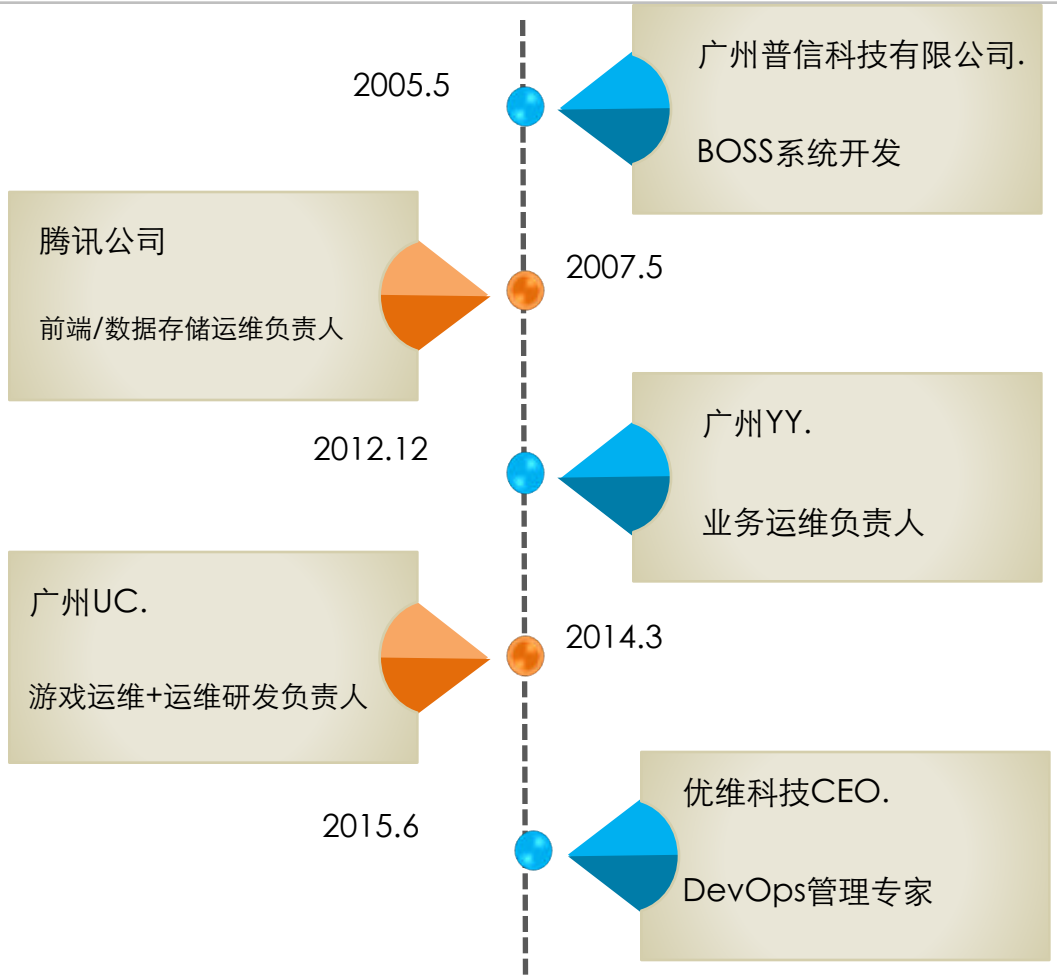
# DevOps运维体系框架与其精益实践

## —以运维为始，以运营为终，以交付为桥


优维科技（深圳）有限公司
DevOps管理专家

# About Me


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

# About Me

1、05到07年参与电信BOSS系统研发，承担了其中资源管理模块（模块化架构）。

2、07年进入腾讯，接手ISD所有应用发布，并构建了ARS系统。

3、08年，接手自动化构建，其中主导从CC到SVN环境的迁移，重构了其中部分脚本。

4、12年之后，主导YY和UC的运维平台体系构建，在UC顺便把游戏的微服务化改造做了。

5、15年创业，全栈运维平台EasyOps，DevOps管理专家。

# 写在前面的话

1、基于交付链(Dev/Test/Ops)的全局优化，而非局部(Ops)优化

2、运维的问题不是仅仅运维侧的问题，是一个IT问题

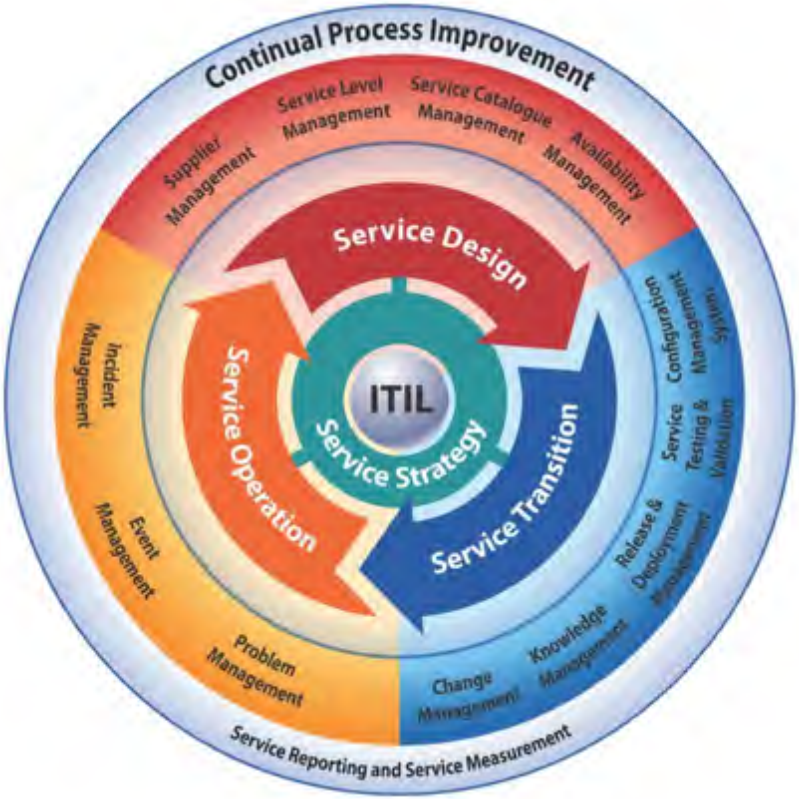
3、运维问题表面是复杂的(组织/人)，本质上是简单的(技术)

