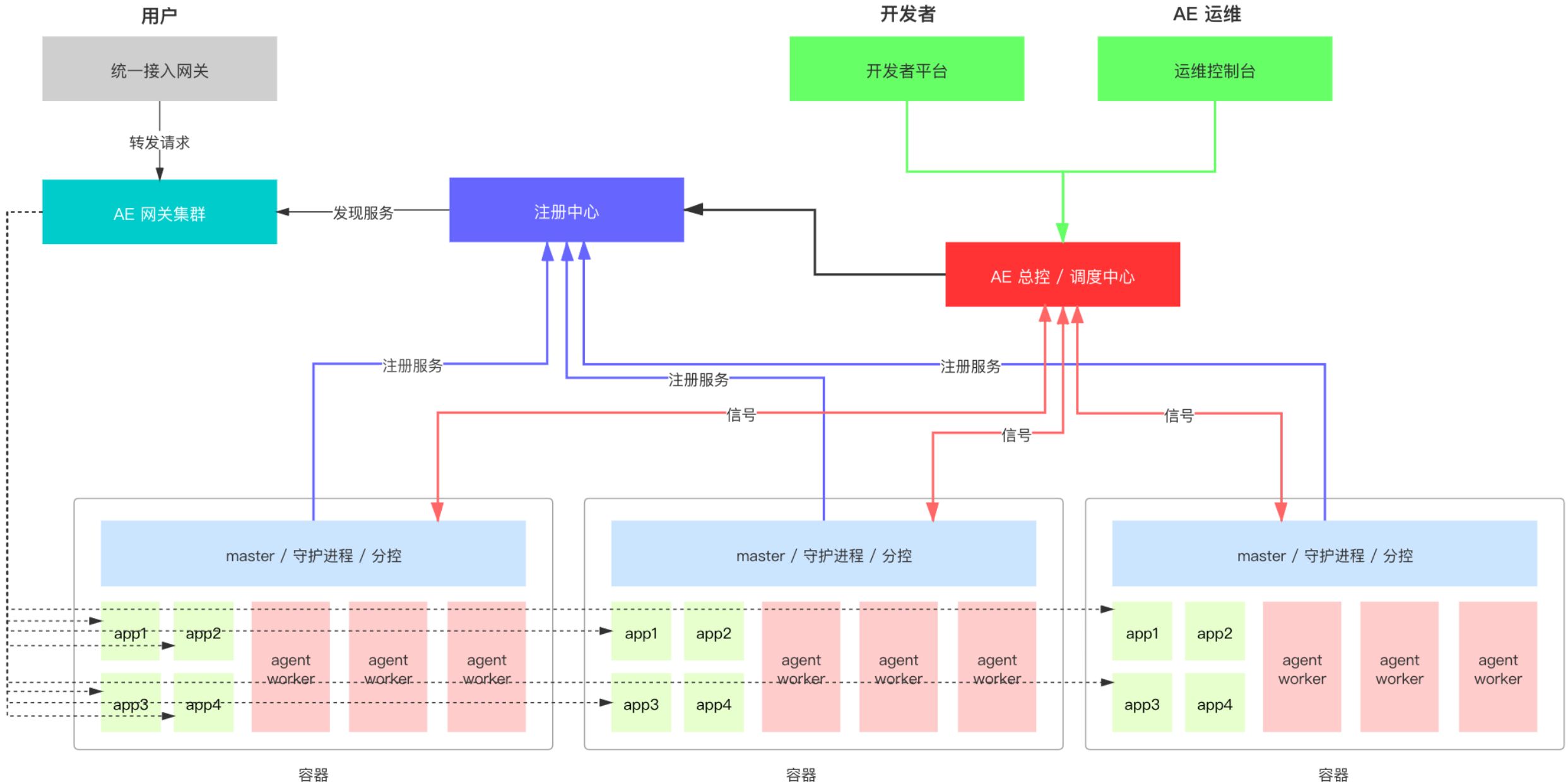
阶段一：自主运行模式


app
app
app
app
agent worker
agent worker
master/守护进程/分控
容器
app
app
app
app
agent worker
agent worker
master/守护进程/分控
容器
app
app
app
app
agent worker
agent worker
master/守护进程/分控
容器
app
app
app
app
agent worker
agent worker
master/守护进程/分控
容器
应用引擎总控

