

# 多行业的 DevOps 落地经验分享

王津银

运维老王, DevOpsMaster TTT讲师

2018 中国·北京

2005. 5

广州普信科技有限公司.  
BOSS系统开发

腾讯公司

前端/数据存储运维负责人

2007. 5

2012. 12

广州YY.

业务运维负责人

广州UC.

游戏运维+运维研发负责人

2014. 3

2015. 6

优维科技CEO.

DevOps管理专家

2018 DevOpsDays | 北京站



- 精益运维发起人，运维老王
- DevOps Master TTT讲师

# 目录

## CONTENTS

01

### 关于行业与 DevOps

不同行业对 DevOps 落地的要求不一样，但是理解是一致的

02

### 案例：互联网+汽车服务，多供应商交付模式

以汽车制造行业的互联网部门为例，介绍在多供应商交付模式下的 DevOps 实践

03

### 案例：互联网2C服务，特性产品组交付模式

以互联网门户网站服务为例，介绍特性产品组交付模式下的 DevOps 实践

04

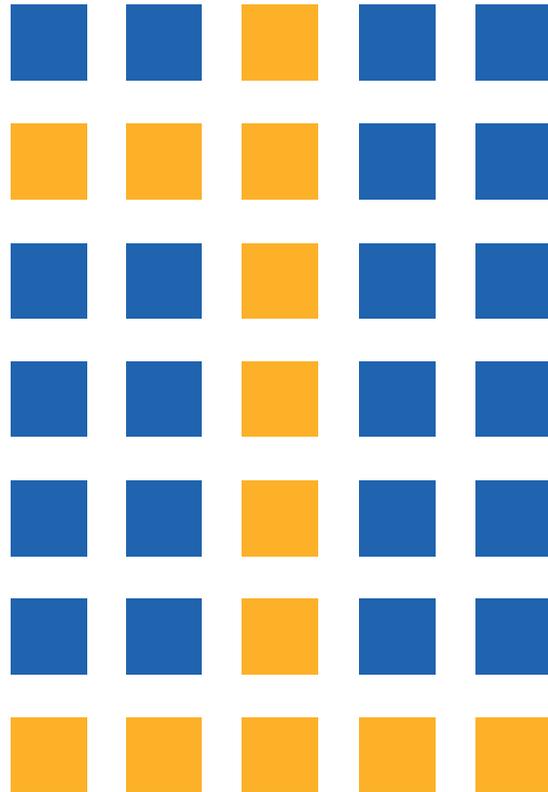
### 案例：证券，跨越传统职能组的交付模式

IT 能力的服务目录化、自动化、规范化，DevOps 与 ITIL 融合

05

### 回顾 DevOps 的本质

通过各个行业的案例，总结 DevOps 的本质



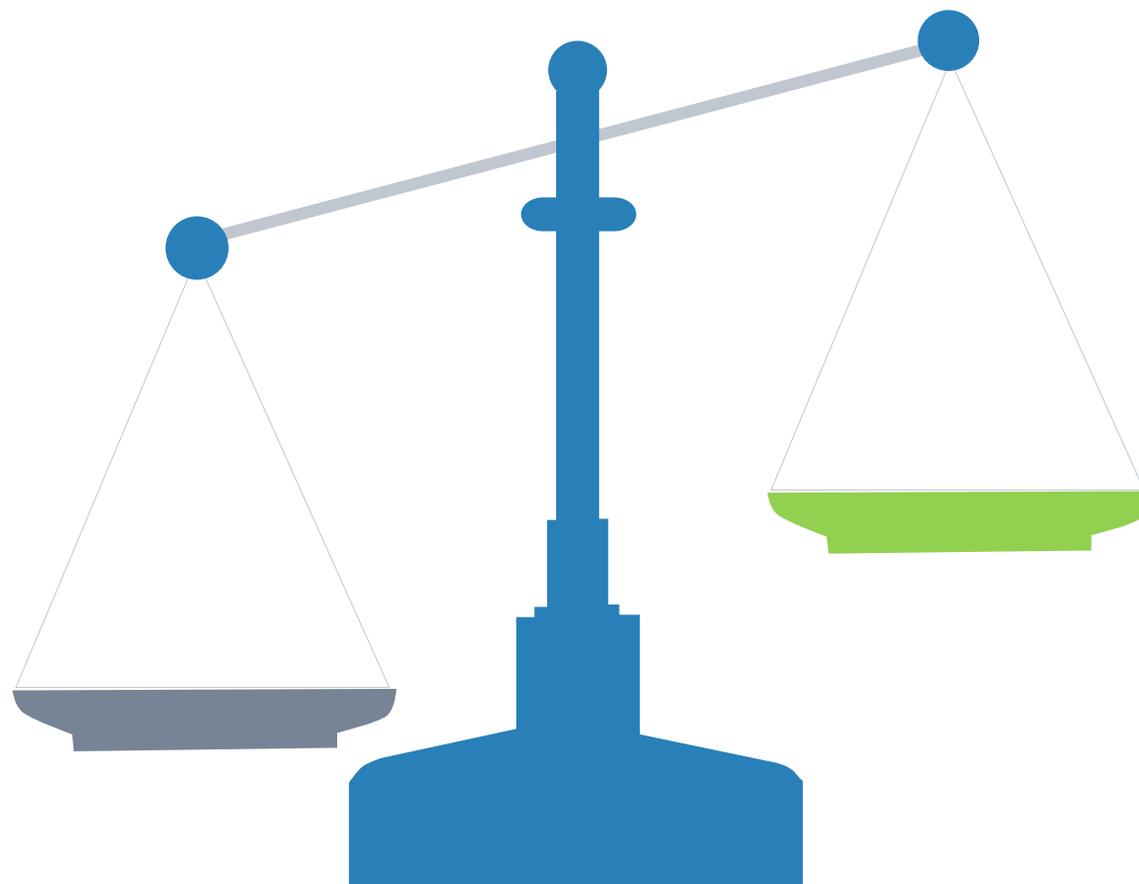
## 关于行业与 DevOps

---

# 不同行业在稳态和敏态间的综合衡量

## 稳态管理

以 ITTL 理念为核心，强调流程和规范、安全和稳定，维护好原有的IT基础设施持续提供服务



## 敏态运营

以 DevOps 理念为核心，通过促进开发、测试和运维团队之间的沟通和协作，打造IT服务完整的生命周期交付链

# 不同行业IT交付特征的差异之处



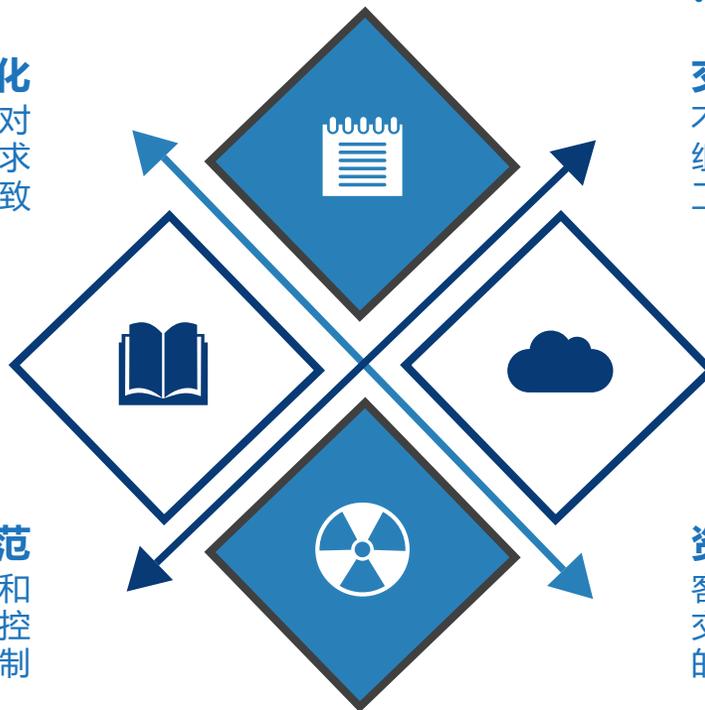
## 产品需求 \* 市场变化

不同行业不同服务面临的客户群体对象不一样，市场变化节奏不同，需求稳定性不一致



## 交付团队 \* 工作模式

不同行业内部的 IT 服务交付团队在组织模式上各有差异，凸显出团队的工作模式的不一致



## 发布需求 \* 运营规范

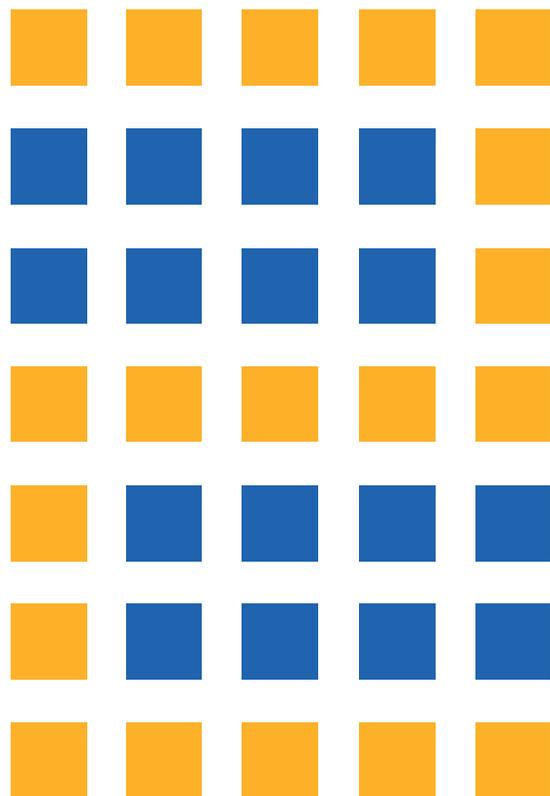
客户群体影响服务规模，变更要求和运营规范，特别是线上运营和发布控制



## 资源需求 \* 环境管理

客户需求的稳定性和迭代频率影响了对交付环境的要求，对资源的自助式服务的需求





## 案例：互联网+汽车服务，多供应商交付模式

---

# 企业画像：业务需求



## 主营业务

互联网部门为汽车提供物联网服务，BS架构



## 需求特征

需求生命周期相对固定，交付频率不高，随新车型进行需求变更



## 需求供应

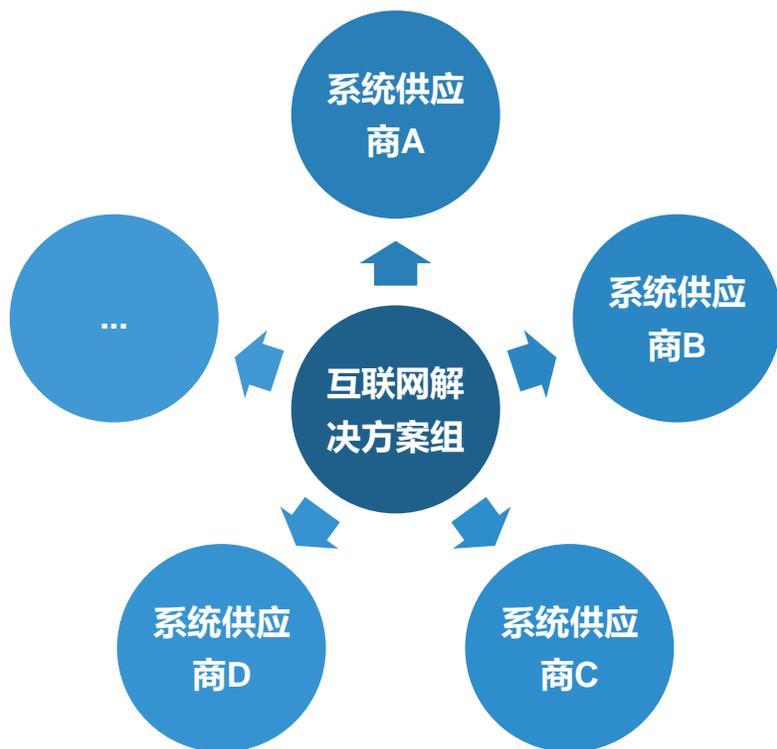
多种供应商，长期稳定的开发合作伙伴



## 用户群体

遍布世界各地的汽车用户

# 企业画像：IT服务交付团队



## □ 互联网解决方案组

为开发合作伙伴（供应商）提供技术帮助，协调需求从开发到测试、上线运营的节奏，控制整个系统交付的生命周期。

在不同的行业内，可能管理组的名称不一样。

## □ 供应商

长期稳定的开发合作伙伴，与解决方案组协作，负责开发各项汽车互联网的软件服务，不同的系统负责的供应商不一样。

# 企业画像：资源和环境需求



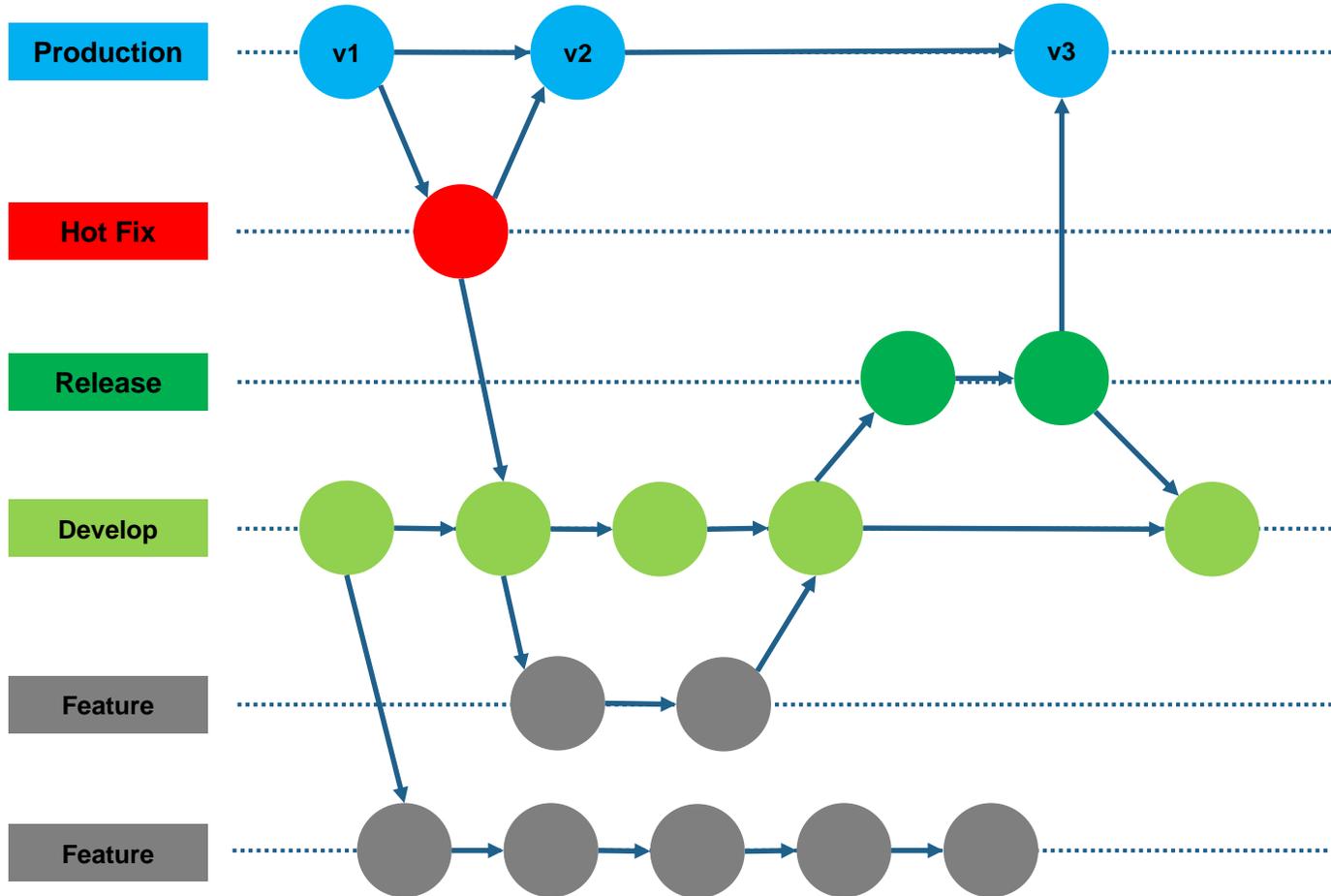
稳定的需求，固定的开发周期，用于开发特性的分支环境也相对稳定，不会经常新建或者销毁