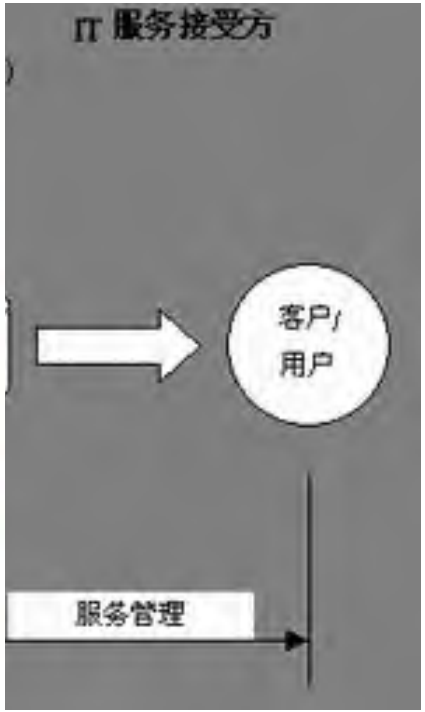
4、坚持技术是内部第一驱动力，业务是外部第一驱动力，很多问题便找到了答案

5、运维人要忘记什么是运维，玩玩跨界

# 运维，从ITSM到ITOM

# 基于ITIL的运维理解




ITSM是ITIL的服务化描述

# 基于DevOps的运维理解



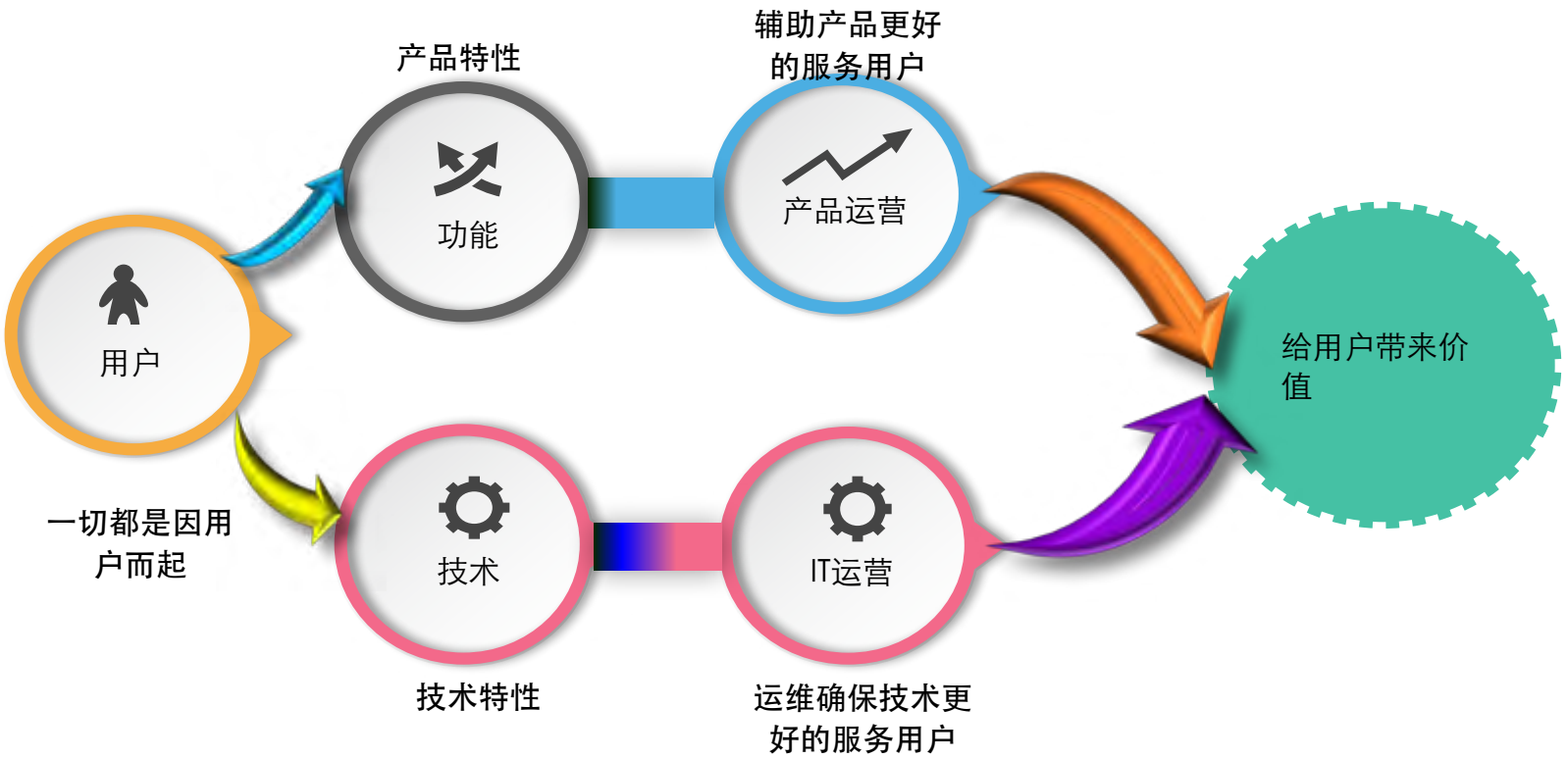
以自动化过程为核心的理念


# 基于DevOps运维就是ITOM


ITIL的运维化是ITSM

DevOps的运维化是ITOM

# 运维2.0=IT运营

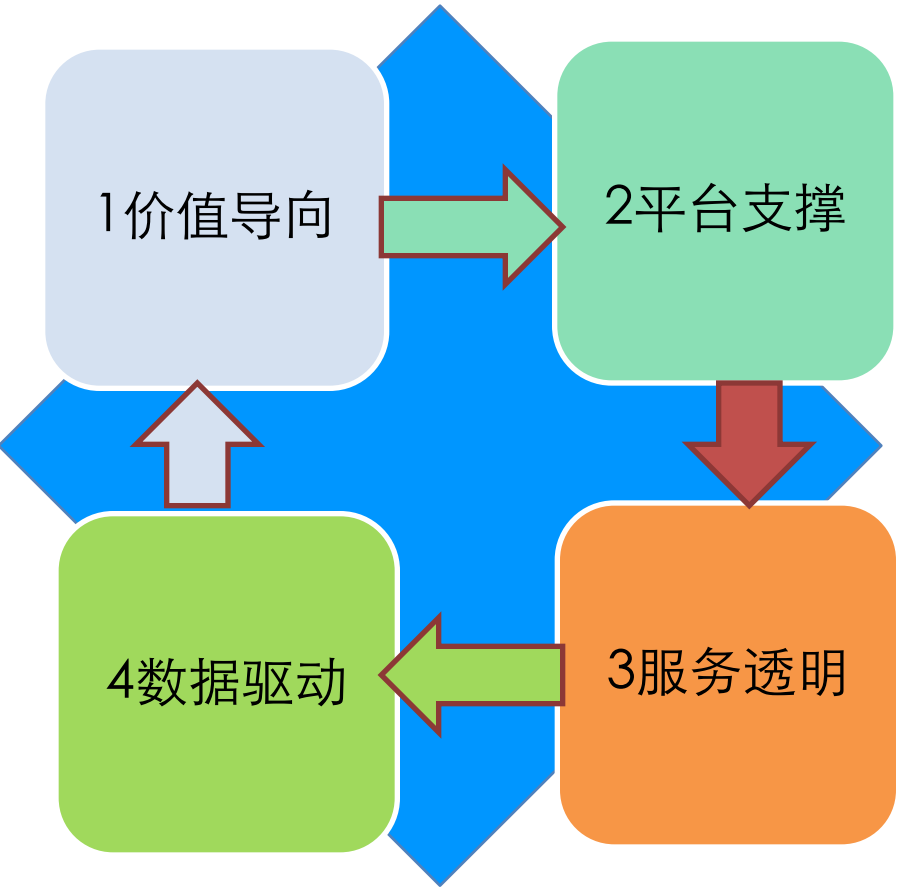

产品特性
辅助产品更好的服务用户
功能
产品运营
用户
给用户带来价值
一切都是因用户而起
技术
IT运营
技术特性
运维确保技术更好的服务用户

# 何为价值?


面向用户思维和面向内部服务思维

面向业务价值思维和面向KPI思维

# IT运营管理的整体原则（16字）




价值：关注的是用户价值

平台支撑：运维平台和研发者平台

服务透明：运维者服务和研发者服务