阶段一：自主运行模式

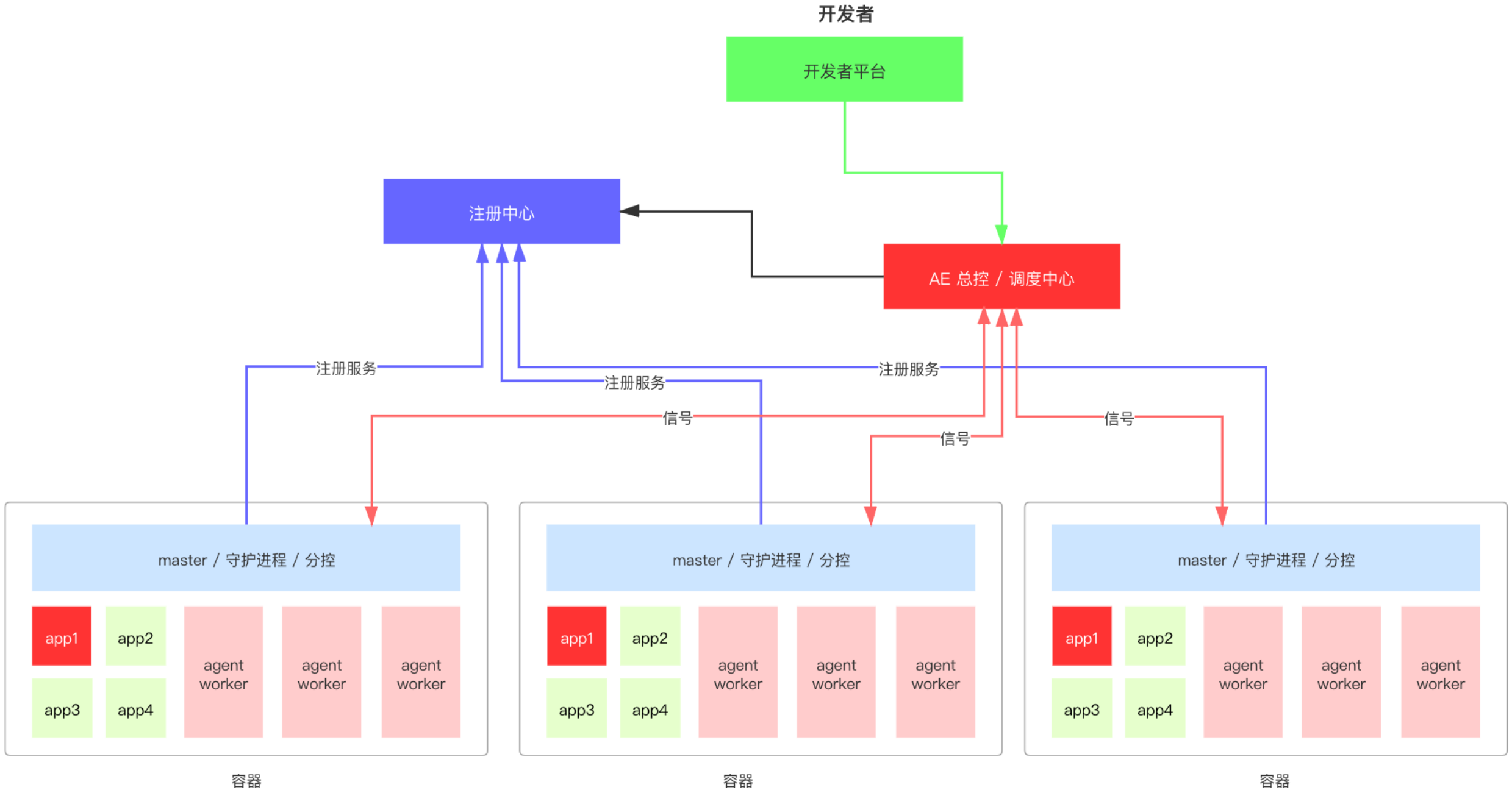
- 进程守护模型：支持进程间通信及 agent worker 机制
- 控制平面：增加一个全局控制中心，统一管控所有分控
- 总控与分控机制：总控与应用建立长连接，支持应用基本数据及状态的查询