众多的供应商协同企业开发各项IT服务，对开发资源、环境管理和发布运营期望得到自助式的服务

因为需要为众多的不同厂商的开发伙伴提供技术支持，因此，在技术栈上需要有一定的准入限制，避免开发失控

发布规模适中，发布频率不高，但是对安全和稳定，用户体验有较高要求，注重发布成功率和回滚速度

# 解决方案：统一分支管理和版本管理

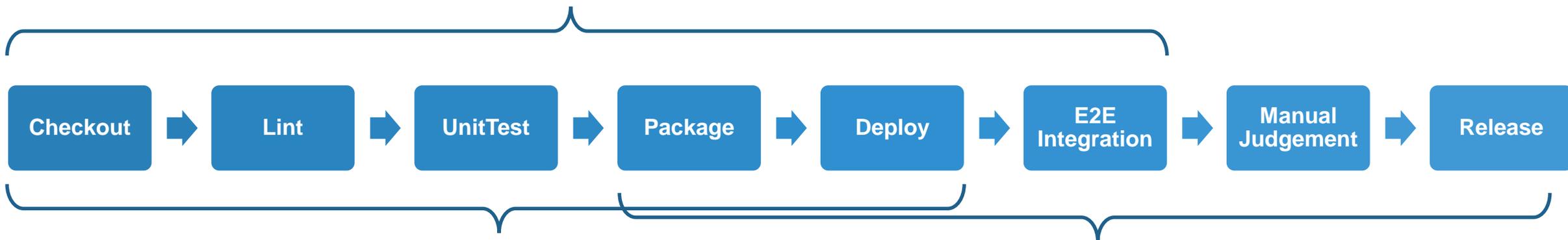


- 01 用户需求分解为特性开发分支(Feature Branch)**  
 经过分解，研发中的用户需求会从 Development 分支中创建特性分支进行开发，多个用户需求的特性分支互不影响，开发完毕合并到 Develop 分支中，Feature 环境开发完毕后销毁
- 02 集成验收时 QA 介入(Release Branch)**  
 在 Feature 期间，QA 已经在编写集成用的测试用例，发起 Release 分支时，QA 识别为提测信号，在 Release 分支上对即将发布的应用进行集成，有且只有一个 Release，并且不允许在 Release 分支中加入新的功能
- 03 标签发布 (Production/Master)**  
 Release 完成时，分支被删除，并且合并到 Master 和 Development 分支，同时在 Master 上更新标签(Tag: V1/V2/...), 此标签名称为开发、测试和运维共同识别的版本号，线上发布使用此版本的代码
- 04 问题修复 (Hot Fix)**  
 当线上出现问题时，直接在最新的 Master 分支上拉起 hotfix 分支，修复问题，完成后和 Release 分支类似，代码合并到 Master 和 Development 分支，同时在 Master 上更新标签

# 解决方案：统一分支管理和版本管理

## 基于 Git-Flow 的持续集成方案

### 发布测试流水线(Code Push)



### 特性开发流水线(Code Push)

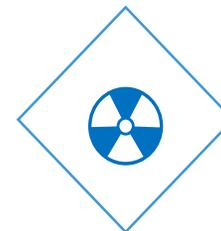
### 发布流水线(构件库发布)



01

#### GitLab WebHook

通过配置 GitLab WebHook 来监听流水线事件



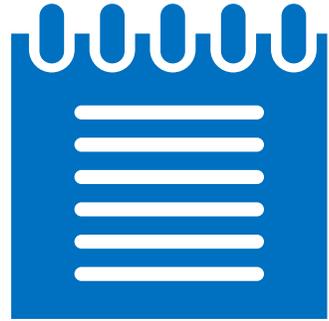
02

#### Branch Based Pipeline

不同的分支执行不同的 Stage, 例如:

- Feature分支: Stage Build
- Release分支: Stage Build + Integration

# 解决方案：基于 Application 的IT资源管理



应用的资源画像



## 基本信息

- 名称
- 负责人
- 备注信息
- ...



## 集群管理

- 开发环境
- 测试环境
- 生产环境
- ...



## 部署资源

- 程序包
- 配置包
- 文件包
- ...



## 构建信息

- 开发语言
- 开发框架
- 代码库地址
- ...



## 网络资源

- 负载均衡
- 内网域名
- 公网域名
- ...

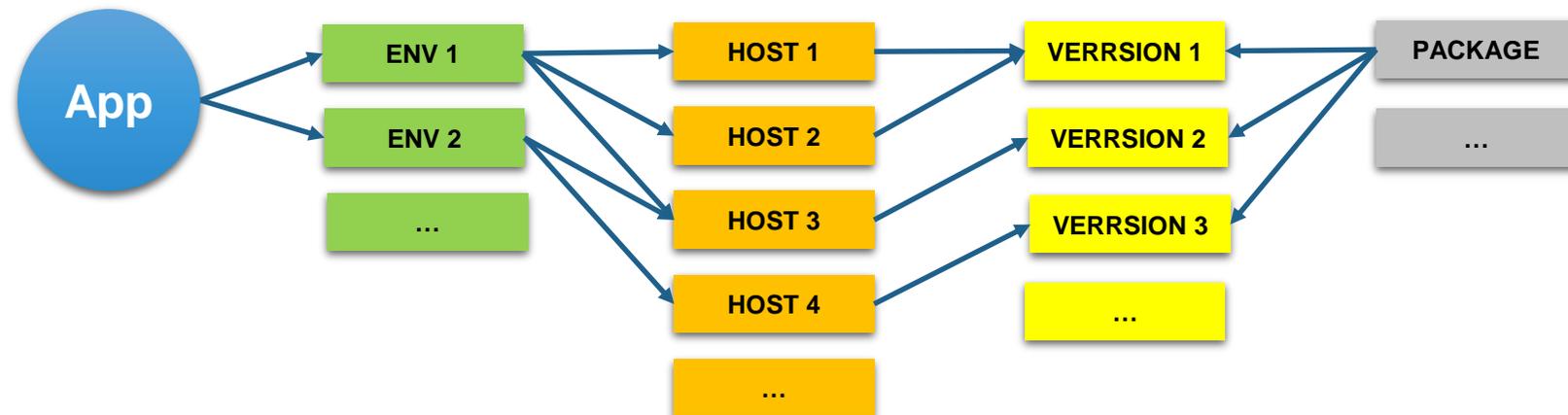


## 环境依赖

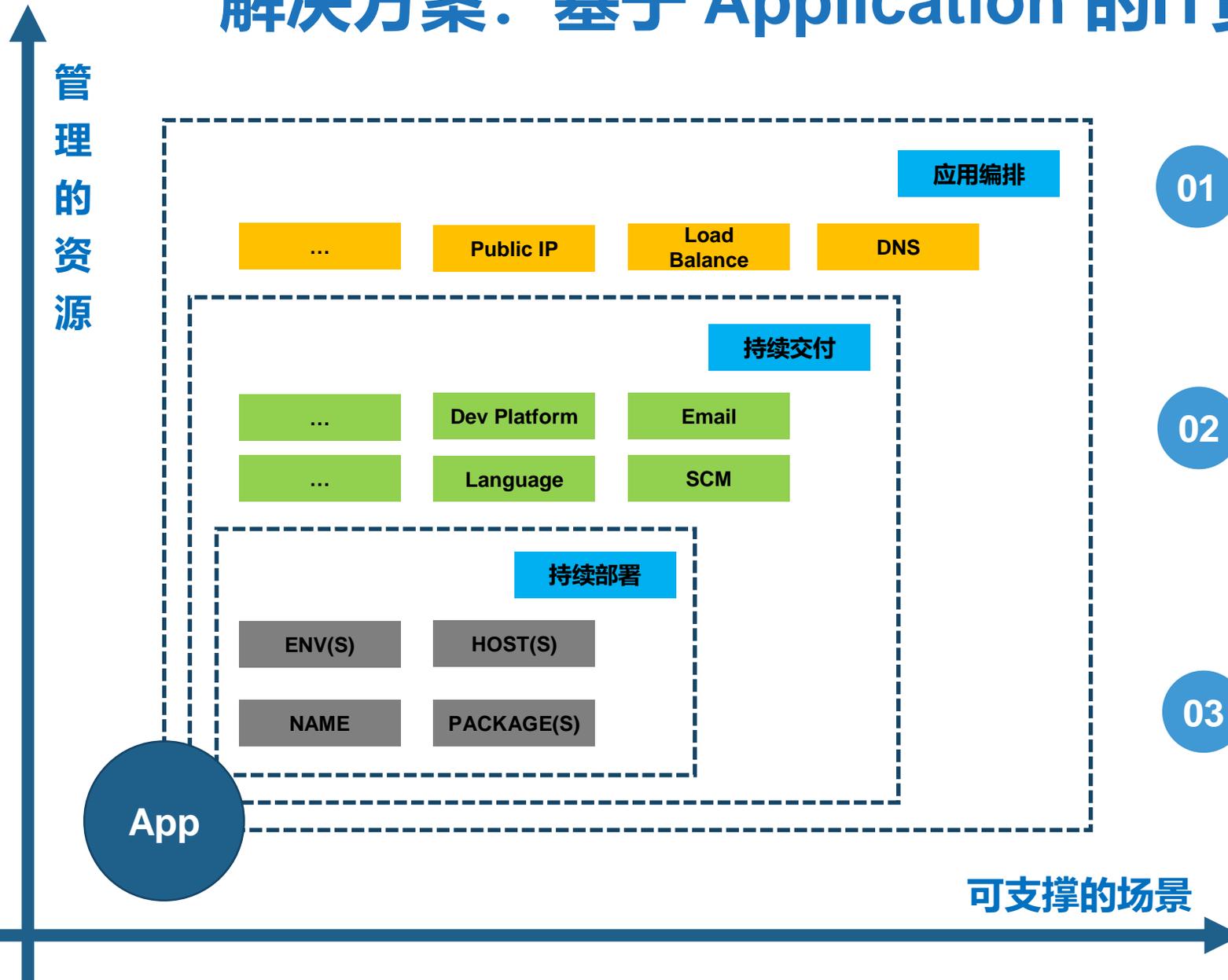
- 基础镜像
- 中间件
- ...



应用的资源拓扑



# 解决方案：基于 Application 的IT资源管理



01

## 抽象IT资源为资源对象

- 把一切的IT资源视为资源对象
- 对象拥有属性和关系

02

## 管理核心对象

- 持续交付的核心对象为
- Application (应用)
  - Environment (环境)
  - Host (主机)

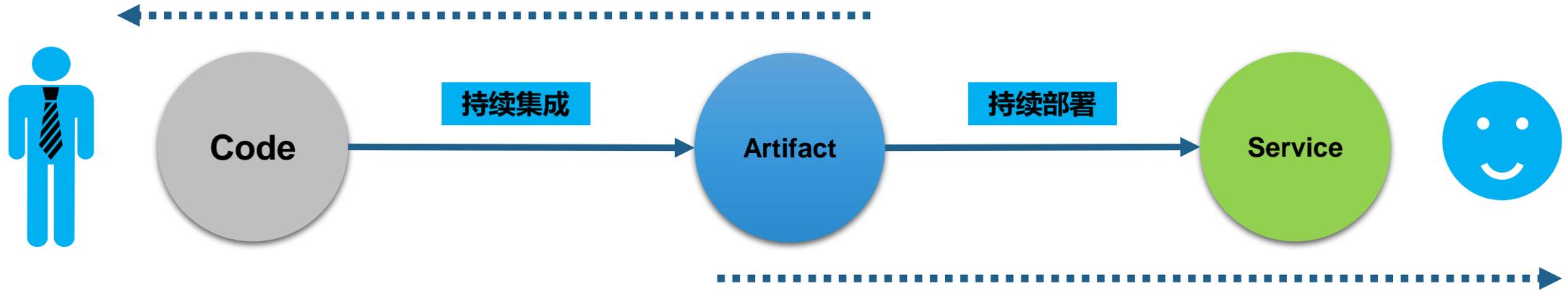
03

## 梳理扩展对象

随着支撑的场景来拓展纳管的资源对象

# 解决方案：基于 Artifact 为核心的持续交付

开发效率（基于SCM资源对象为核心的构建管理）



用户体验（基于Service资源对象为核心的服务管理）



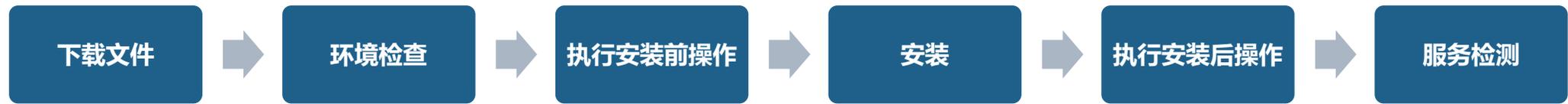
Artifact 的生命周期

# 解决方案：基于 Artifact 为核心的持续交付

## 对 Artifact 生命周期的场景进行标准化落地

□ Artifact 部署类型：升级（不同版本间的切换）、安装、卸载

□ Artifact 安装流程



□ Artifact 升级流程

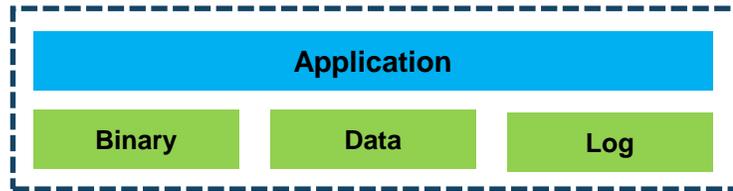


□ Artifact 卸载流程

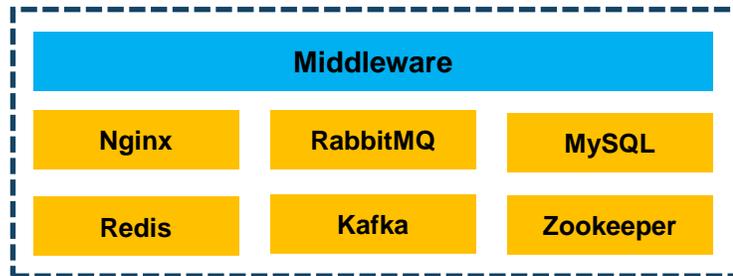


# 解决方案：基于 Artifact 为核心的持续交付

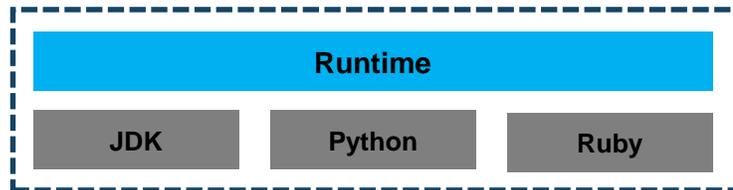
## 对 Artifact 的运行环境进行标准化落地（物理OS部署）



示例项目	cmdb
BASE	/middleware/easyops/apps
HOME	/middleware/easyops/apps/cmdb
日志和数据	/middleware/easyops/data/cmdb
用户 (组)	wlsadmin:wlsadmin