数据驱动：没有数据就无法衡量，也就无法驱动

# DevOps运维体系的整体架构（过程）

数据化运维

1、标准化应用服务组件
2、标准化应用部署及管理
3、标准化配置管理
4、标准化协议

标准化

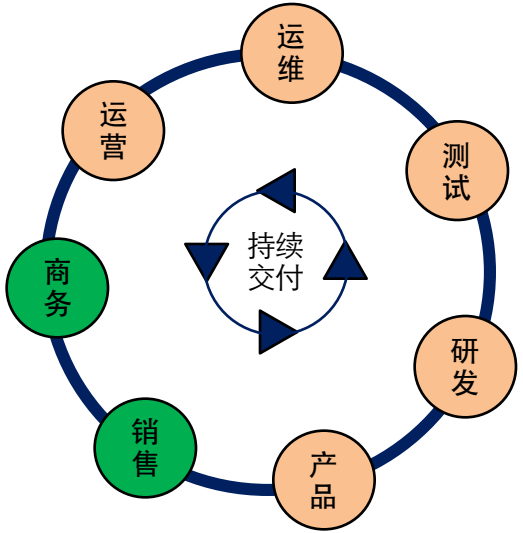
服务化（架构）

1、分布式关系存储
2、分布式文件存储
3、分布式cache服务
4、统一调度中心

无状态化（架构）

1、进一步人工或自动梳理业务访问路径，彻底的去状态。
2、从服务降级、柔性可用、过载保护、双中心调度、立体监控等多个层面给业务提供优化

自动化运维




1、DevOps以技术为核心，促进组织变革，达成组织之间的协同。

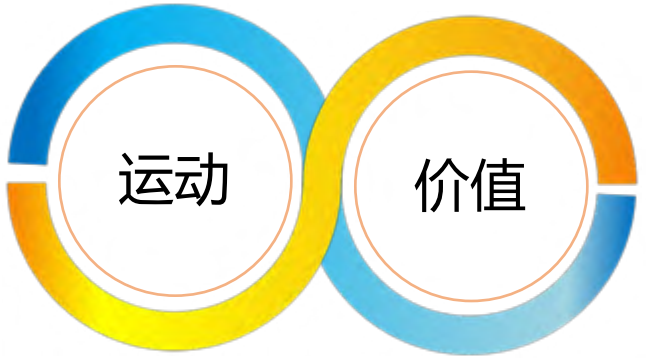
2、持续交付平台整合多方角色实现面向用户的产品和价值交付。

3、销售和商务这两个角色在部分企业不一定存在。

# DevOps的常识性理解

# DevOps 的运动起源

一天10次部署
基础设施即代码
敏捷基础设施运动
敏捷系统管理运动
平台即服务运动。

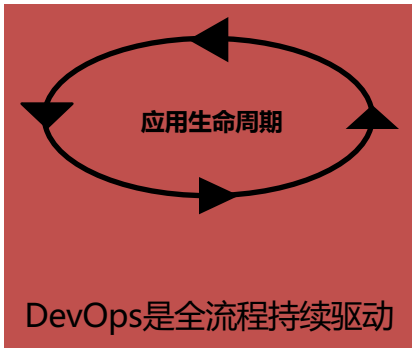