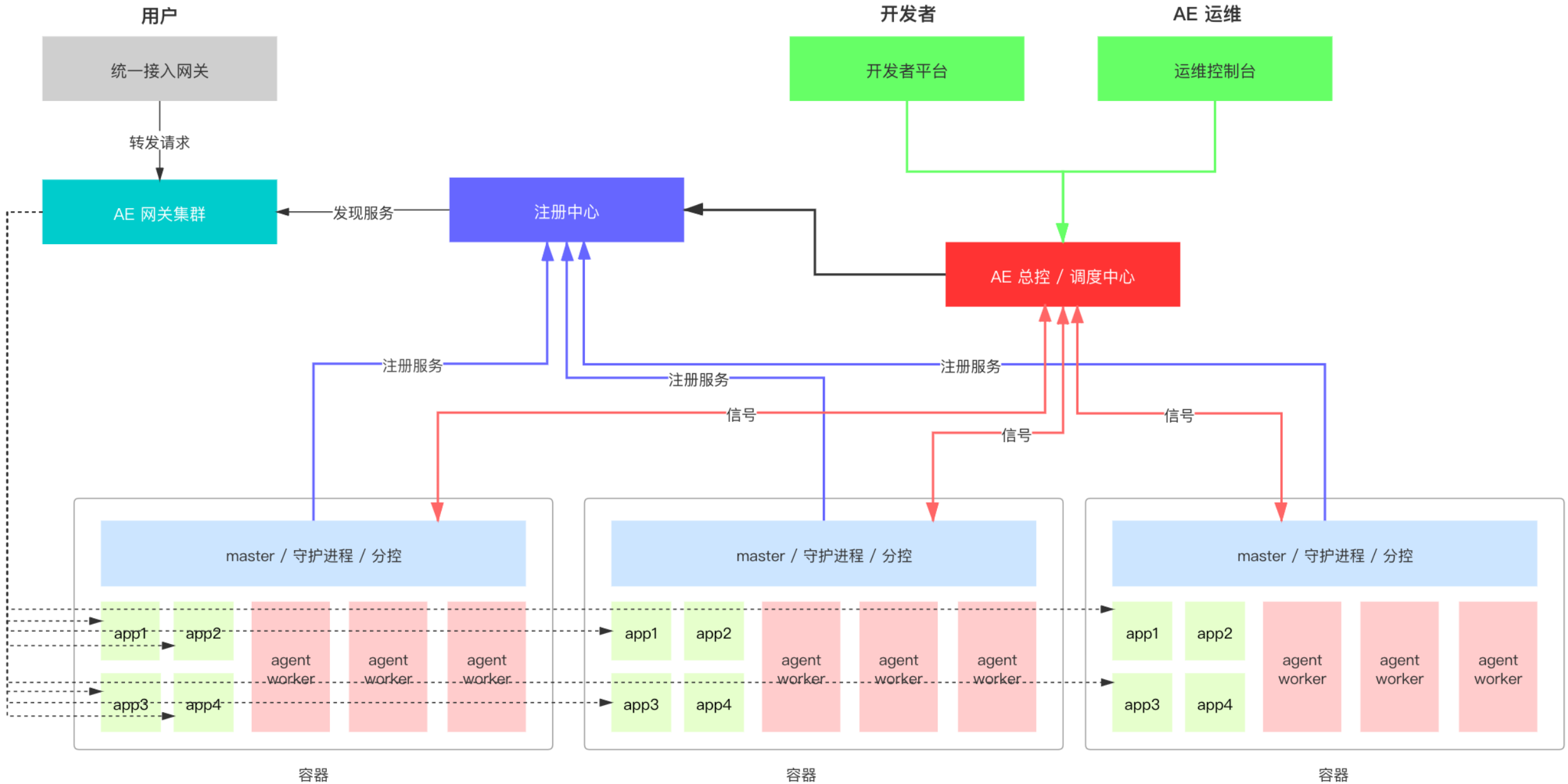
阶段二：平台托管模式


用户
开发者
AE 运维
统一接入网关
开发者平台
运维控制台
转发请求
AE 网关集群
发现服务
注册中心
AE 总控 / 调度中心
注册服务
注册服务
注册服务
信号
信号
信号
master / 守护进程 / 分控
master / 守护进程 / 分控
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
app1
app2
app3
app4
agent worker
agent worker
agent worker
app1
app2
app3
app4
agent worker
agent worker
agent worker
容器
容器
容器

开发者会有哪些操作?

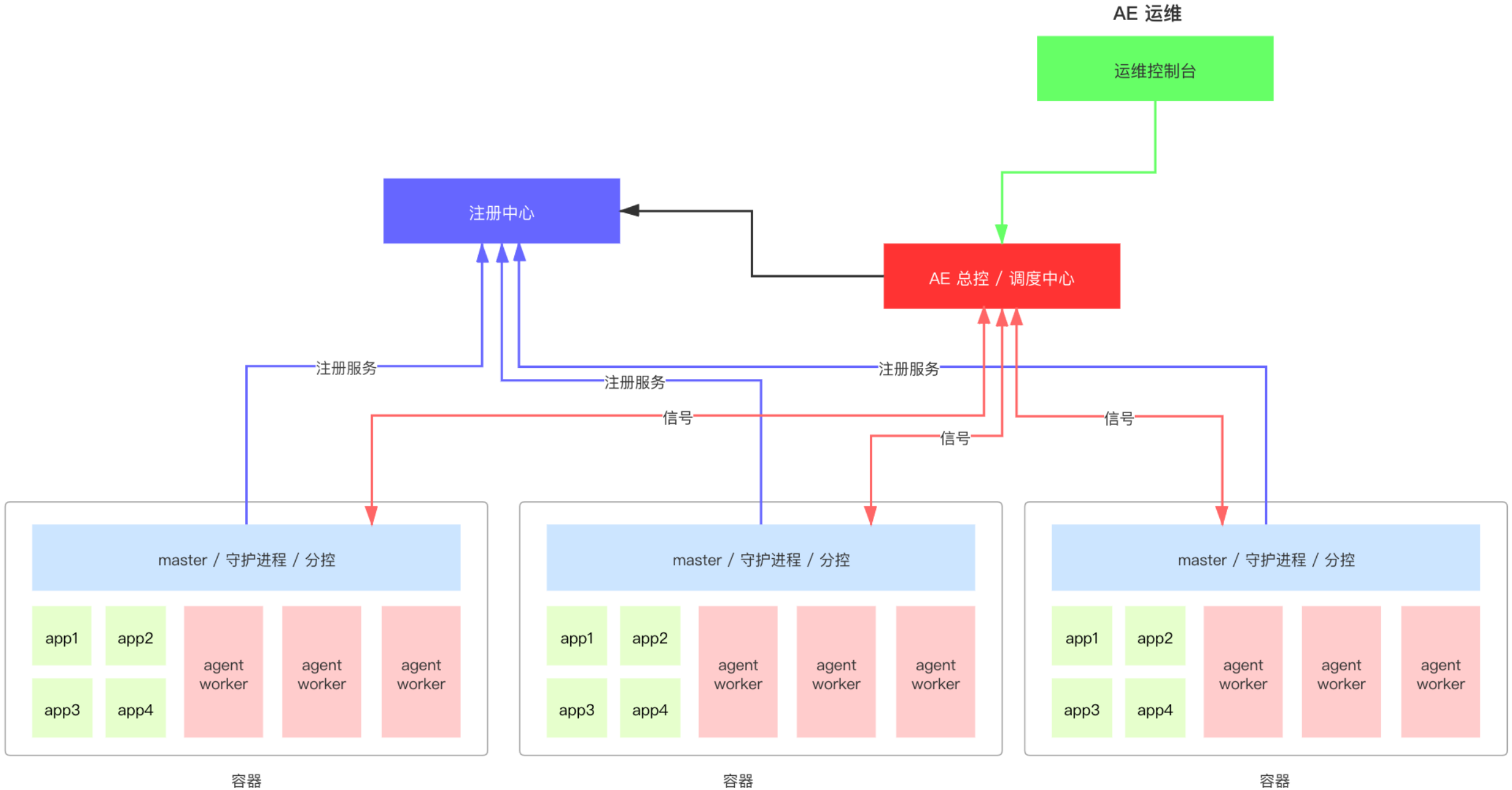
- 应用启动、重启、停止、上线、下线
- 应用服务状态查询
- 应用动态配置下发