示例项目	Apache Tomcat, Version 9.0
BASE	/middleware/easyops/softwarewares
HOME	/middleware/easyops/softwarewares/linux_apapche_tomcat_9.0
DATA&LOG	/middleware/easyops/data/linux_apapche_tomcat_9.0
USER	wlsadmin:wlsadmin

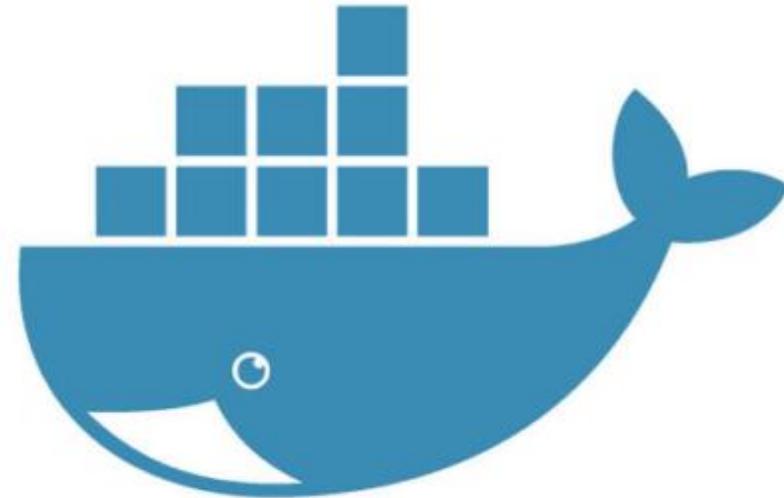
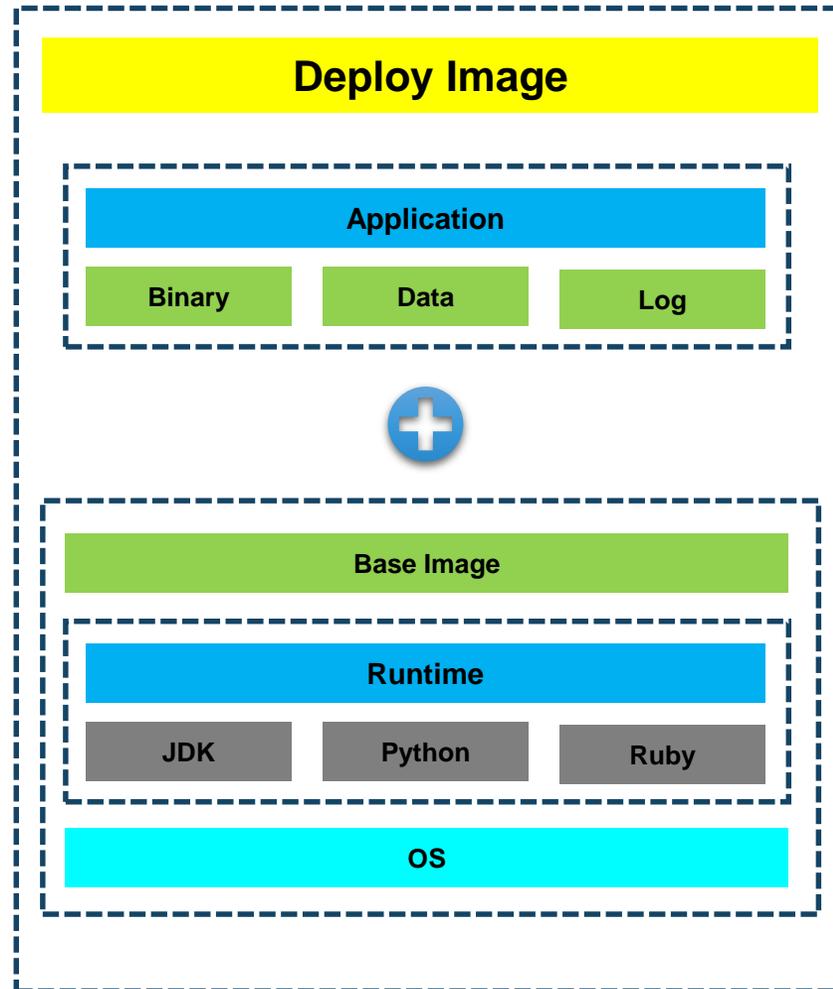


EXAMPLE	SunSpot JDK, Version 1.8
BASE	/middleware/easyops/runtime
HOME	/middleware/easyops/runtime/jdk_1.8
USER	easyops:easyops



# 解决方案：基于 Artifact 为核心的持续交付

对 Artifact 的运行环境进行标准化落地 (Docker 集群部署)



# 解决方案：基于 Nginx 流量灰度发布管理



F5 LTM

Global DNS

Nginx

Local DNS

HOST

HOST

HOST

- 用户请求通过 Global DNS 到达 F5 LTM
- App 拥有自己的内网 DNS，注册到 Nginx 配置文件中
- Nginx Backend 分为蓝绿区域，配合发布流水线，通过配置文件进行流量的自动切换

[conf.d]  
Nginx 蓝绿区域模板

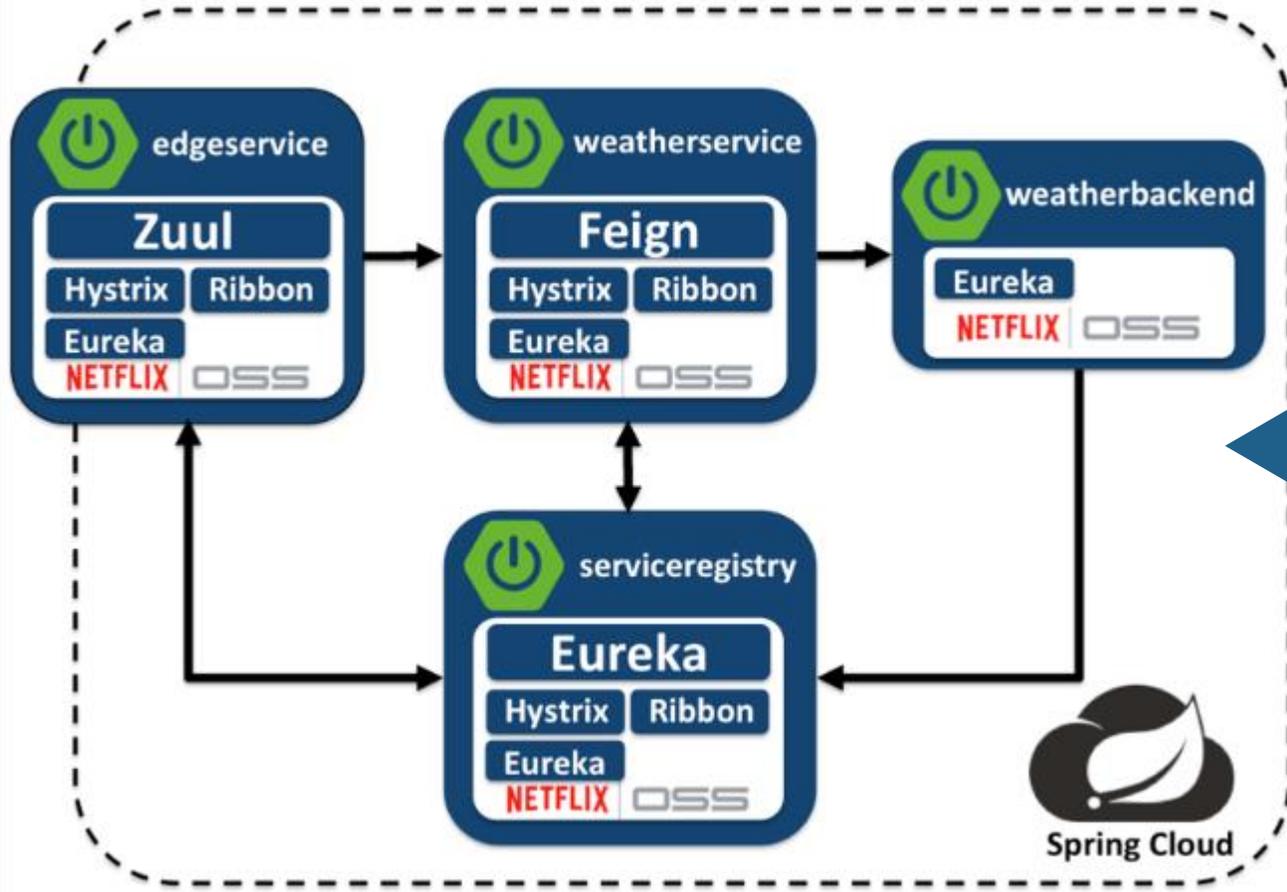
`${APP_NAME}-ALL.conf.template`

`${APP_NAME}-BLUE.conf.template`

`${APP_NAME}-GREEN.conf.template`

编号	变量	部署路径	用途
1	NGINX_HOME	/middleware/easyops/software/nginx	NGINX的家目录
2	CONF_PATH	/middleware/easyops/software/nginx/conf	Nginx配置文件目录
3	TEMPLATE_PATH	/middleware/easyops/software/nginx/conf/conf.d	Nginx区域流量模板文件目录

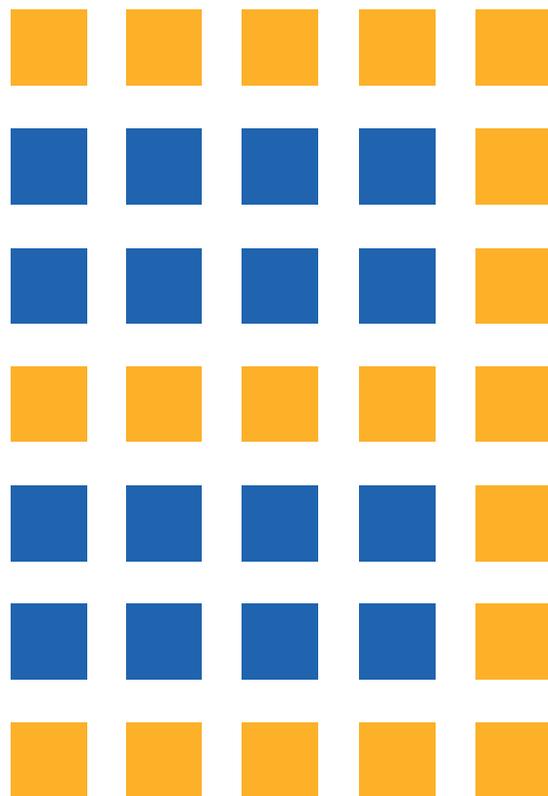
# 解决方案：基于 Spring Cloud 的统一开发



## Spring Cloud Platform

- ❑ 名字服务: Spring Cloud Eureka
- ❑ 负载均衡: Spring Cloud Ribbon
- ❑ 配置中心: Spring Cloud Config Server
- ❑ 链路跟踪: Spring Cloud Sleuth
- ❑ 消息总线: Spring Cloud Bus
- ❑ 路由网关: Spring Cloud Zuul
- ❑ ...