面向用户的价值

# DevOps 是一种思维

# DevOps之精益五原则

# DevOps 是一种价值观




DevOps是全流程持续驱动

持续优化

DevOps提供一种清晰的模式，确保了团队之间的合作

共享责任

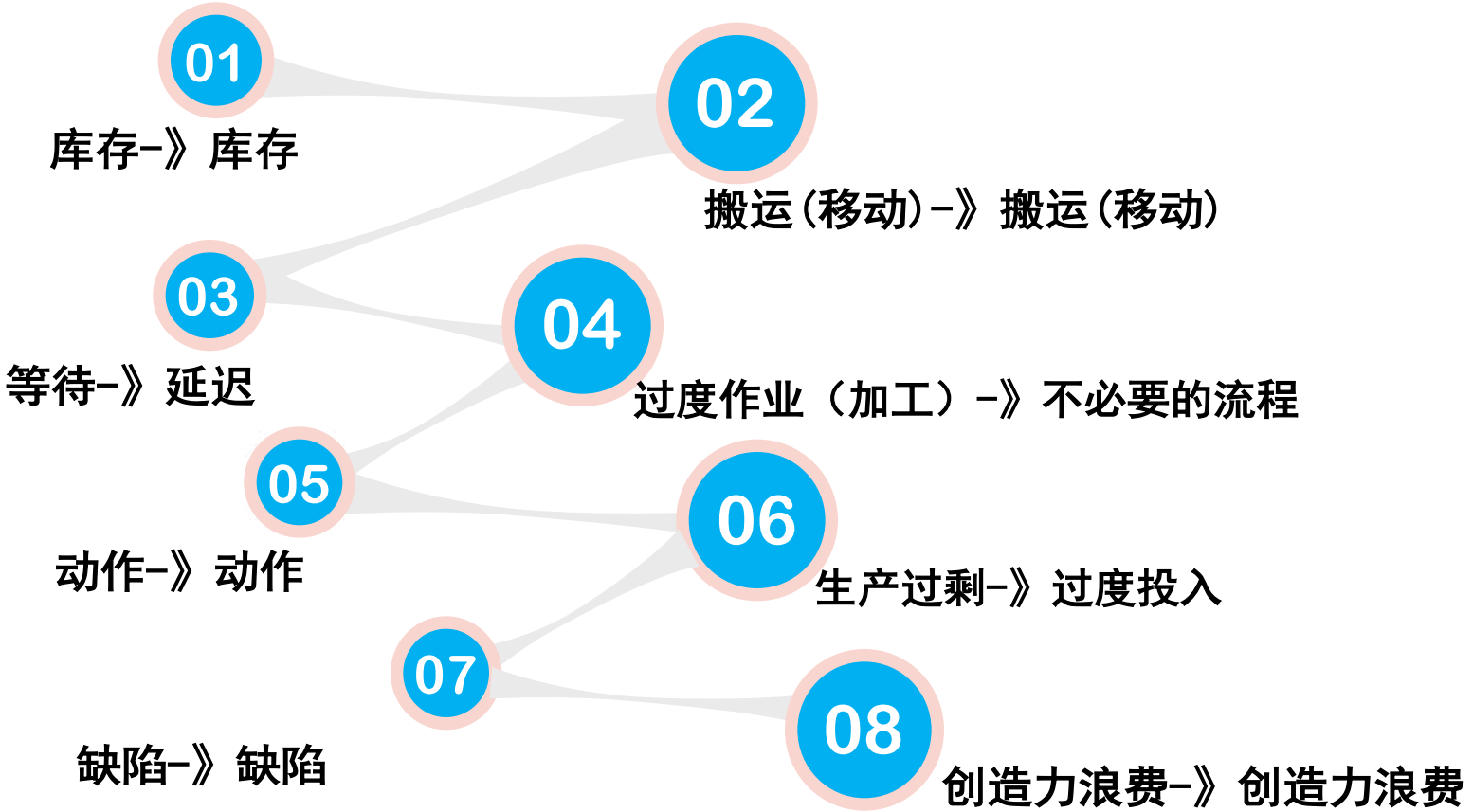
DevOps杜绝真正的浪费

杜绝浪费

DevOps 关注用户价值，快速交付

关注用户

# DevOps 之杜绝浪费



# IT中的浪费举例

**库存：半成品、部分完成的工作**

- 已完成但尚未签入的代码
- 没有相关说明文档的代码
- 未测试的代码
- 没人使用的代码
- 被注释掉的代码

**生产过剩：额外的功能**

- 客户/用户不需要或不使用的功能
- 当新增功能时，可能也同时新增了bug

**过度加工：不必要的流程**

- 额外的步骤或流程，从用户角度不产生价值
- 重新学习功能、类，或代码实现（无说明的代码）
- 对已经达到需求的代码进行重构

**搬运（传输）：交接**

- 开发人员之间的代码交接
- 开发人员和测试人员间软件的交接
- 软件从开发到部署的交接