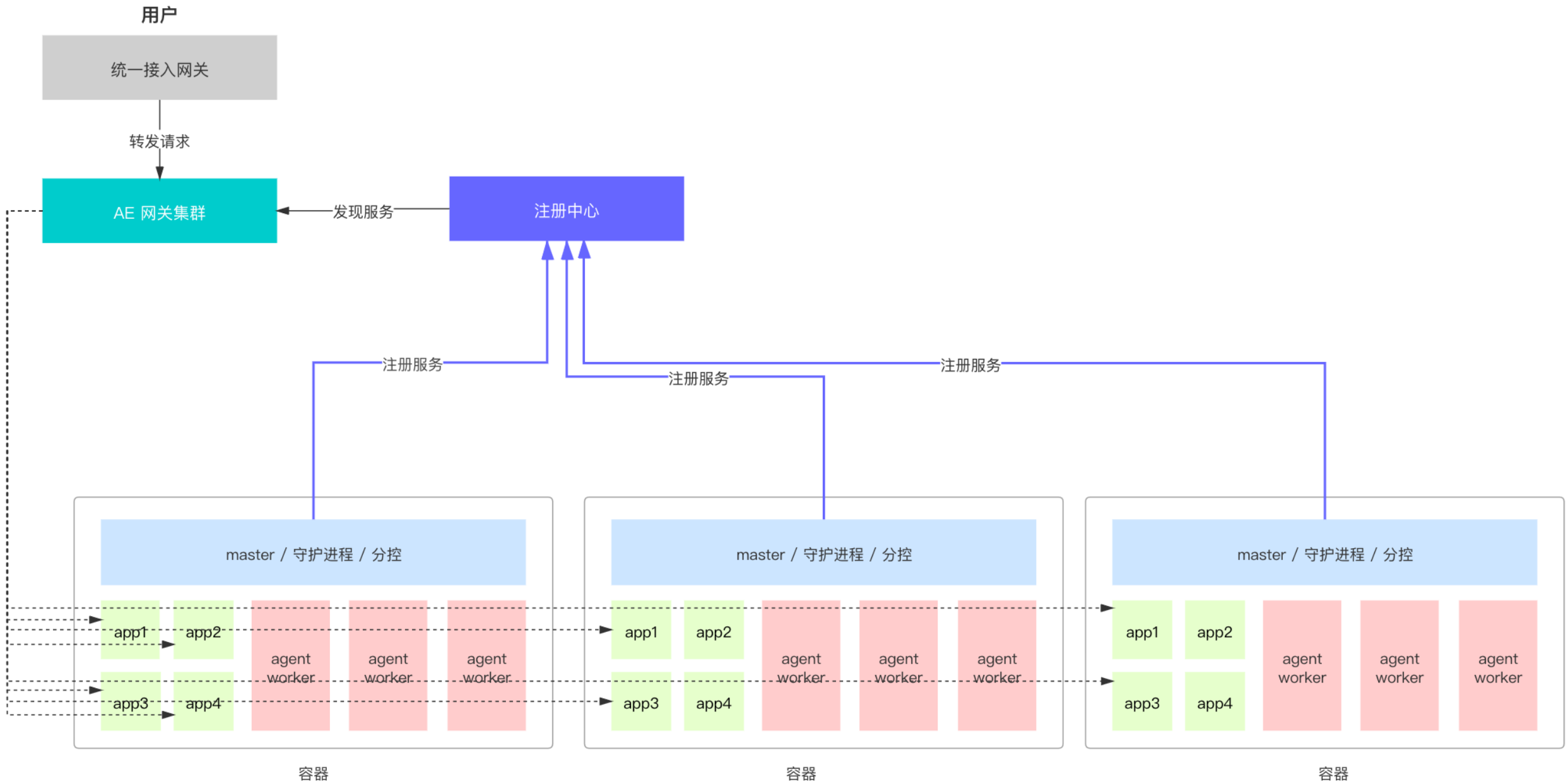

开发者
开发者平台
注册中心
AE 总控 / 调度中心
注册服务
注册服务
注册服务
信号
信号
信号
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
容器
容器
容器