MicroService Framework



## 互联网2C服务，特性产品组交付模式

---

# 企业画像：业务需求



## 主营业务

互联网的门户网站，为广大的用户群体提供海量游戏资讯和周边服务，日UV数千万级



## 需求特征

用户需求不稳定，变更频率高，极具时效性



## 需求供应

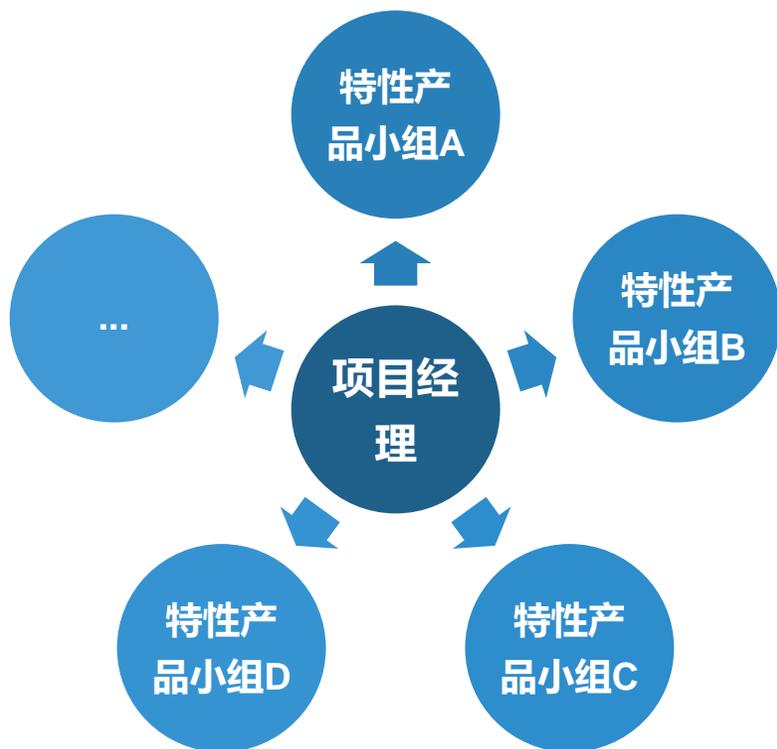
虚拟的特性产品小组



## 用户群体

遍布全国各地的游戏玩家

# 企业画像：IT服务交付团队



## □ 特性产品小组

全职能的虚拟产品服务团队，一般包含产品、UI、前端后端、QA、运维等的完整能力，为产品保驾护航。

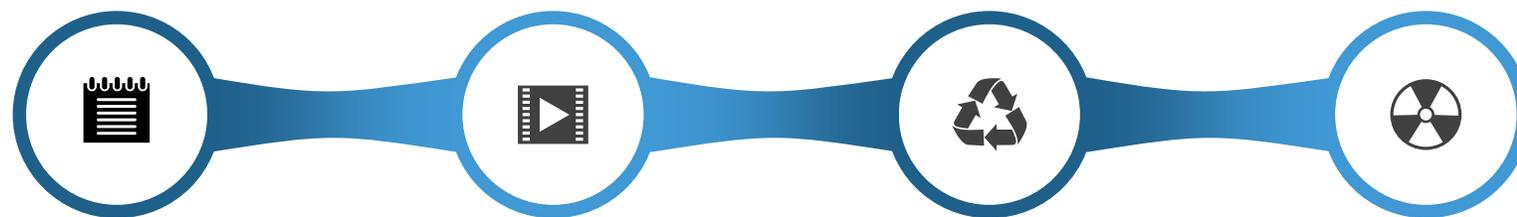
一般情况下，人员空闲时可以自由调配到其他小组。

## □ 项目经理

项目经理需要协调各个特性产品小组的研发进度，人力资源，把控需求交付的周期和风险。

一般情况下，项目经理可以同时管理1到3个左右的特性产品组。

# 企业画像：资源和环境需求



## 频繁变更

海量的玩家群体，需求呈现VUCA的新常态，需求迭代频繁

## 自助式服务

高频率的变更对自助式服务提出更高的要求，从资源申请到环境落地，从持续集成到持续部署，甚至线上发布、监控服务都可采用自助式服务

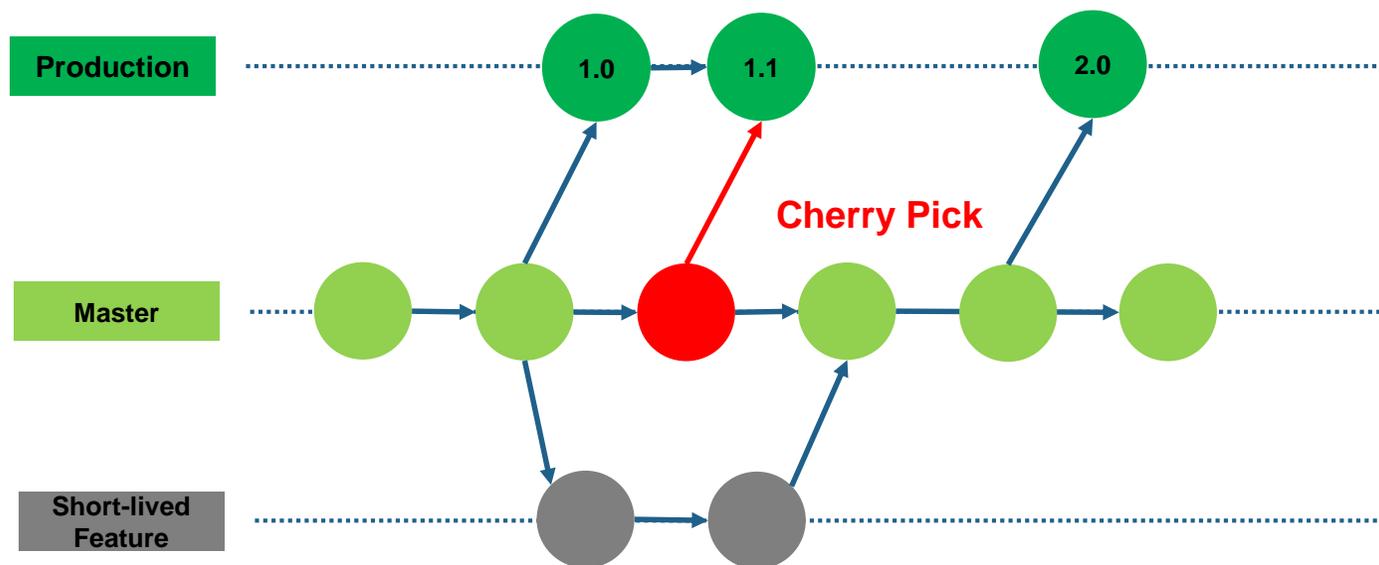
## 异地高可用，提供有损服务

多机房多中心部署，用于业务扩容、容灾备份和提高各个地区用户就近访问的速度，被调服务如有宕机，应该听过服务降级，防止雪崩效应，保持自身稳定

## 发布管理

发布频率高，发布规模大，发布策略多为灰度发布，提供相对范围较小的有损发布，同时需要快速回滚能力

# 解决方案：TBD + 特性开关分支管理



01

## 用户需求细分，主干代码开发

用户需求细分为1到2天的若个研发任务后，研发人员直接在 Master 分支上提交代码，Master 分支会有严格的持续集成，而研发人员的代码也会经过 CodeReview 后，才被合并到 Master 分支

02

## 分支发布 (Release Branch)

拉出发布分支，在上面标记标签，以便识别版本号，对此版本进行深度的集成验收，如果发现有问题，直接在 Master 分支上提交代码后，通过 Cherry Pickup 到发布分支上，更新标签 (版本号)

03

## Short-Lived Feature Branch

根据实际情况，用户的需求也可以被拉出短期的特性分支进行开发，开发完成后通过 MergeRequest 合并到 Master 主干上，这些短期特性分支通常是 task 或者 topic 级别的，并非长期的需求分支，尤其注意他们在合并回主干前，会拉取主干的代码保持自己的更新

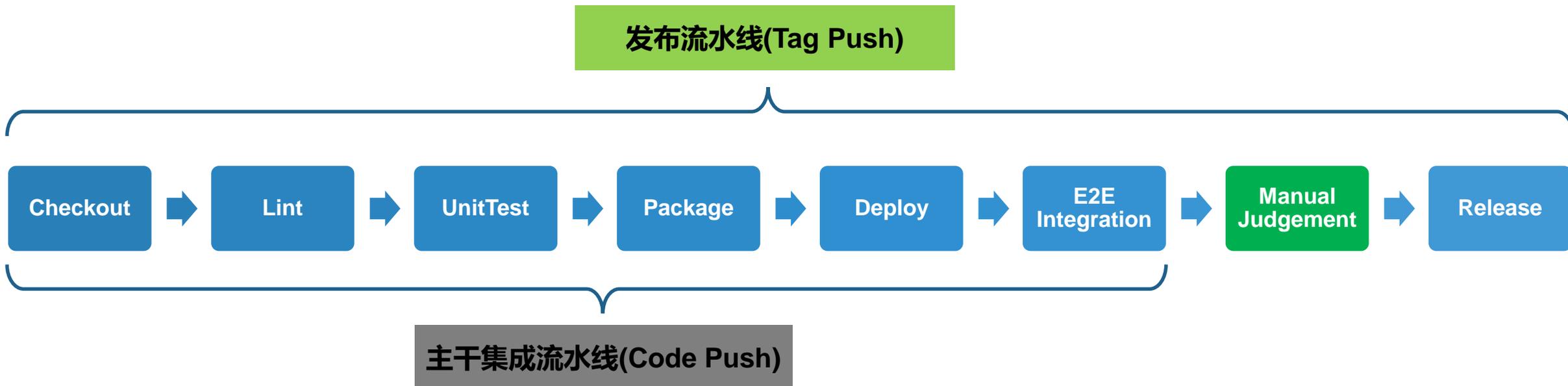
04

## 特性开关 (Feature Flag)

由于功能是直接提交到 Master 分支上了，如果一些功能只开发一半，或者开发完成了但是由于市场策略想延迟对用户的披露时间，那么可以把用户的需求设置为特性开关，控制用户是否可见，一般是在前端控制，此能力非常重要