**动作：任务切换**

- 多任务（同时分配多种工作）是一种浪费
- 同时参与多个团队工作，会造成更多停顿

**等待：延期**

- 团队协同中最大的浪费
- 等待项目审批、等待变更流程
- 等待资源、等待第三方响应

**缺陷：**

- 错误、返工、故障、功能缺失等

# 运维中的浪费举例

流程驱动运维的意识严重
而非技术，责任隔离

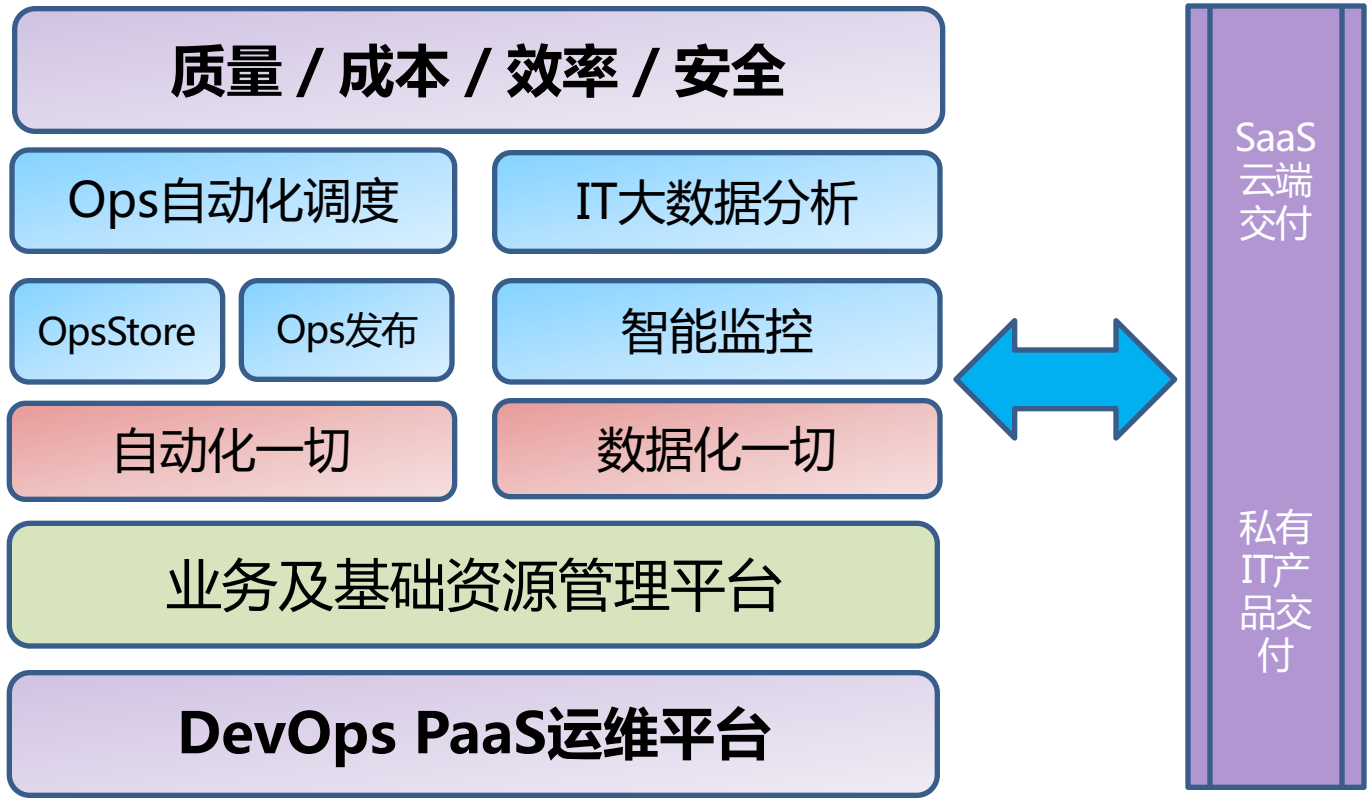
流程设置过重
加入很多不必要的节点，浪费严重

没有把流程和自动化结合
起来，流程效率低下

技术投入不够
设立很多非技术判断标准

**基于ITIL的流程化ITSM需要升级到DevOps运维**

# DevOps 是一种工具观



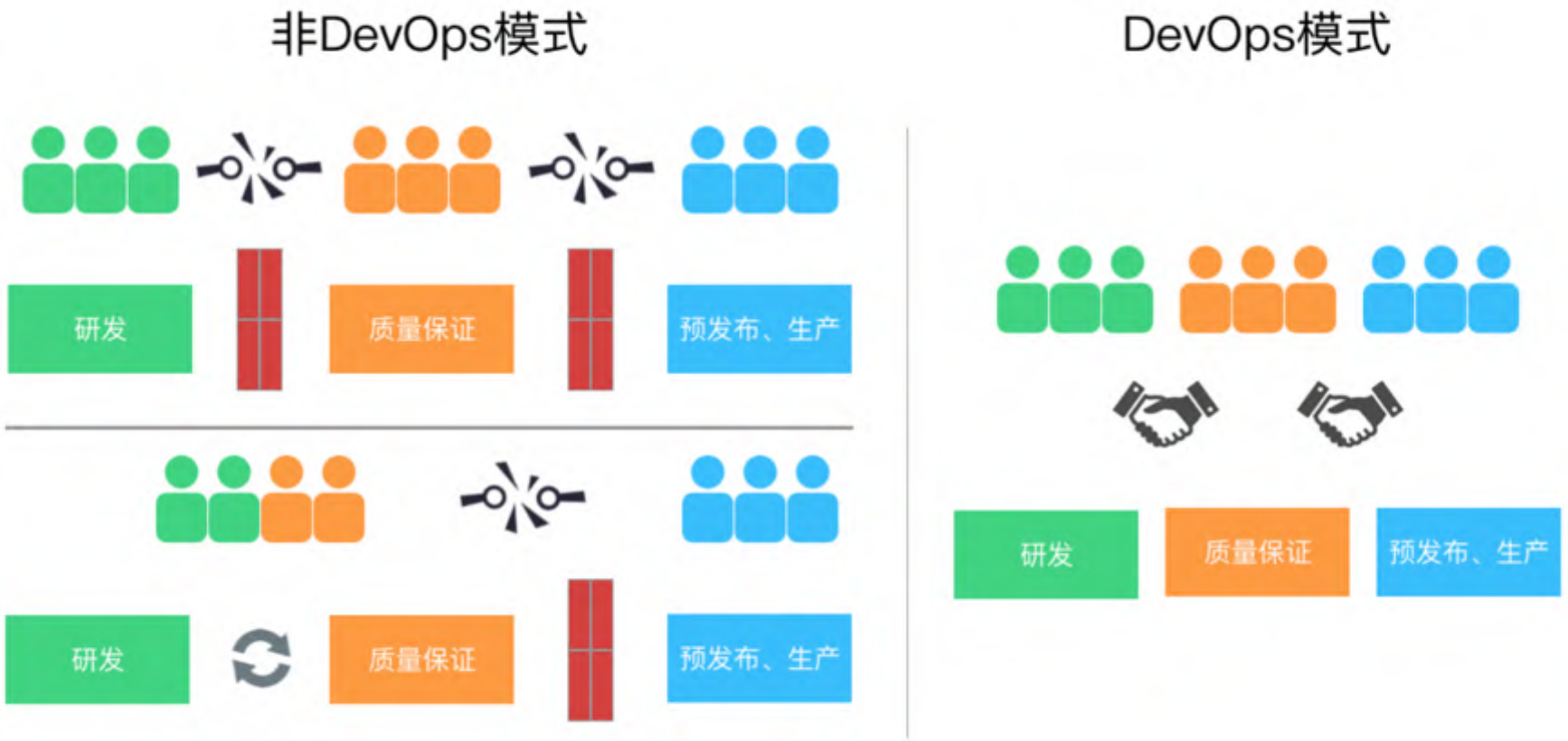

- 云 + PaaS + 一站式 运维 服务，是运维的核心形态

# DEVOPS是一种文化观

- 文化：
  - 合作（collaboration）
  - 自动化（automation）
  - 精益（Lean）
  - 度量（measuring）
  - 共享（sharing）