用户
开发者
AE 运维
统一接入网关
开发者平台
运维控制台
转发请求
AE 网关集群
发现服务
注册中心
AE 总控 / 调度中心
注册服务
注册服务
注册服务
信号
信号
信号
master / 守护进程 / 分控
master / 守护进程 / 分控
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
app1
app2
app3
app4
agent worker
agent worker
agent worker
app1
app2
app3
app4
agent worker
agent worker
agent worker
容器
容器
容器

AE 运维会有哪些操作?

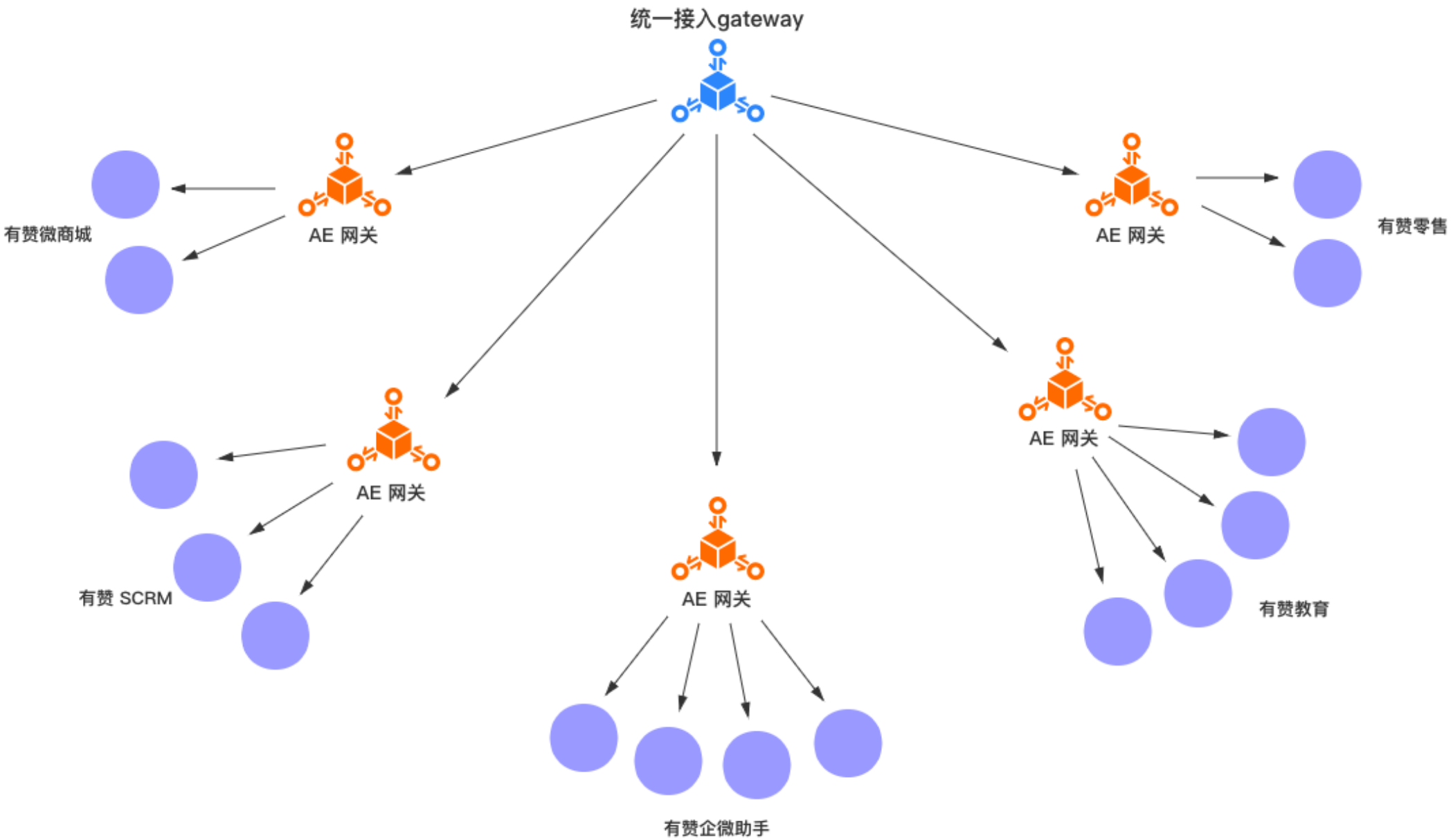
- 应用启动、重启、停止、上下、下线
- 应用扩容、隔离


AE 运维
运维控制台
注册中心
AE 总控 / 调度中心
注册服务
注册服务
注册服务
信号
信号
信号
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
容器
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
容器
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
容器