# 解决方案：统一分支管理和版本管理

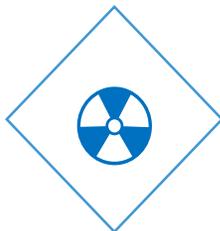
## 基于 Trunk Based 的持续集成方案



01

### GitLab WebHook

通过配置 GitLab WebHook 来监听流水线事件



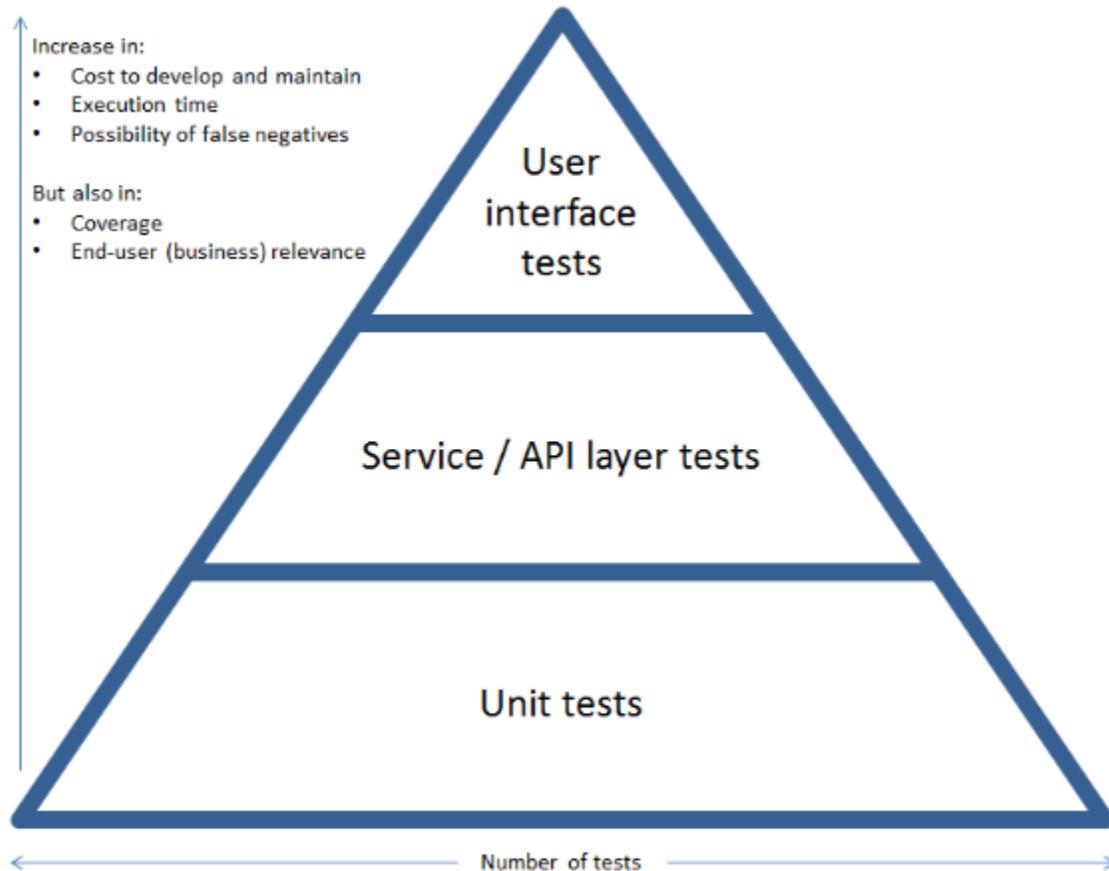
02

### 集成流水线

- 主干集成流水线：通过监听代码提交事件来触发；
- 测试发布流水线：通过监听标签提交事件来触发，QA团队的深度介入和问题修复

# 解决方案：把控质量，测试分层体系和自动化

基于测试分层体系，选择适当的场合，执行适当的测试方案



## 自动化测试收益排行（反馈问题和自动化成本）



从上到下分别为：

- API layer tests
- Service layer tests
- Unit tests
- UI tests



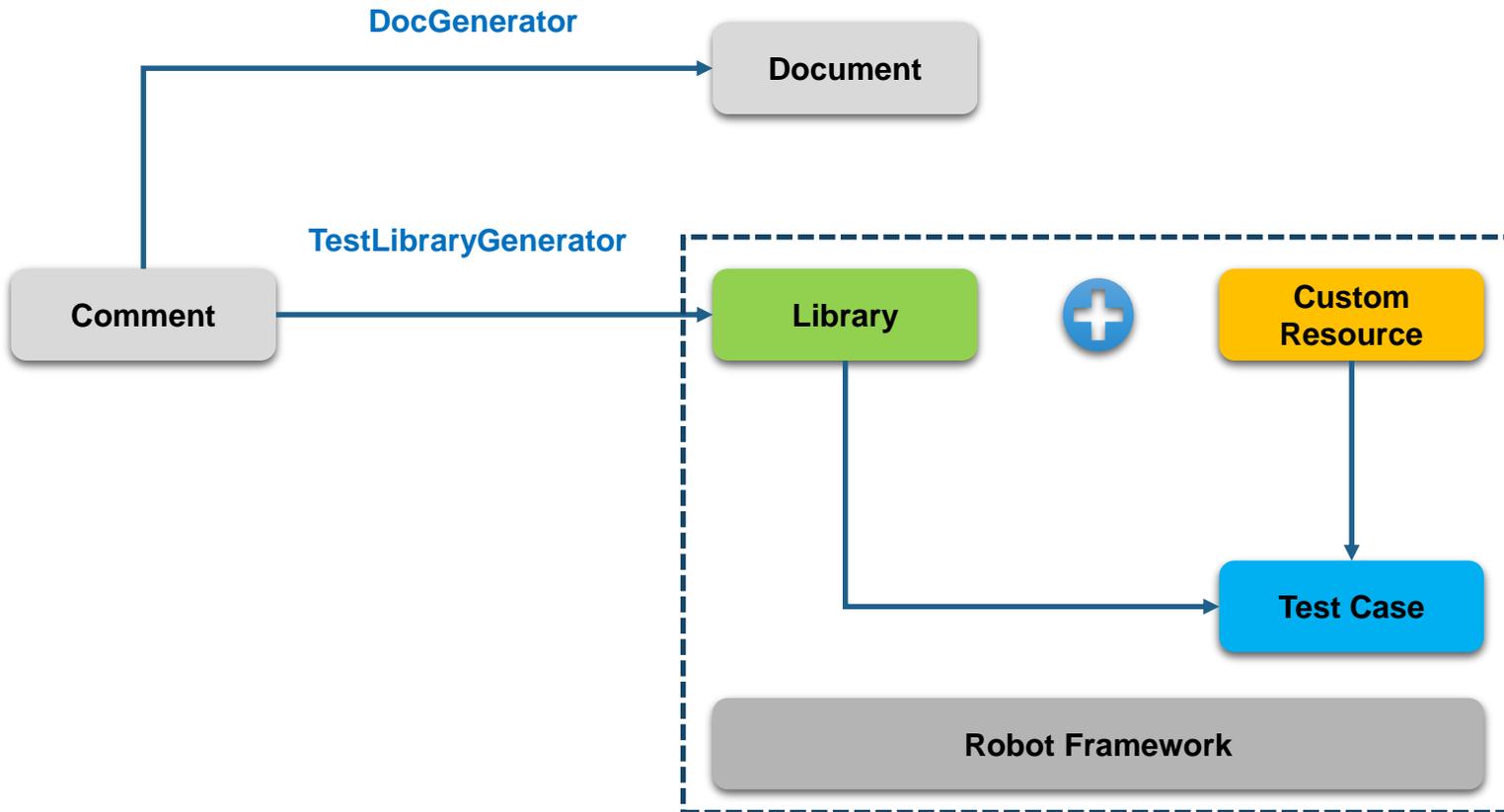
## 何时执行何种测试方案

关键因素：执行耗时（集成成本），综合衡量收本和收益，选择在适当的时机进行相应的测试，灵活调整，例如：

- 单元测试和集成用例在主干持续集成流水线上执行
- UI验收测试在标签发布流水线上执行
- 每次 Hot Fix 必定运行核心的集成用例进行验收

# 解决方案：把控质量，测试分层体系和自动化

## 基于注释(Comment)的全自动化单 API 接口测试



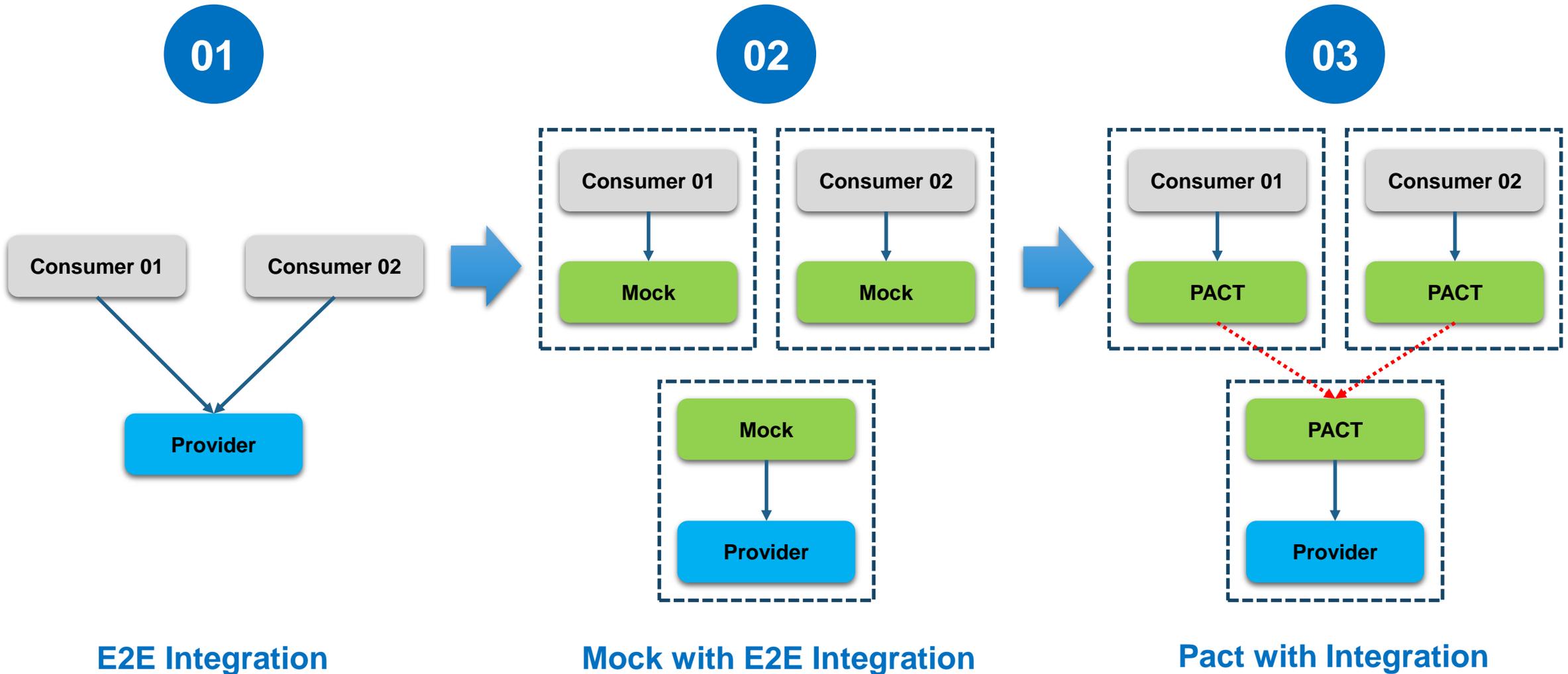
```

/**
 * @name 创建归档版本
 * @url POST /archive
 *
 * @author leon
 * @doc
 * @keyword create archive
 *
 * @param string packageId 包Id
 * @param file file 需要归档的文件
 * @param string message 归档备注说明
 * @param string unzip* 是否自动解压
 * 支持.tar.gz/.tar/.gz/.tgz/.war/.zip文件
 *
 * @return 成功创建的新版本ID, 签名信息, 若有包配置文件也会一并
 返回
 */
  
```

- 基于 Robot Framework 的封装
- 扫描接口注释, 使用关键字 @keywordd 在集成时自动生成测试用例
- 测试用例数据维护于代码库, 跟着版本分支走

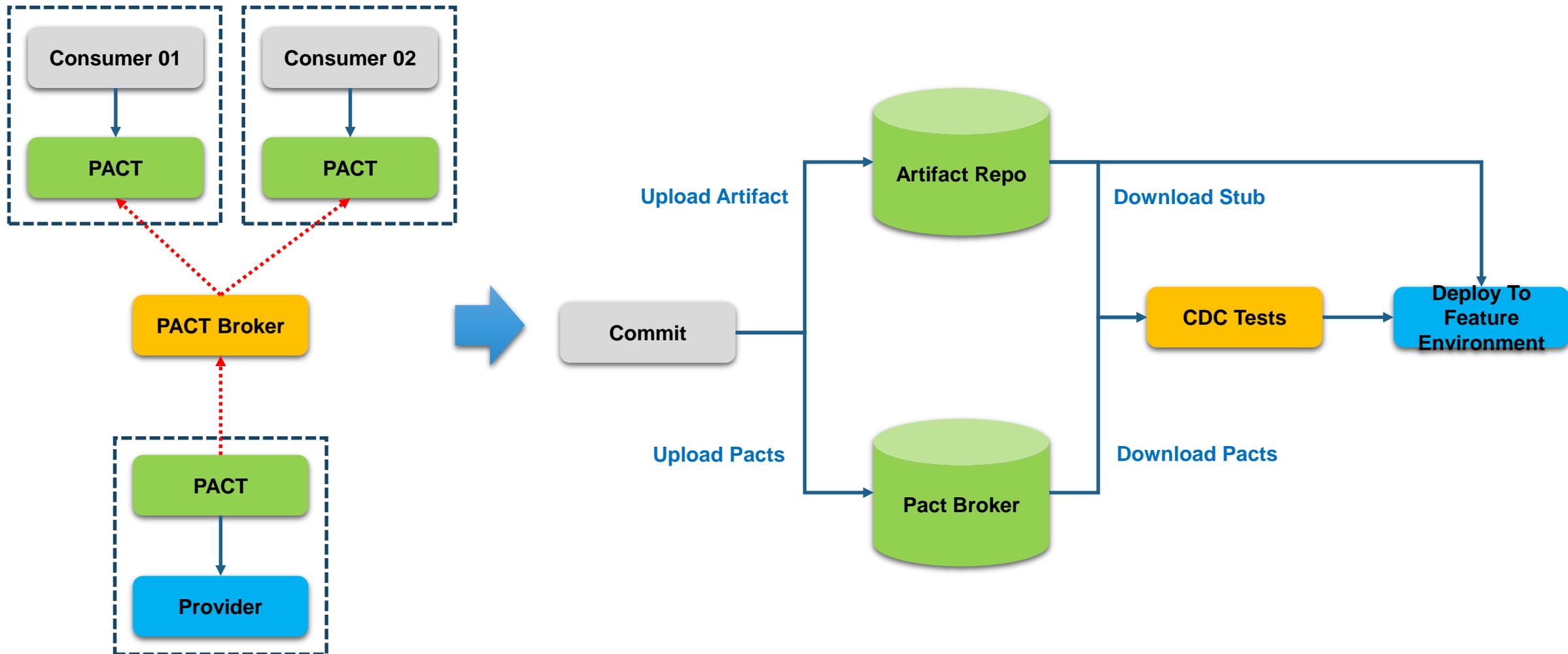
# 解决方案：把控质量，测试分层体系和自动化方案

基于契约测试(Contract Testing)快速构建开发集成环境



# 解决方案：把控质量，测试分层体系和自动化部署

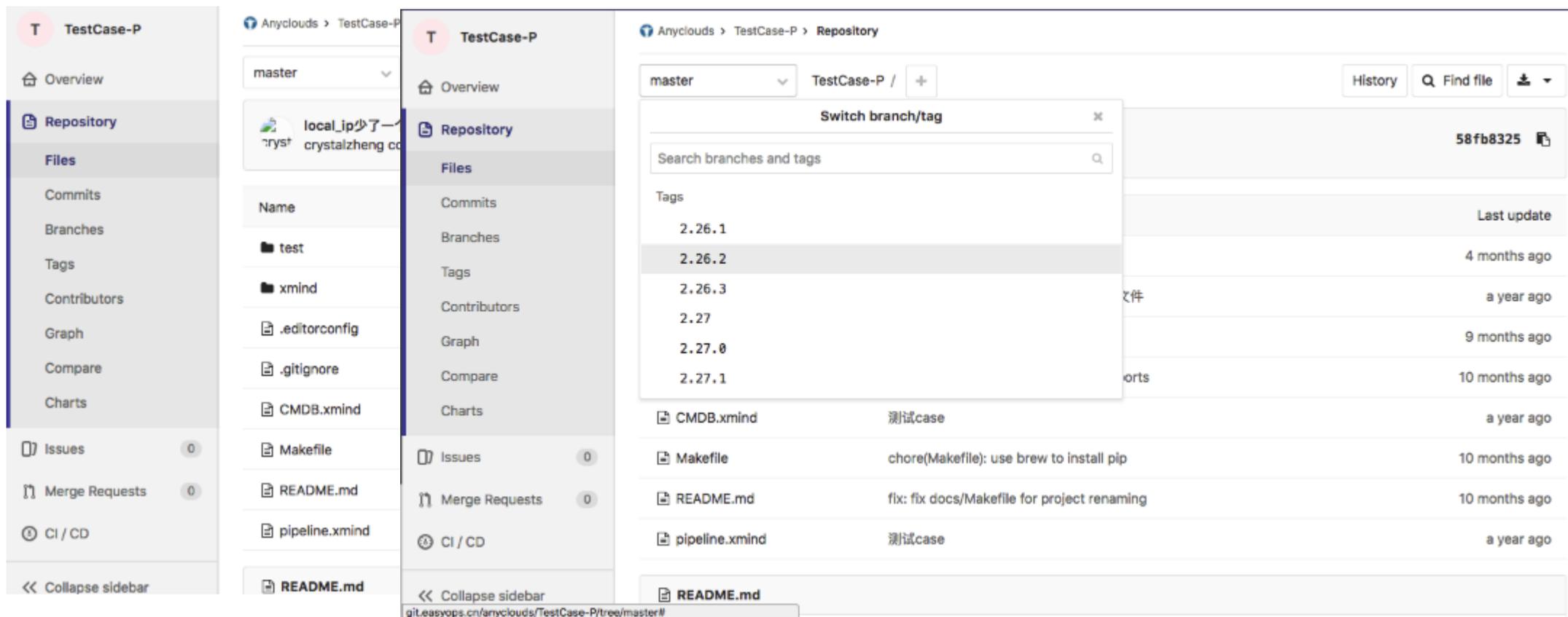
持续集成：Using Pact Broker, Mock With Short-Lived Feature Environment



# 解决方案：把控质量，测试分层体系和自动化

## UI 自动化测试的一些实践经验(1)

- 测试用例应该存放到源代码管理库中，并且版本号和业务系统保持一致
- 业务系统某个版本发布前，拉取版本号对应的配套测试用例集



The screenshot displays the Anyclouds Git interface for a repository named 'TestCase-P'. The interface is divided into three main sections:

- Left Sidebar:** Contains navigation options: Overview, Repository (selected), Files, Commits, Branches, Tags, Contributors, Graph, Compare, Charts, Issues (0), Merge Requests (0), and CI/CD. A 'Collapse sidebar' button is at the bottom.
- Center Panel:** Shows the repository structure under the 'master' branch. It lists files and folders: 'test', 'xmind', '.editorconfig', '.gitignore', 'CMDB.xmind', 'Makefile', 'README.md', and 'pipeline.xmind'.
- Right Panel:** Displays the file list for the repository. It includes a search bar, a 'History' button, and a table of files with their commit hashes and last update times.
 

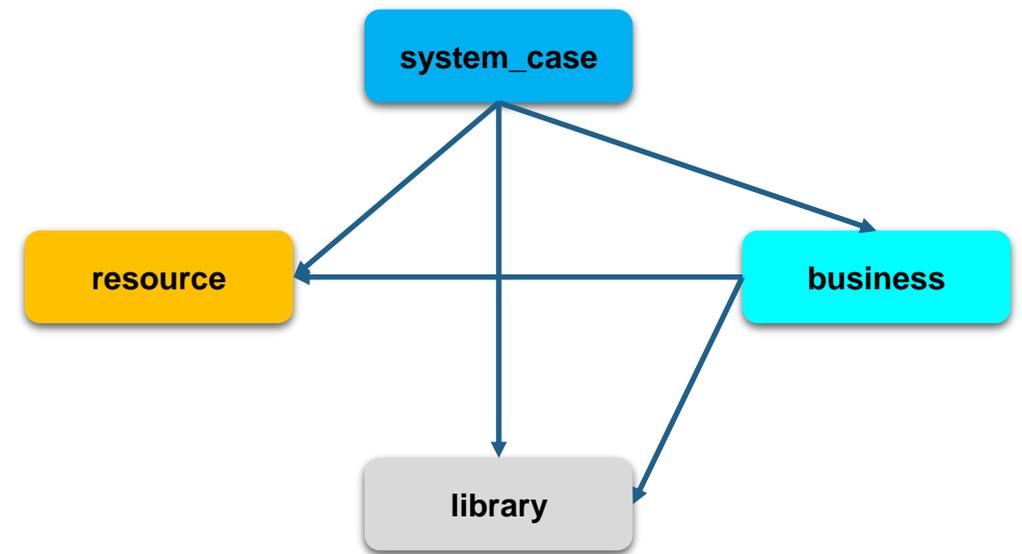
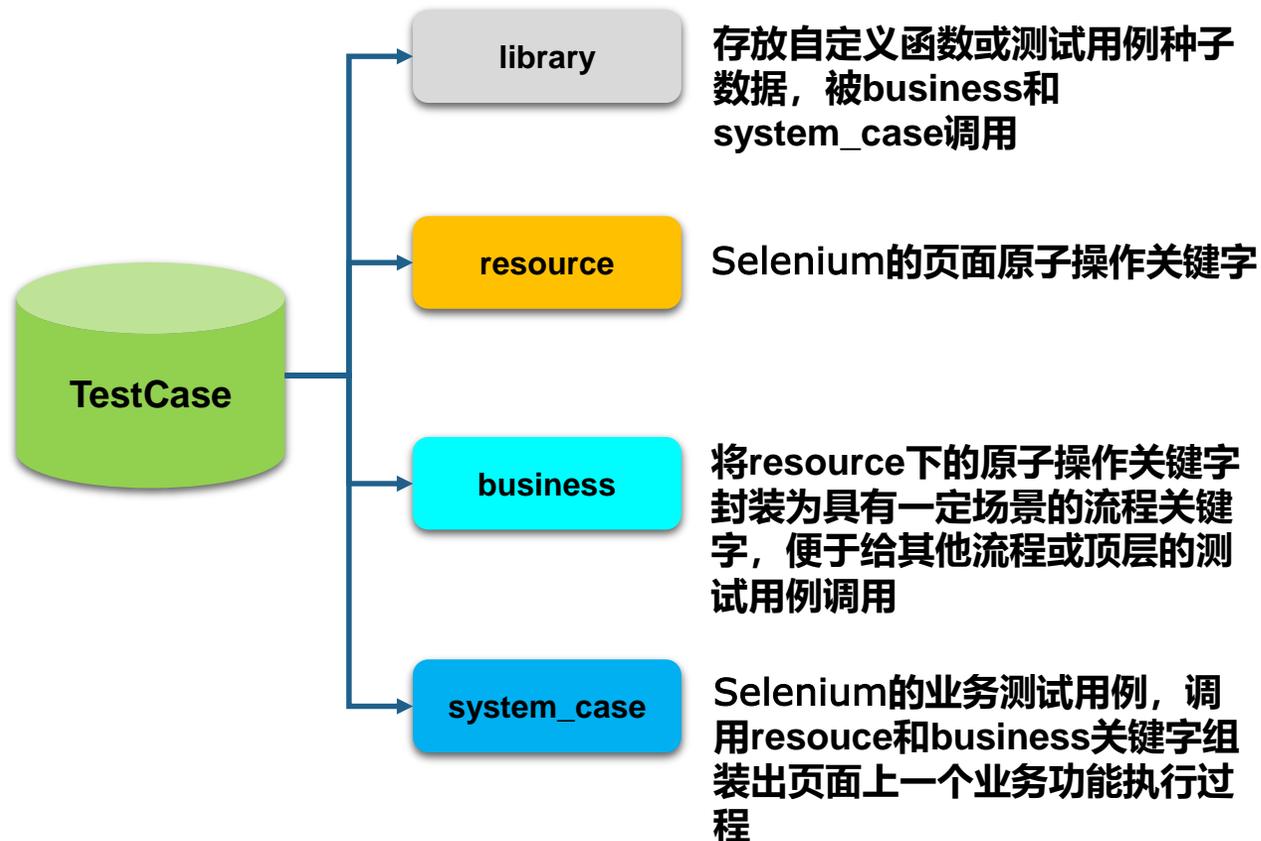
File Name	Commit Hash	Last Update
CMDB.xmind	58fb8325	a year ago
Makefile		10 months ago
README.md		10 months ago
pipeline.xmind		a year ago

A 'Switch branch/tag' dialog box is open in the center, showing a search bar and a list of tags: 2.26.1, 2.26.2 (highlighted), 2.26.3, 2.27, 2.27.0, and 2.27.1.