    - 首字母缩写"CALMS"（“平静”）最能代表这一点

# DevOps是软件交付模式



# DevOps之整体框架图

# DevOps运维体系组成部分

DevOps运维体系，分成横向和纵向两部分：

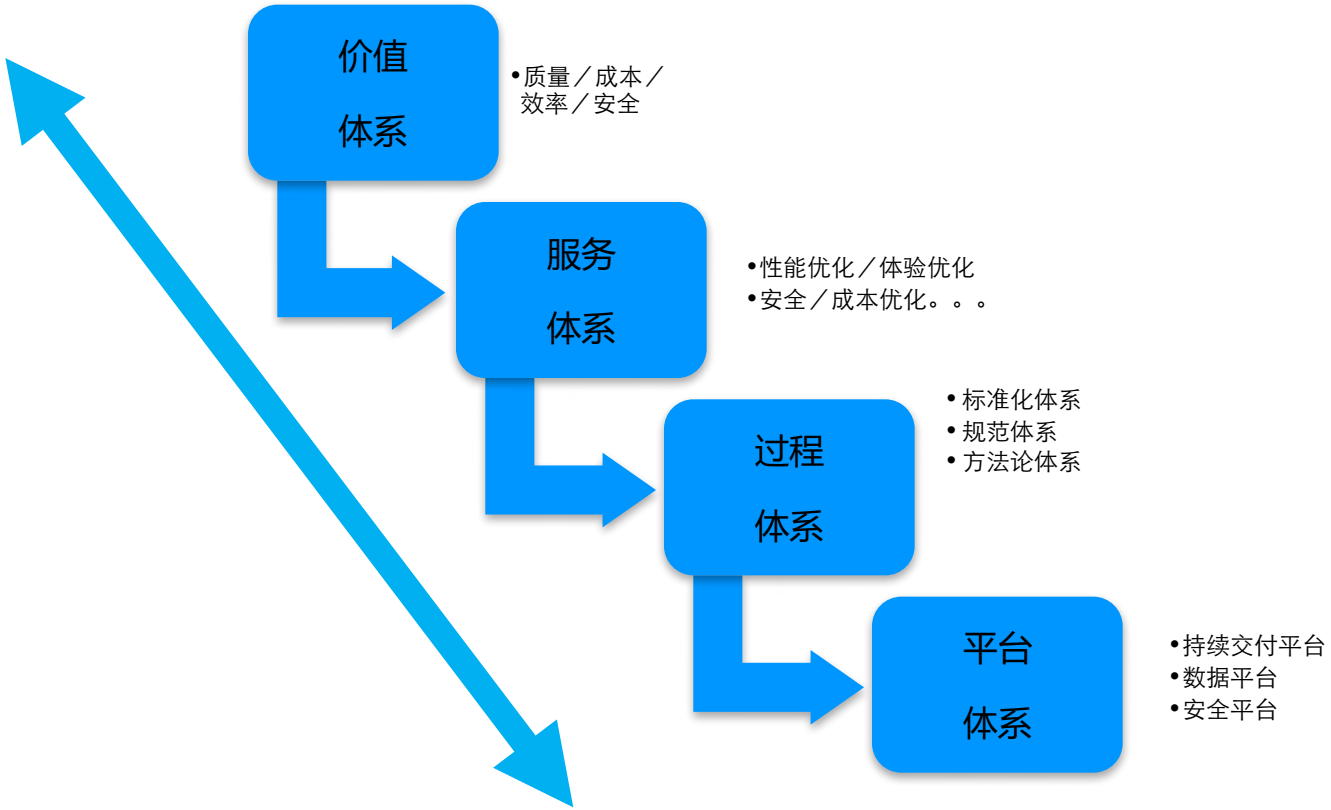
横向：DevOps运维6个维度（组织、过程、架构、工具、基础设施、度量）+1个文化

- 组织，DevOps首先必须打破组织之间的隔阂，其次DevOps团队建立面向产品而非项目的协作能力，是跨能力。
- 过程，轻量级流程和自动化工具的完美结合，确保企业的高度敏捷性；自动化为先，而后再流程。
- 架构，架构是DevOps成功的重要影响因素；巨石、单体架构是快速交付的最大障碍；架构与持续交付紧密联系。
- 工具，运维平台或者工具在提升运维效率的同时，也在影响着质量和成本；高效的应用化平台能力确保故障快速恢复
- 基础设施，从虚拟化到IaaS到容器化，敏捷的基础设施是高效精益运维的基础。
- 度量：建立全面的DevOps运维度量体系，正向驱动运维、研发、测试团队不断完成面向用户的交付。