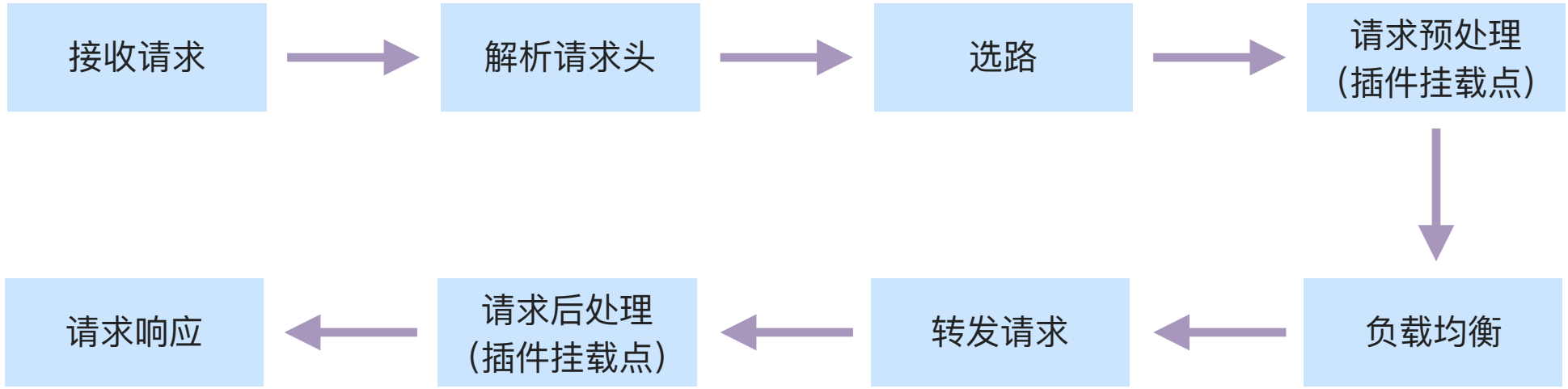

用户
统一接入网关
转发请求
AE 网关集群
发现服务
注册中心
注册服务
注册服务
注册服务
master / 守护进程 / 分控
master / 守护进程 / 分控
master / 守护进程 / 分控
app1
app2
app3
app4
agent worker
agent worker
agent worker
app1
app2
app3
app4
agent worker
agent worker
agent worker
app1
app2
app3
app4
agent worker
agent worker
agent worker
容器
容器
容器

系统组件	描述
开发者平台	应用引擎产品服务，直接面向终端用户，也就是开发者。
运维控制台	平台运维产品端，查看 AE 内部的各种详细状态，并执行相应的运维操作。
调度中心（总控）	AE 大脑，负责整个 AE 的核心状态维护、调度策略生成和计算、工作负载调度以及各个子系统的管理。
宿主控制器（分控）	与总控建立长连接，接收总控下发的信号，并反馈自身以及节点和实例的状态。
注册中心	注册中心，负责维护系统各种元数据，包括宿主、实例状态等。
网关	数据面入口，业务流量网关，维护一个动态路由表，将进入 AE 的请求进行检查、预处理、选路、负载均衡以及转发和日志。