# 解决方案：把控质量，测试分层体系和自动化

## UI 自动化测试的一些实践经验(2)

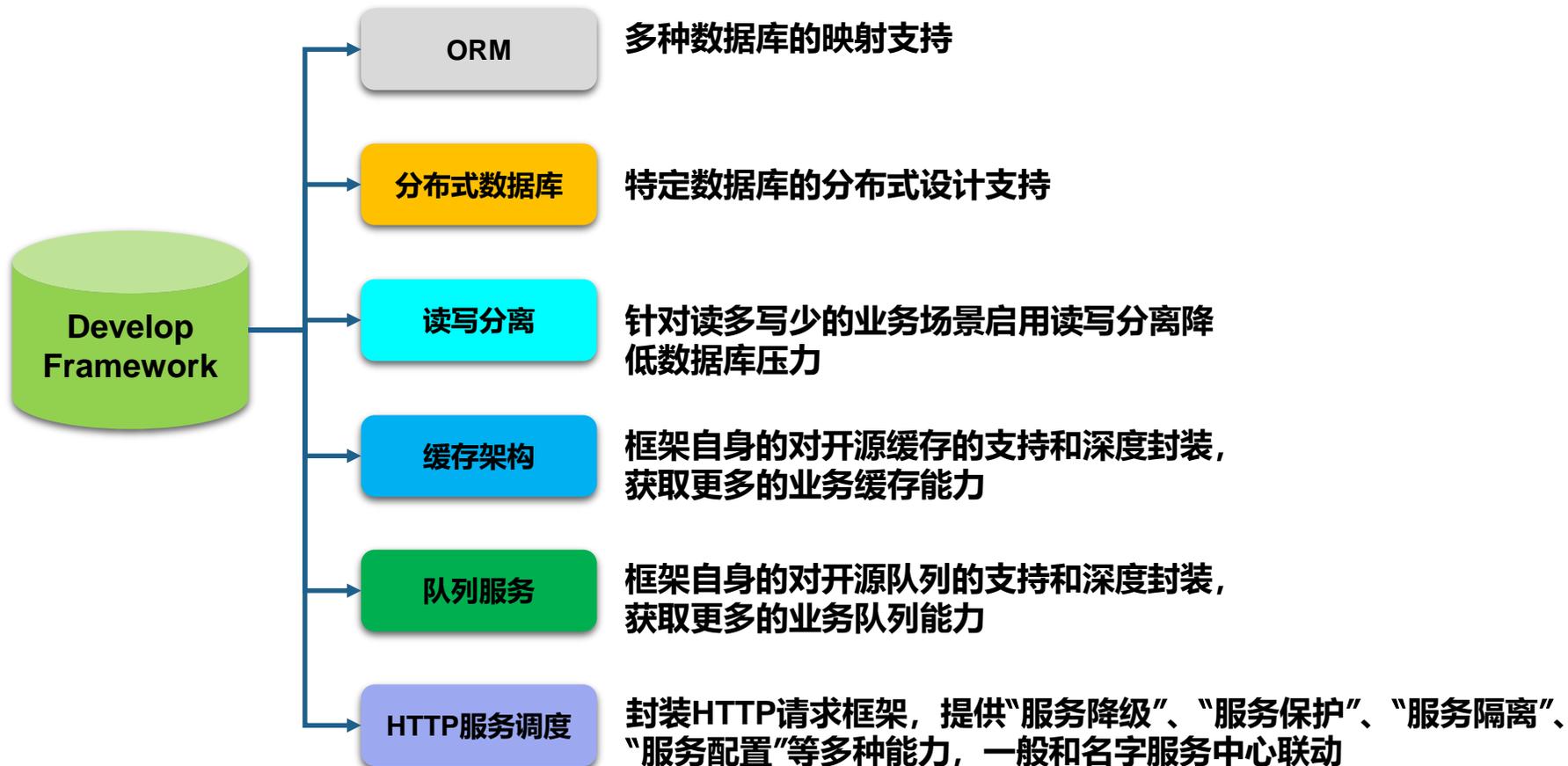
- 原子关键字、流程关键字要独立出来，给具体的业务场景所引用
- 根据业务系统功能模块划分父子目录，建议一级菜单和二级菜单建立文件夹，二级菜单下以功能名称命名建立套件文件