纵向：DevOps能力评估模型（五个等级）

# DevOps运维体系之平台架构

# 运维平台在运维体系中的位置

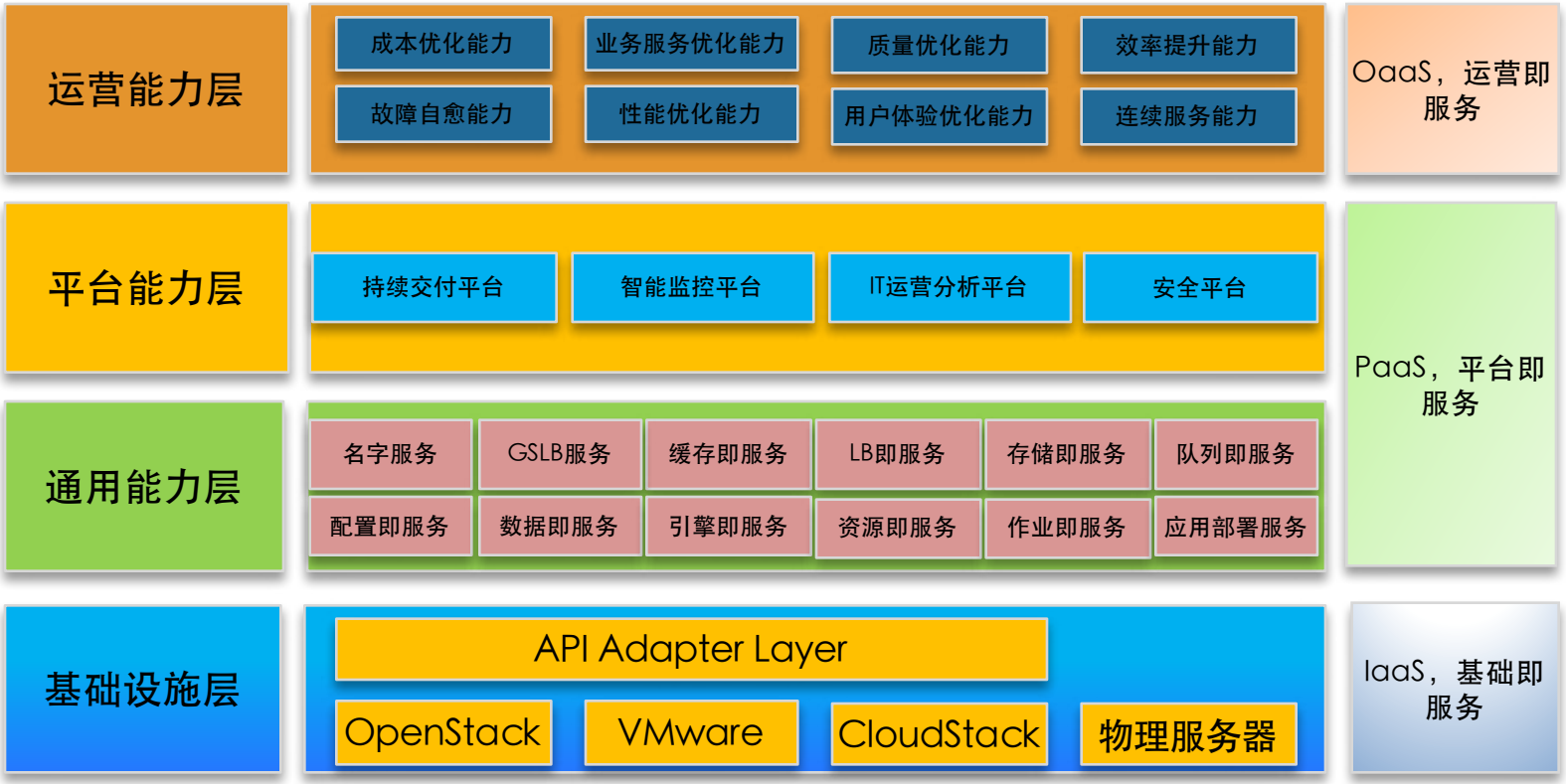


# DevOps运维平台的能力设计环


- 全栈的运维平台，是要考虑各种角色／场景／能力下的交付需求，面向能力域的交付能力是一种专业化的服务能力交付。


- 价值流分析法是基于场景的分析，端到端的交付实现。

# 运维平台概念模型

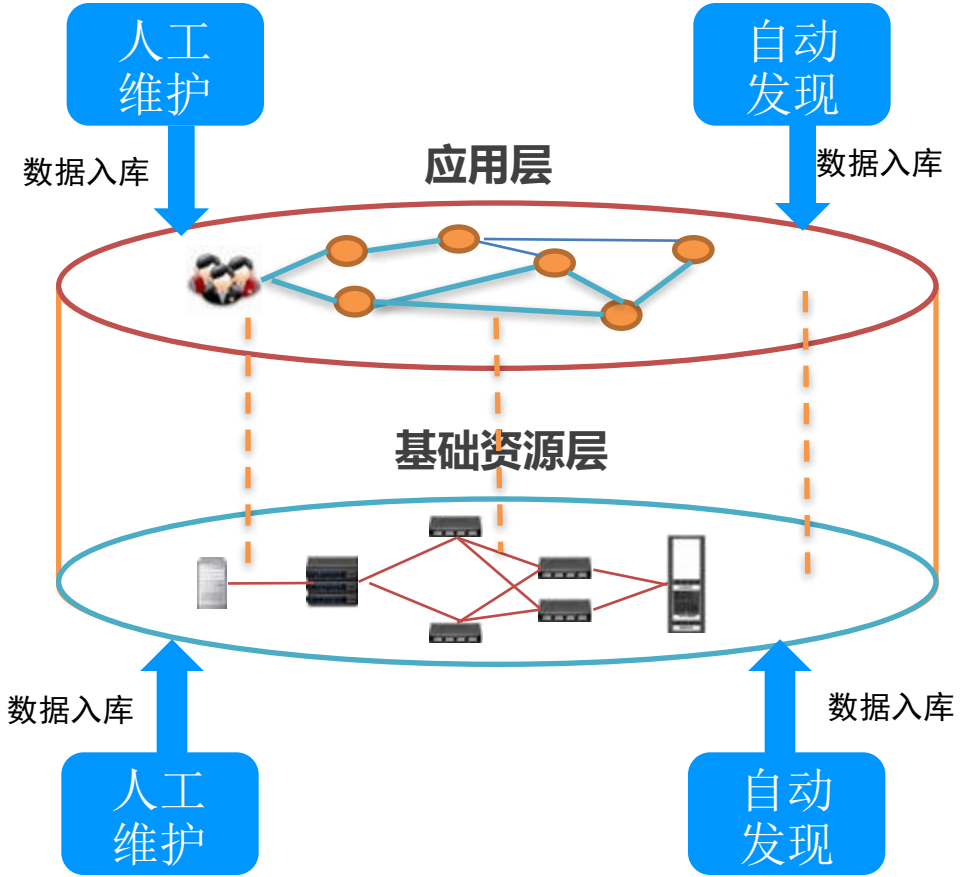



# 运维的平台能力分层建设

# 运维全平台系统概览(二级)

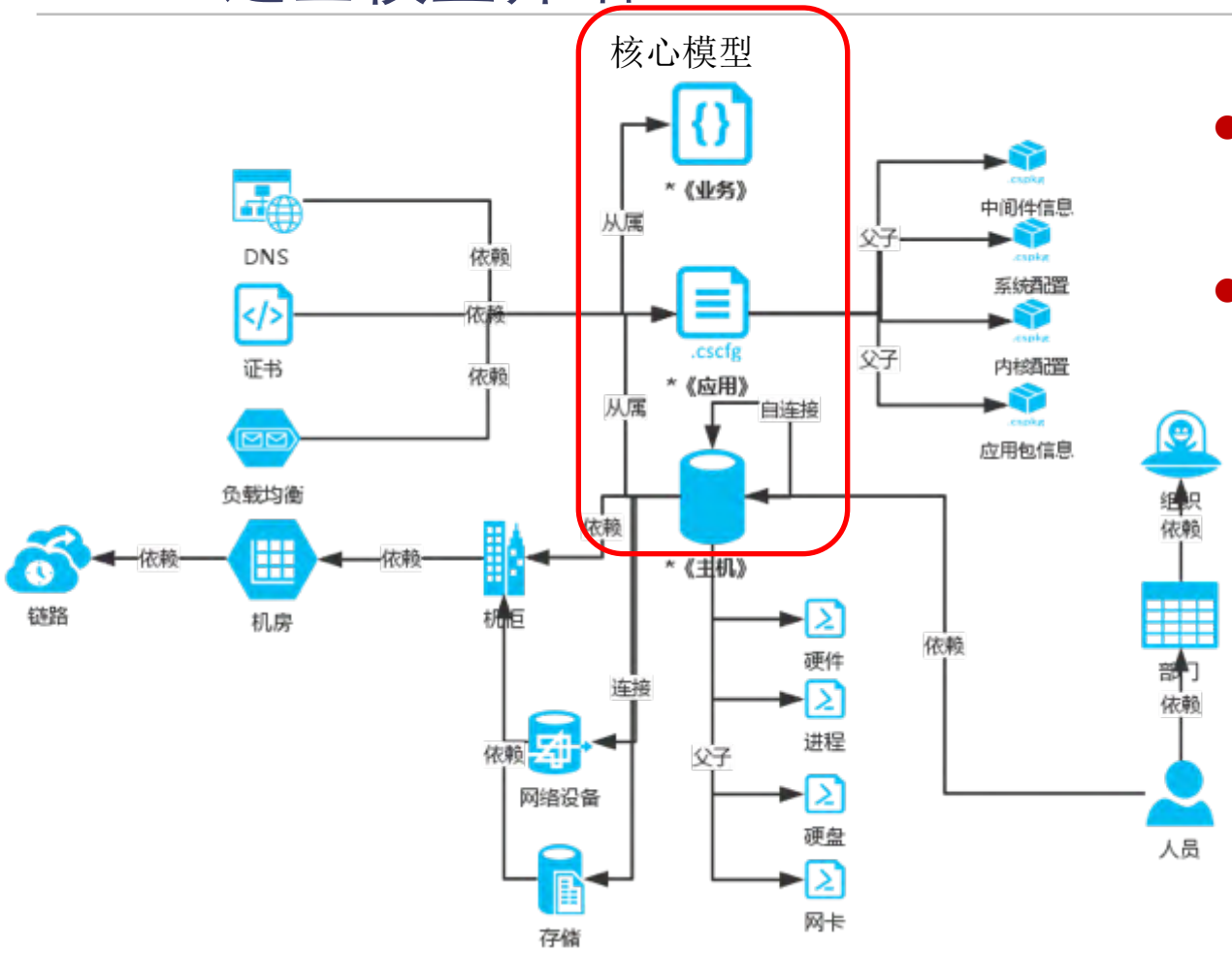