多层 gateway 架构



网关数据 pipeline



网关设计模式

01

请求路由

服务调用者不需要知道服务的地址，全部交给网关处理

02

服务发现

当服务注册有变更时，通过服务发现及时更新服务信息

05

灰度发布

对新版本服务实例进行灰度验证

03

负载均衡

网关需要做负载均衡策略，Round-Robin 轮询 + 权重

04

弹力设计

异步、重试、幂等、流控、熔断、监视

06

安全设计

Session 验证、授权、数据校验

网关的设计重点

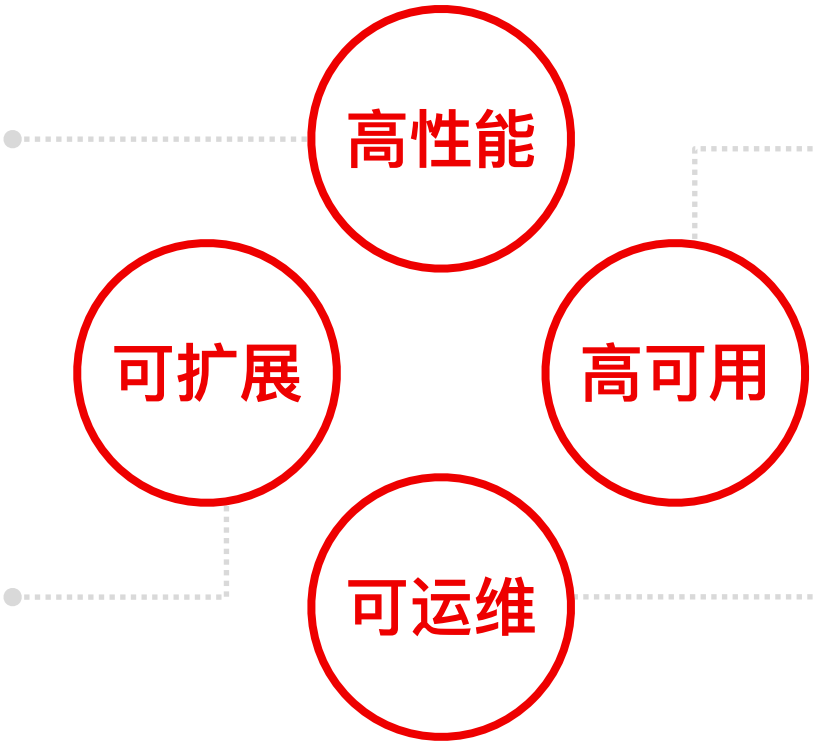
高性能

在架构层面，网关不能成为性能瓶颈，使用异步非阻塞的 I/O 模型。

- Linux epoll
- Windows I/O Completion Port
- Netty、Spring Reactor 的 NIO

可扩展

网关需要承接所有的流量和请求，还需要在网关上加入一些业务逻辑，所以，网关需要支持插件扩展，方便业务进行二次开发。



高可用

网关的稳定性直接决定了所有服务的可用性，所以网关必须是一个高可用的组件。

- 集群化
- 热更新

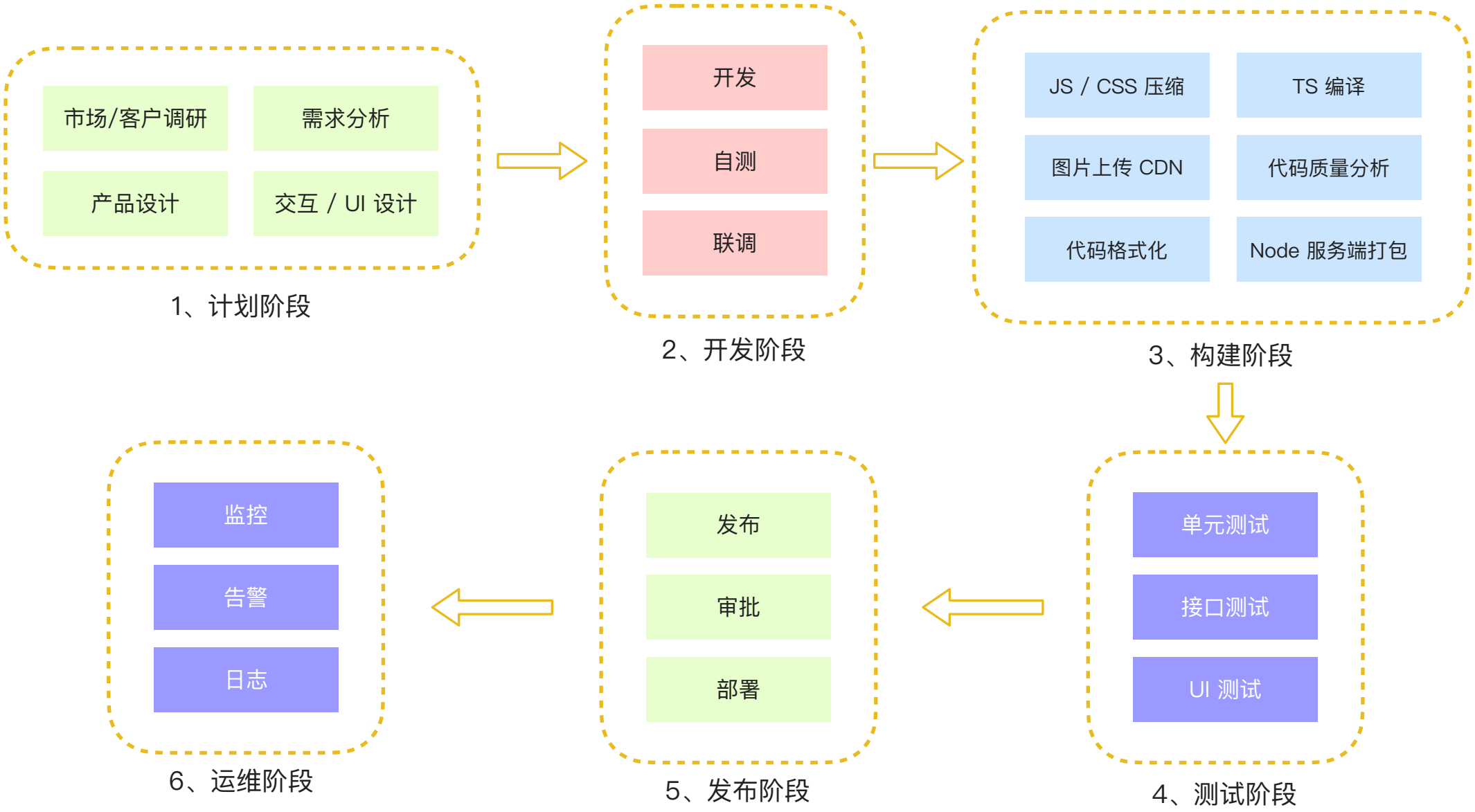
可运维

- 应用监视：提供应用监控数据
- 弹力设计：保护后端服务
- DevOps：提供自动化运维工具

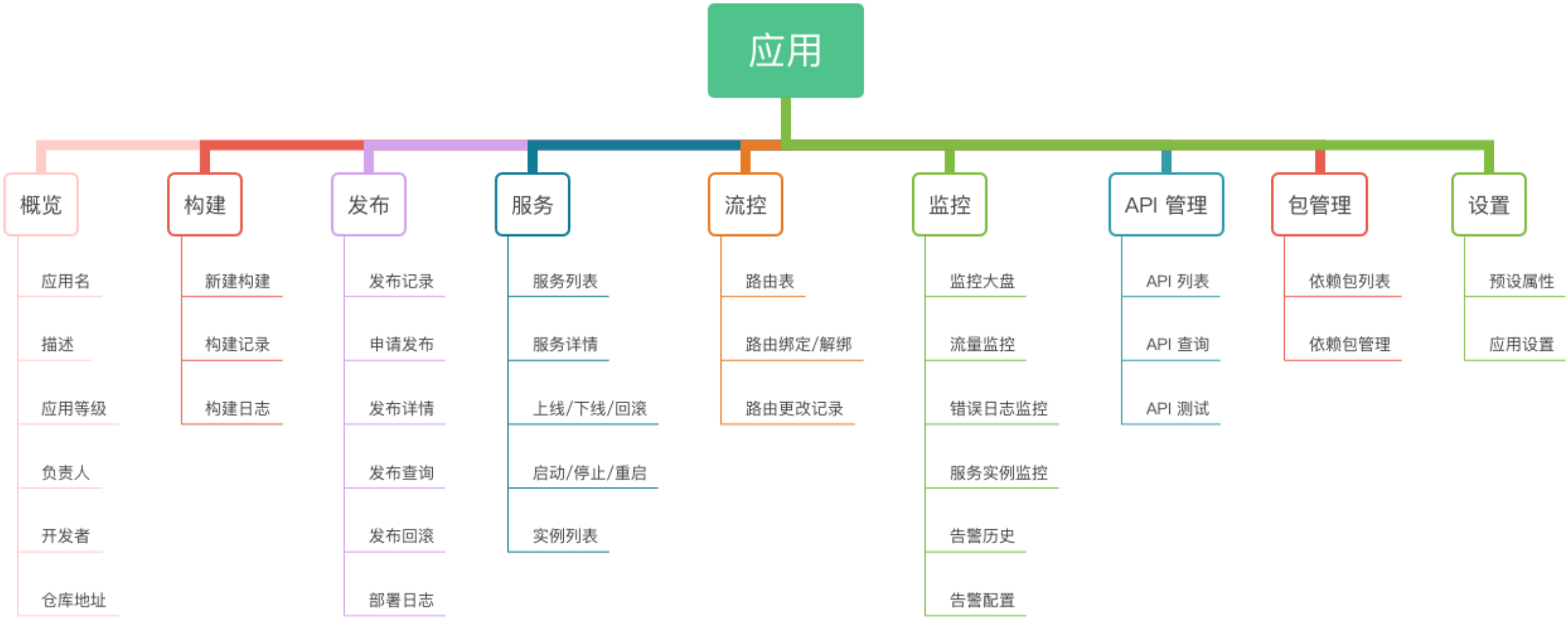
Part 3：应用引擎产品设计

应用引擎产品定位是怎样的?

前端产品研发流程


市场/客户调研
需求分析
产品设计
交互 / UI 设计
1、计划阶段
开发
自测
联调
2、开发阶段
JS / CSS 压缩
TS 编译
图片上传 CDN
代码质量分析
代码格式化
Node 服务端打包
3、构建阶段