模块调用示意图

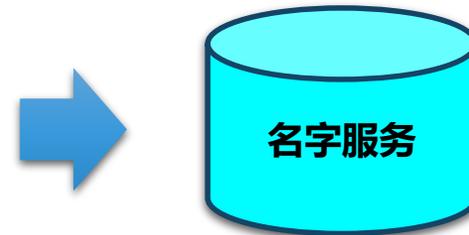
# 解决方案：内部开发框架定制化，提供多种研发自服务

内部定制的开发框架，由大牛级别的框架定制组来支撑，支持众多的开发服务



# 解决方案：内部开发框架定制化，提供多种研发自服务

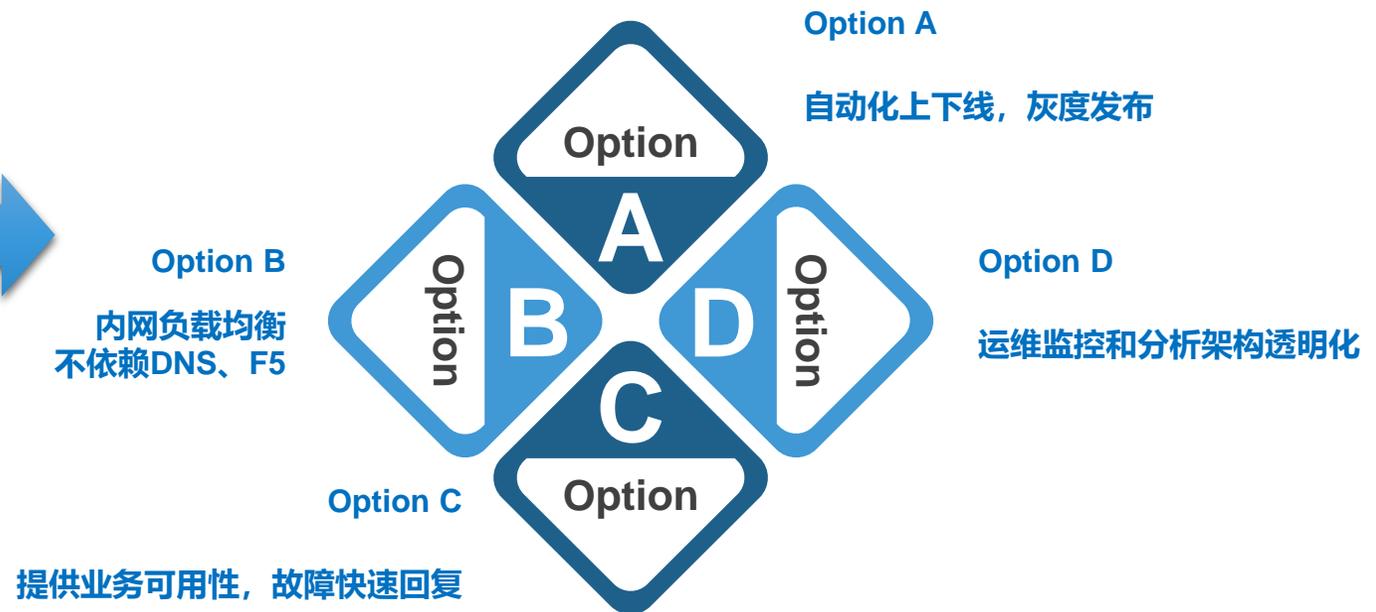
定制的开发框架通过支持名字服务，提供多种多样的服务能力



你告诉我“名字”，我给你提供“服务”。

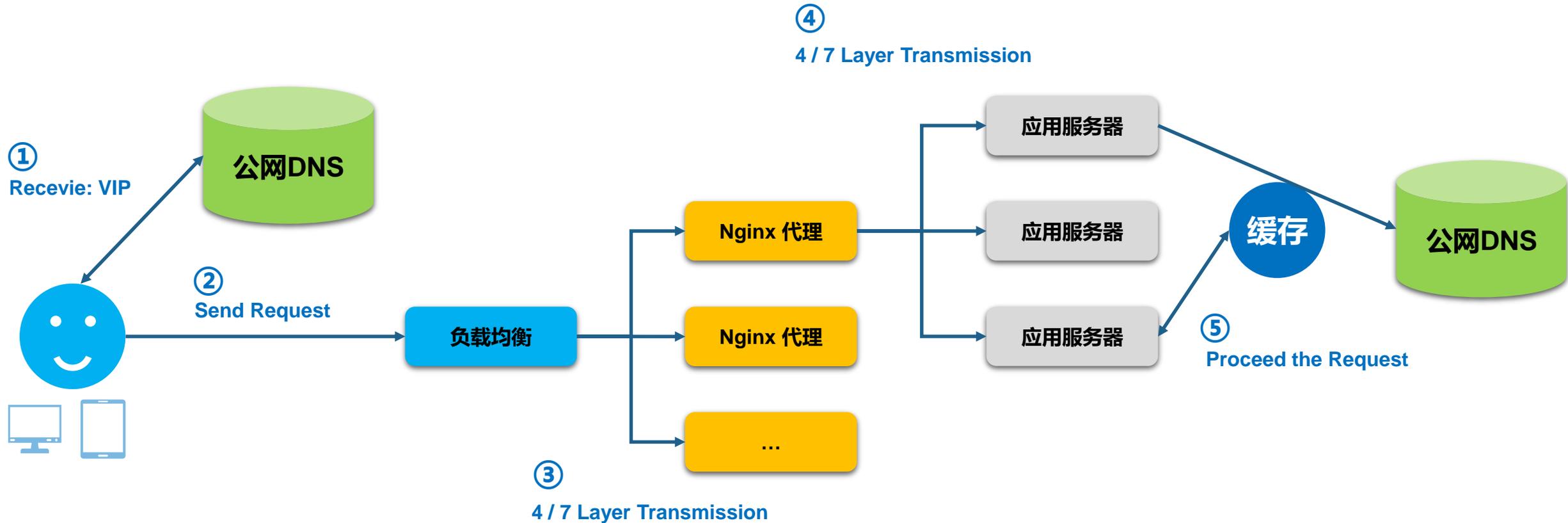
# 解决方案：内部开发框架定制化，提供多种研发自服务

定制的开发框架通过支持名字服务，提供多种多样的服务能力



# 解决方案：请求流优化用户体验

用户请求流是优化和排障的关键思路



# 解决方案：请求流优化用户体验

## 优化和监控 DNS（请求流图注①）

### 基于的 Bind DNS View 的智能 DNS

采用 BIND 时，通过设定不同的 ACL 来提供不同的 View，返回适合的 IP



### 适当设定你的 DNS TTL

DNS TTL 可以设定 DNS Server 对域名的缓存时间，如果经常更换服务 IP，适当降低 TTL 的时间，具体设定按照实际情况进行



### 使用 HTTP(S) DNS 来防止 DNS 劫持

采集 C/S 架构时，可以内置 HTTP(S) DNS 客户端，直接通过 HTTP(S) 协议获取 IP，防止 DNS 劫持



### 全国多地多机房拨测反馈 DNS 解析质量

可以利用一些开源或者商业产品，测试 DNS 在全国各地的解析质量，防止局部地区异常



### 使用 CDN 缓存静态服务

使用 CDN 可以承接原有的静态文件的解析服务，把 DNS 请求转接到 CDN 服务上，加快用户的就近访问速度



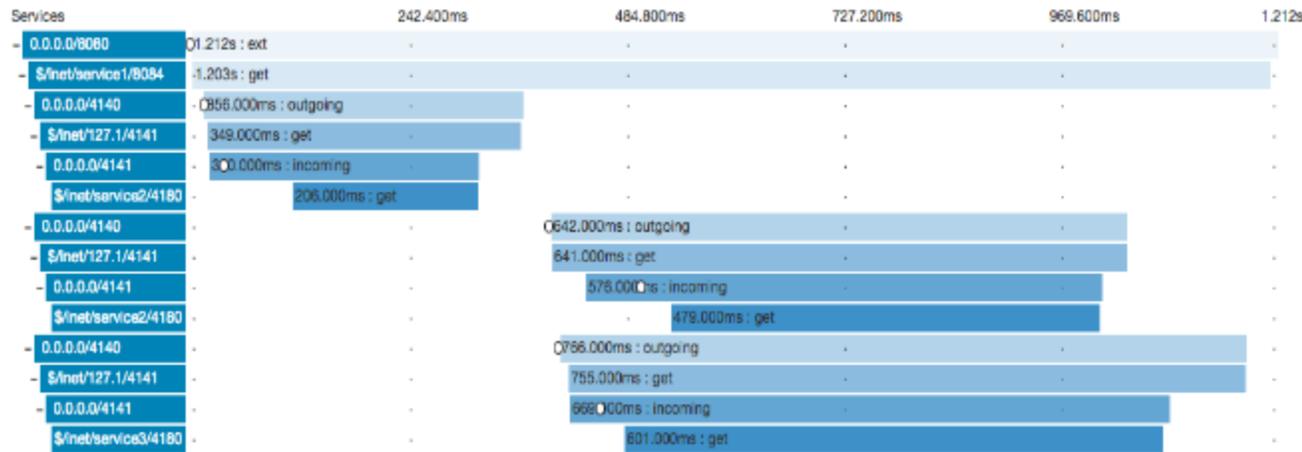
### 前端使用 css sprite 合并零碎的小图片

前端可以使用 css sprite 来合并零碎的小图片到一张大图上，从而减少用户的 DNS 请求时间



# 解决方案：请求流优化用户体验

## 优化和监控 DNS（请求流图注③④⑤）：分布式跟踪



### Trace

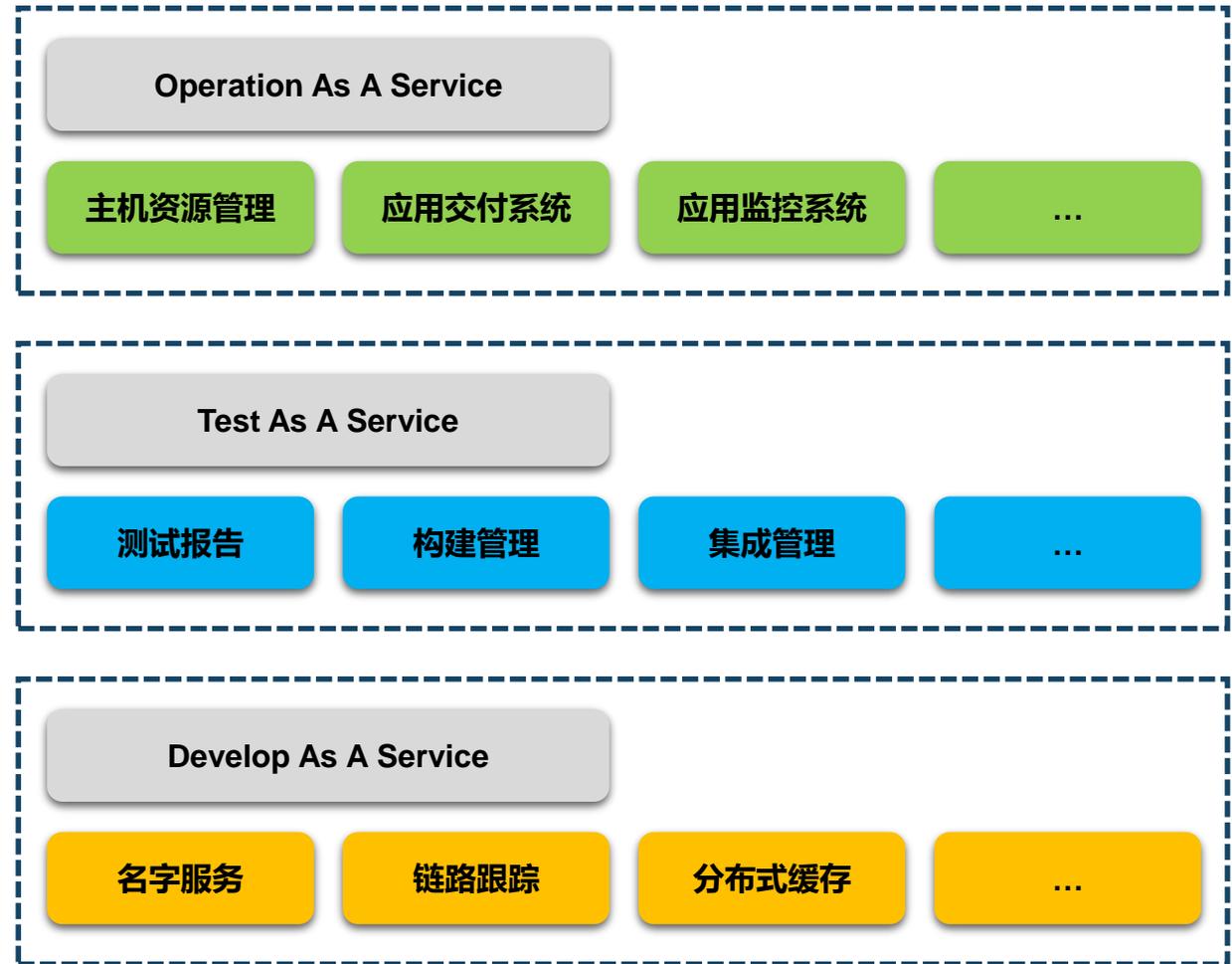
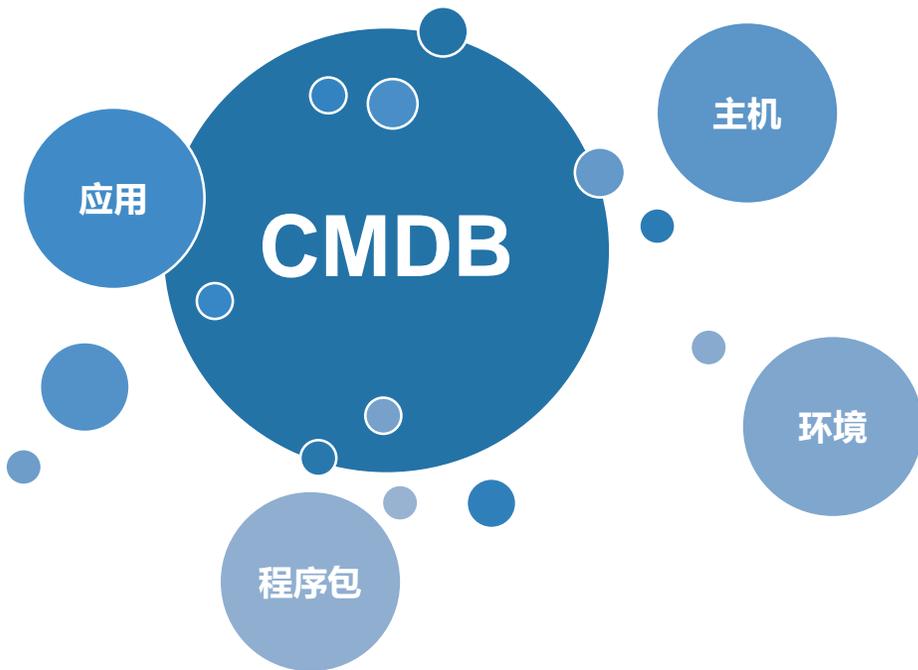
- 代表一个事务或者流程在(分布式)系统中的执行过程。
- 在 OpenTracing 的理论中，Trace由多一个Span组成的一个有向无环图(DAG)



### Span

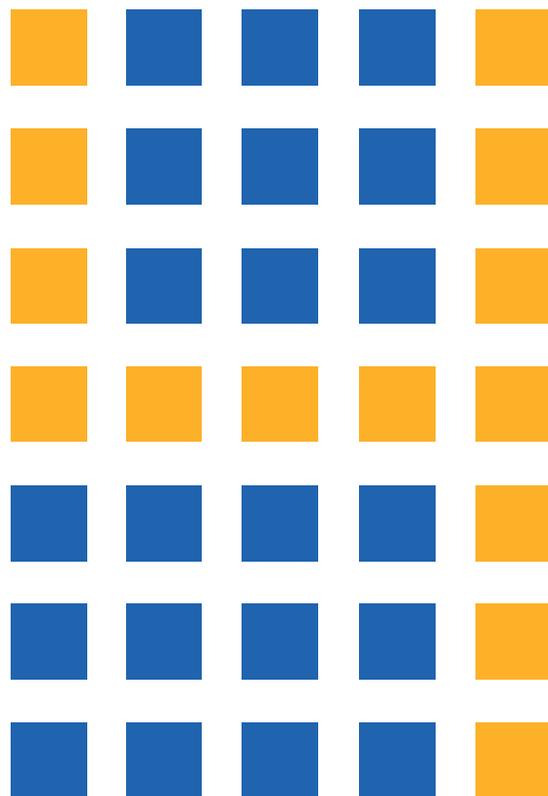
- 代表一个事务或者流程在(分布式)系统中的执行过程中的连续性执行片段

# 解决方案：平台服务化，提高自助水平



□ 以IT资源管理作为平台数据支撑来源

□ IAAS/PAAS, 把人力转变为服务, 释放生产力, 提供自助式环境



## 案例：证券，跨越传统职能组的交付模式

---

# 企业画像：业务需求



## 主营业务

以互联网业务为主，传统的核心交易系统无法构建完整的交付工具链



## 需求特征

APP、门户等系统形态，需求更新频繁



## 需求供应

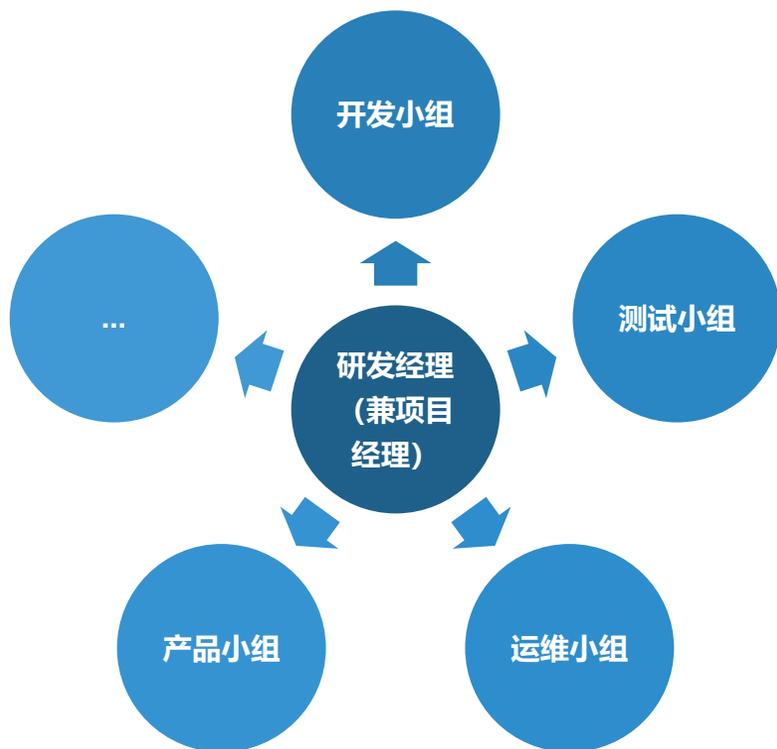
以需求部门提供需求为主



## 用户群体

全国各地的炒股、行情、资讯等用户

# 企业画像：IT服务交付团队



## □ 功能小组

跨职能小组，里面包含了开发、测试、运维等人员，但是这些职能部门是彼此独立的，研发经理是核心人员，驱动整个交付。

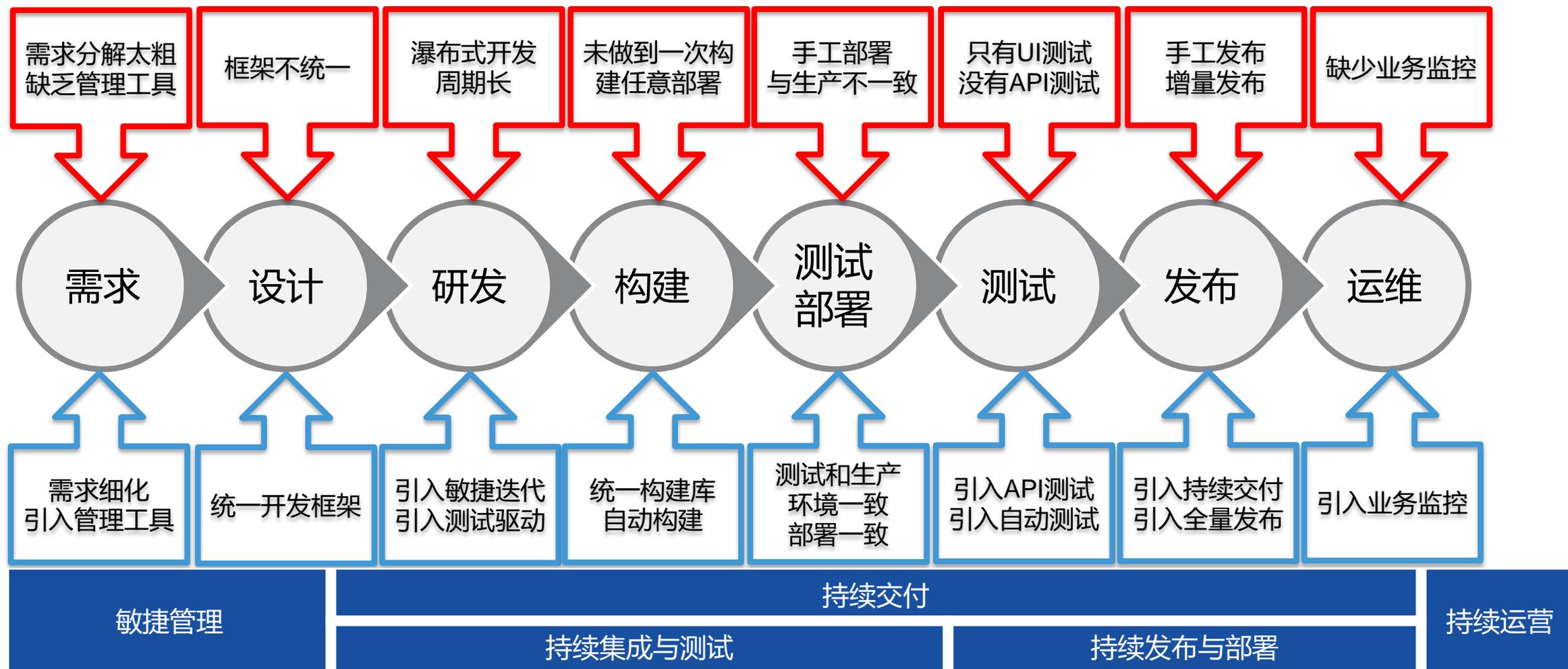
## □ 研发经理

研发经理就是研发的Leader兼任，关注整个交付周期，产品人员提炼需求，交付到研发手里，研发实现.....

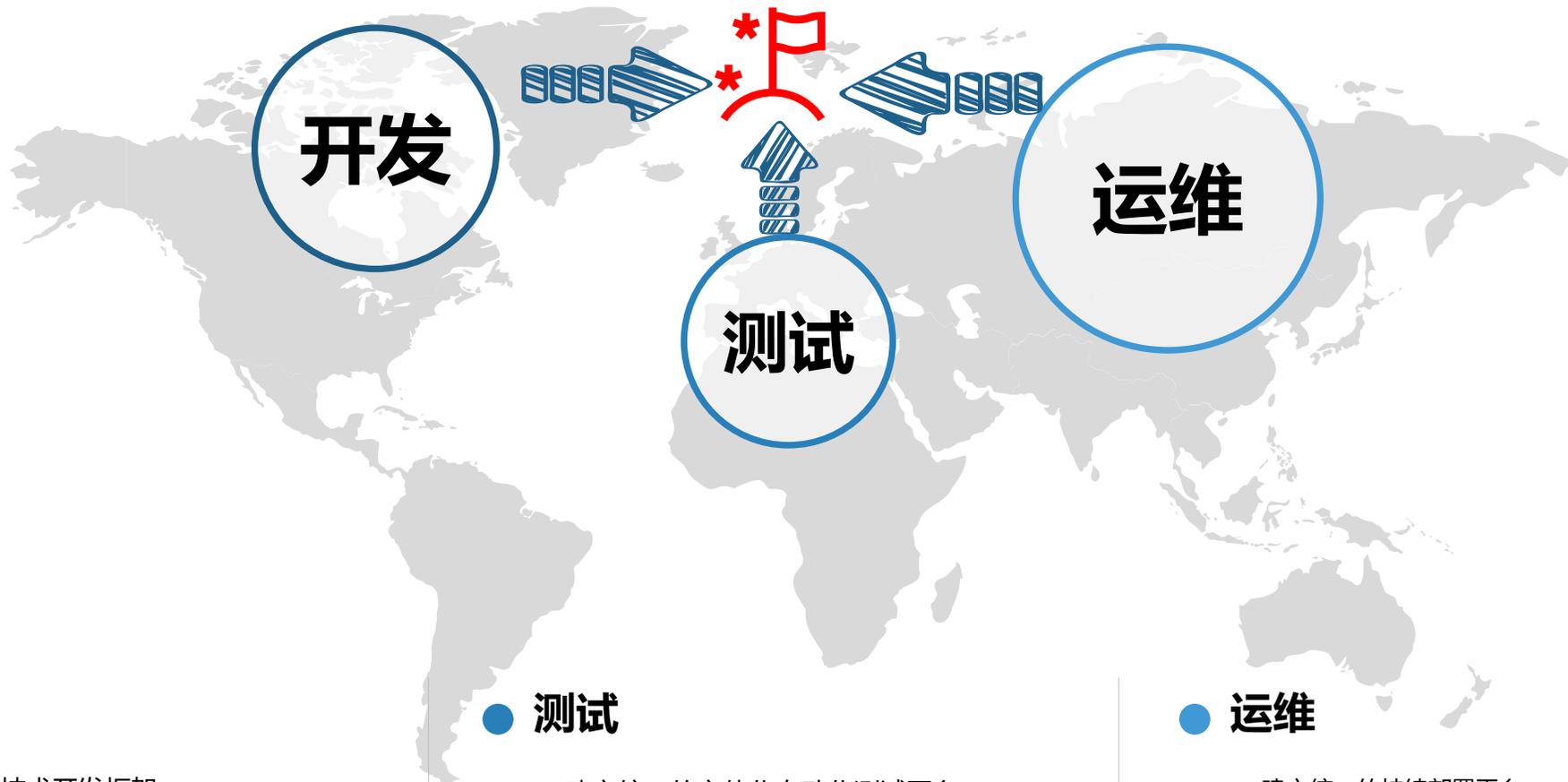
# 企业画像：资源和环境需求



# 企业调研：面临的核⼼问题



# 解决方案：基于局部优化的井冈山会师模



## ● 开发

- 建立统一的技术开发框架
- 功能性需求和非功能性需求统一考虑
- 代码开发规范
- 建立统一的持续集成规范

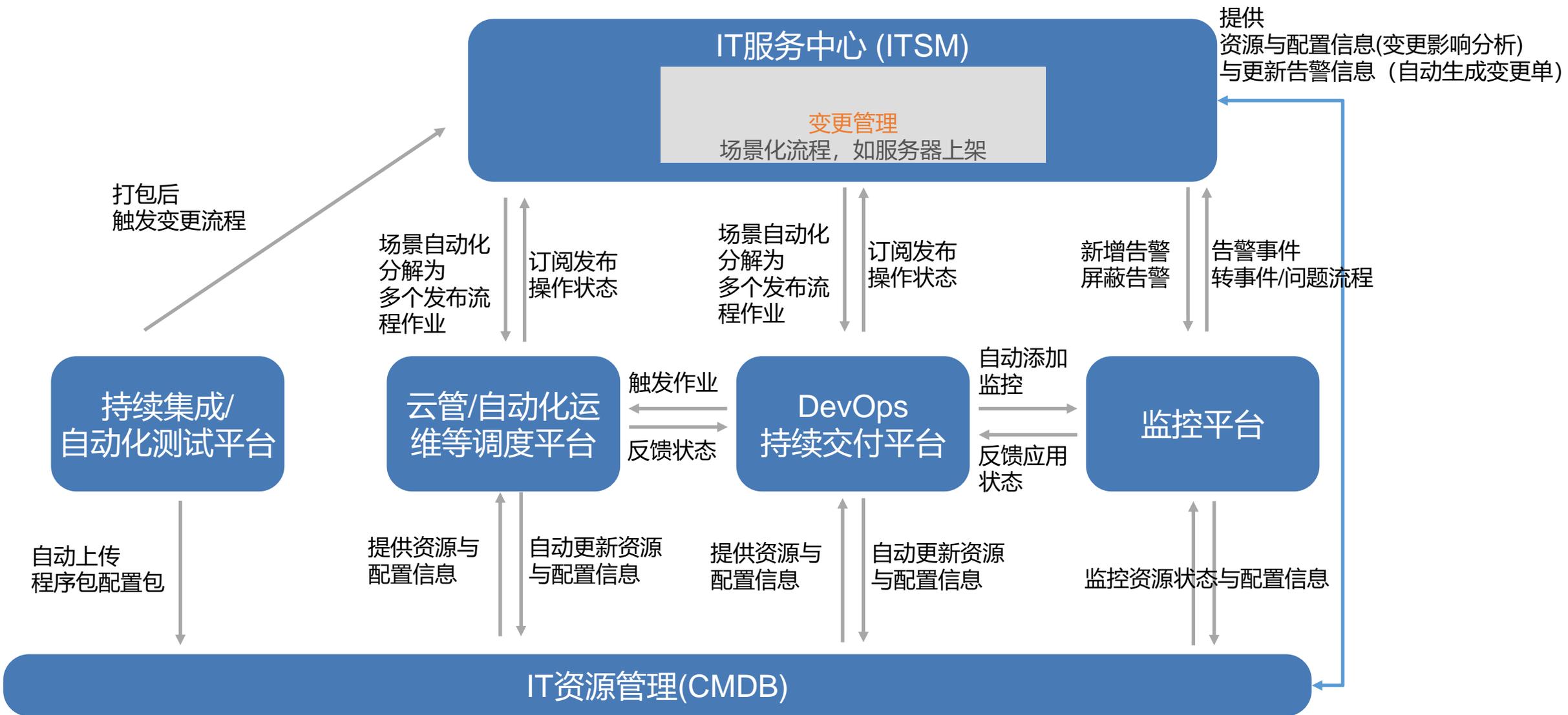
## ● 测试

- 建立统一的立体化自动化测试平台
- 接管统一的持续集成管理规范
- 测试驱动研发

## ● 运维

- 建立统一的持续部署平台
- 建立统一的应用监控平台
- 建立统一的应用资源管理平台
- 建议统一的标准化交付规范，利用生产倒推开发、测试

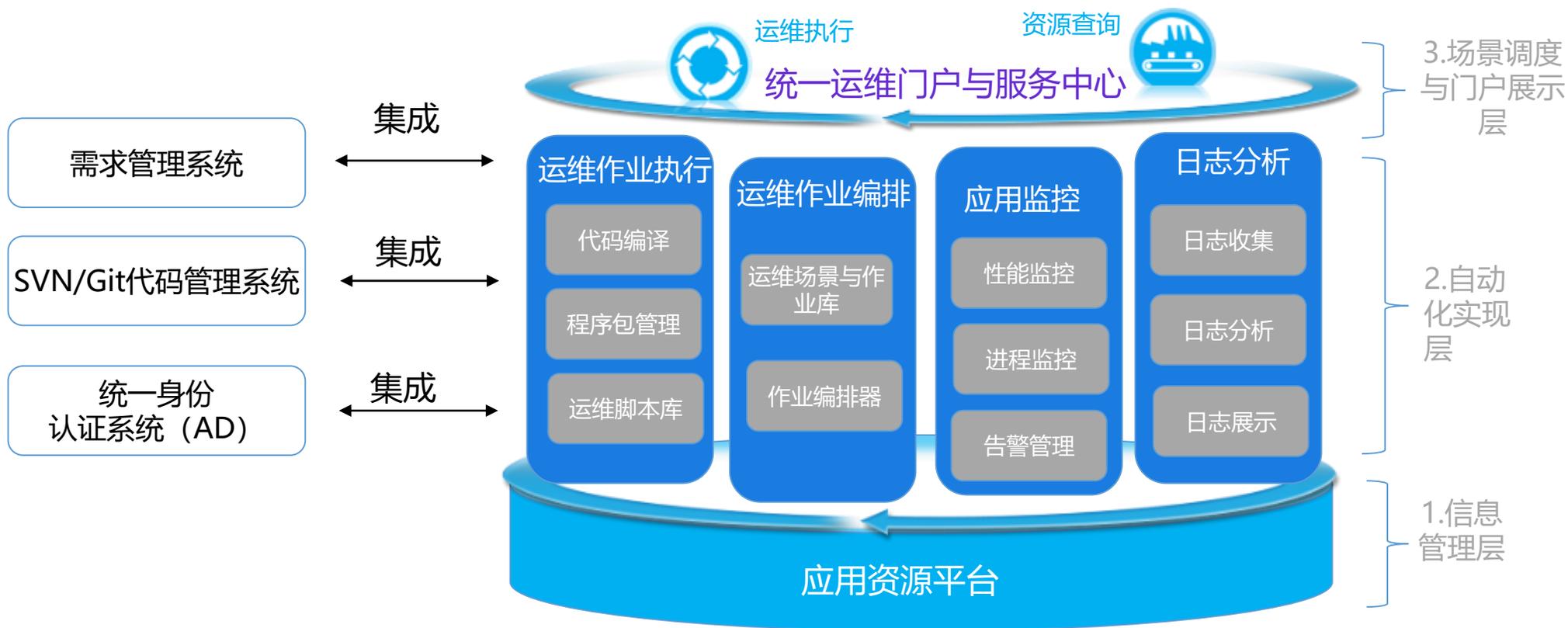
# 解决方案：ITIL与DevOps的融合，兼顾合规而变

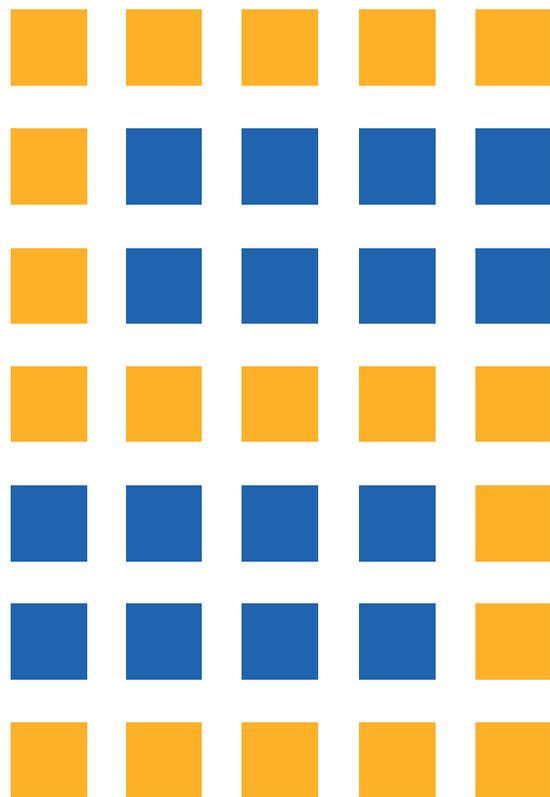


# 解决方案：以生产环境为核心目标的运维自动化

上述场景可抽象出一体化运维服务体系的三层功能需求：

- 1.运维信息管理层——建立以**应用为中心的CMDB**，实现基于应用的信息资源记录/组织和调度管理，是自动化平台的信息核心；
- 2.运维自动化实现层——实现运维作业的自动化执行和监控的技术实现
- 3.运维门户调度展示层——基于不同运维角色的操作和展示与场景流程调度

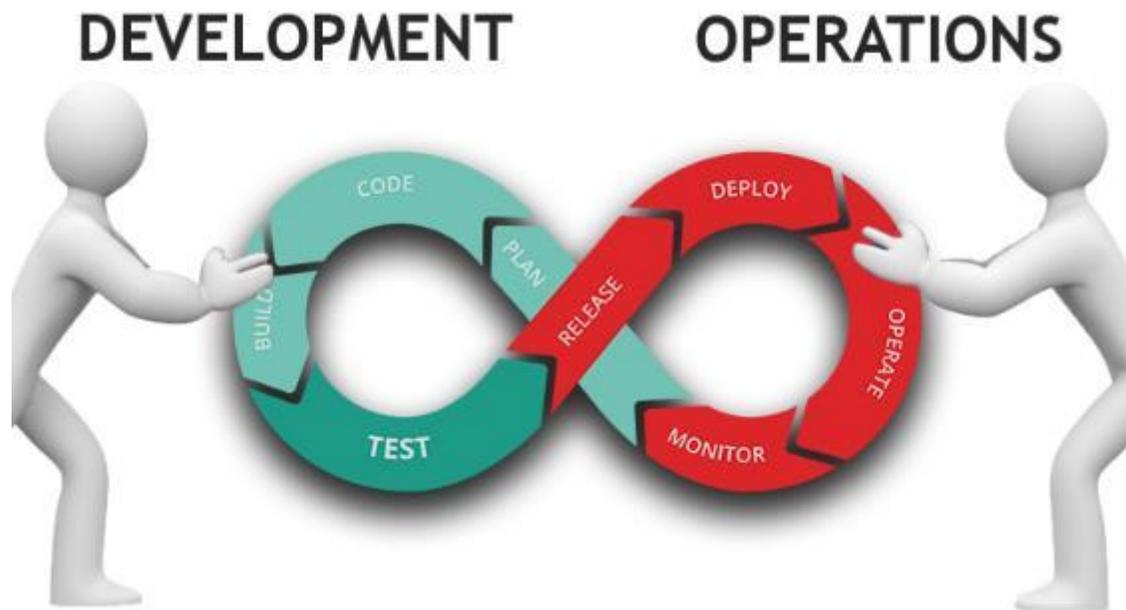




## 回顾 DevOps 的本质

---

# DevOps最核心的矛盾：交付链的缺失



## □ 组织因素

组织之间存在职能竖井是最大的障碍、组织的部门墙、组织没有打破常规的文化

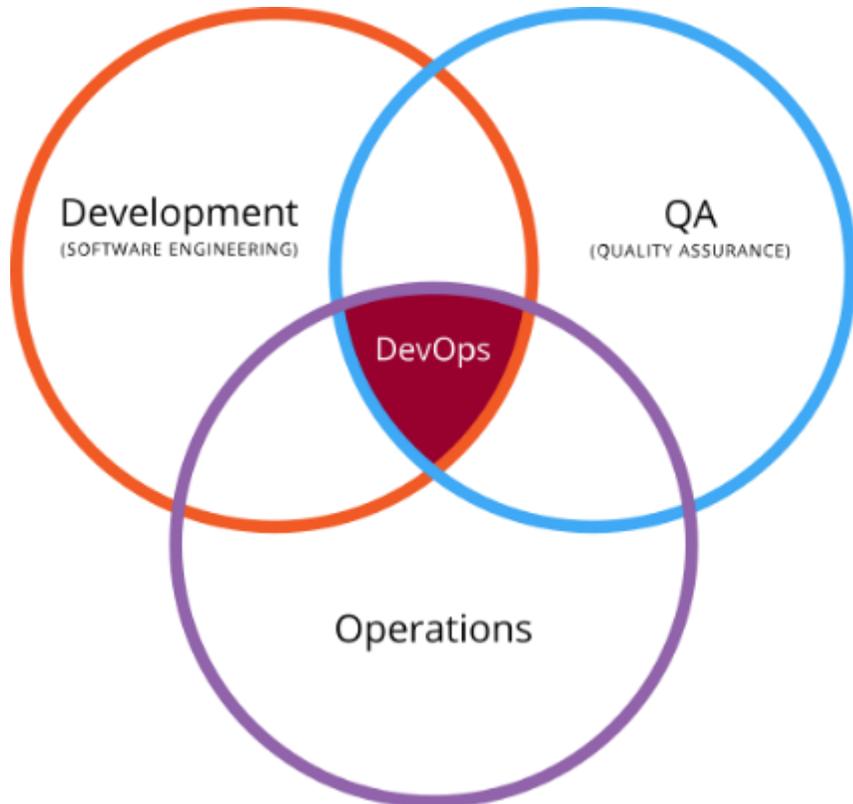
## □ 流程因素

过去重流程，而非过程的思维建设，导致工具链能力缺失.....

## □ 人员因素

人员的能力、意识是DevOps顺利实施的障碍

# DevOps 的本质



**DevOps is the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.**

DevOps 是一种由开发、测试和运维团队协同工作，为企业的IT服务，提供从开发设计到生产运营的完整交付链的实践。

**DevOps 道阻且长，但，是必然趋势**



# THANKS