单元测试
接口测试
UI 测试
4、测试阶段
发布
审批
部署
5、发布阶段
监控
告警
日志
6、运维阶段

应用引擎是面向前端提供的集开发、测试、部署、运维及监控为一体的开发者平台。


应用
概览
应用名
描述
应用等级
负责人
开发者
仓库地址
构建
新建构建
构建记录
构建日志
发布
发布记录
申请发布
发布详情
发布查询
发布回滚
部署日志
服务
服务列表
服务详情
上线/下线/回滚
启动/停止/重启
实例列表
流控
路由表
路由绑定/解绑
路由更改记录
监控
监控大盘
流量监控
错误日志监控
服务实例监控
告警历史
告警配置
API 管理
API 列表
API 查询
API 测试
包管理
依赖包列表
依赖包管理
设置
预设属性
应用设置

box-demo 概览 构建 发布 服务 流控 设置

申请发布

环境： 生产环境 预发环境 测试环境

类型： 请选择发布类型

发布人： 选择负责人

版本： 请选择应用版本

状态： 请选择状态

查 询 重 置

发布记录

版本	commit	发布人	环境	类型	状态	发布时间	操作
1.0.3	[redacted]	[redacted]	prod	灰度	完成上线	2021-05-13 17:27:47	发布详情 | 回 滚
	[redacted]	[redacted]	prod	灰度	完成上线	2021-05-12 21:49:43	发布详情 | 回 滚
1.0.2	[redacted]	[redacted]	prod	标准	发布取消	2021-05-12 18:03:16	发布详情
1.0.2	[redacted]	[redacted]	prod	回滚	发布成功	2021-05-12 17:36:16	发布详情
1.0.2	[redacted]	[redacted]	prod	灰度	完成上线	2021-05-12 17:30:15	发布详情 | 回 滚