# CMDB，从资产到资源管理的蜕变

# CMDB 逻辑设计（两层架构）



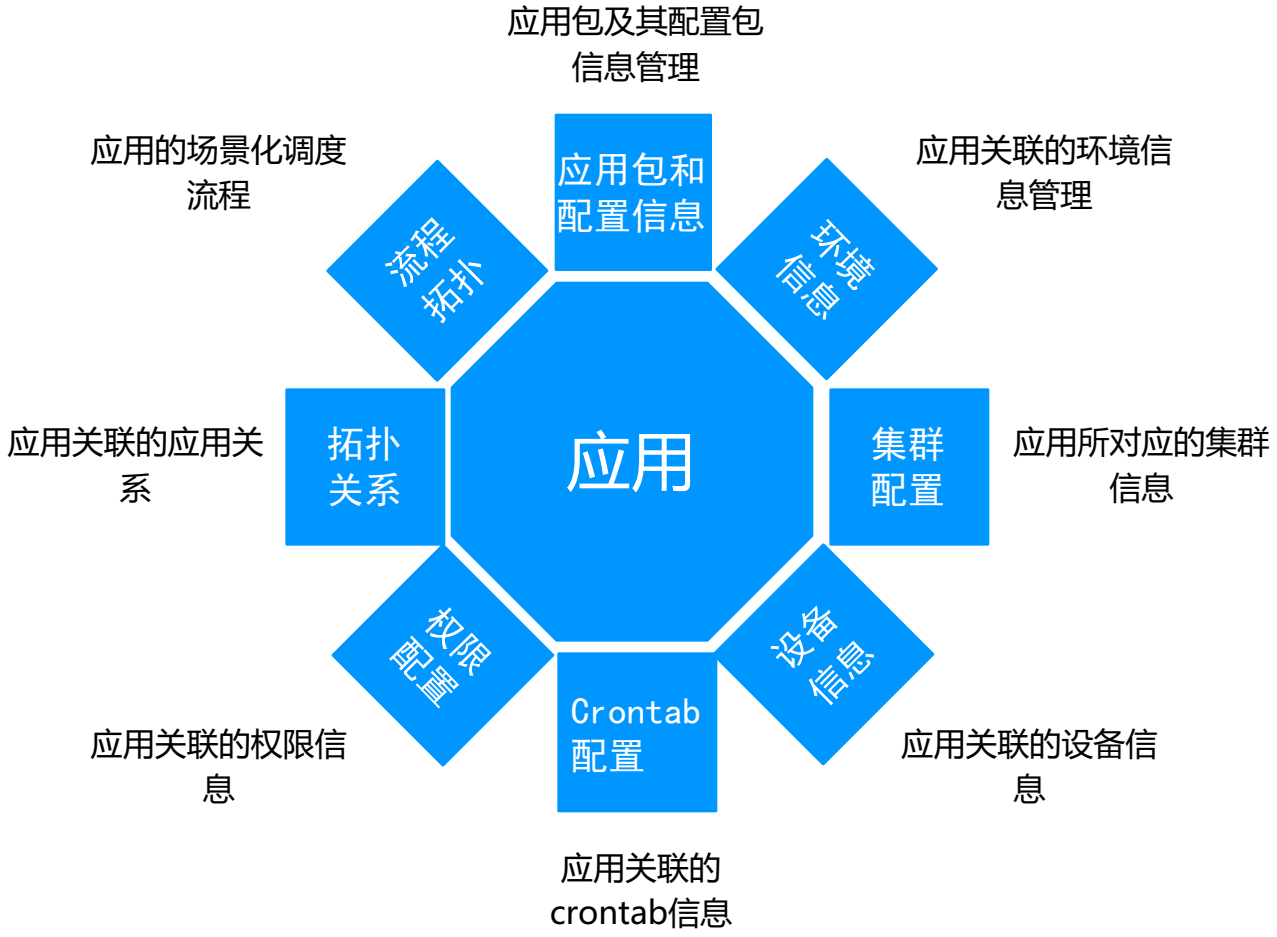

- CMDB架构分基础资源层架构和应用资源层架构
- 应用层资源架构把相关的资源以应用为中心实现资源整合。
- 资源及其资源的关系称之为拓扑（应用拓扑、物理拓扑）
- 资源管理方式有人工维护和自动发现两种方式。流程是人工维护的一种复杂场景和手段。

# CMDB之全模型介绍




- CMDB分核心模型和扩展模型
- 建立以应用为中心的资源管理模型

# CMDB 逻辑设计（应用为主）



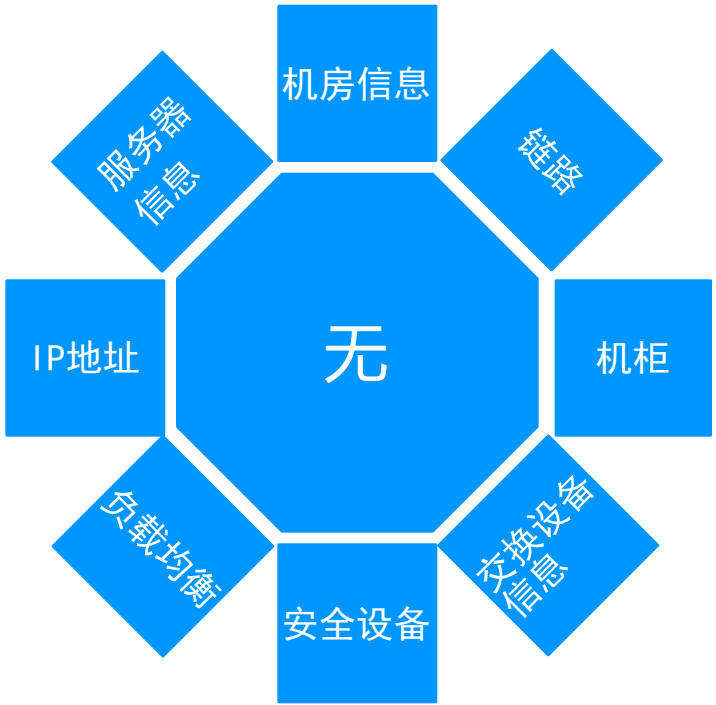

01 业务

02 应用

03 集群

04 主机

# CMDB 逻辑设计（基础资源）



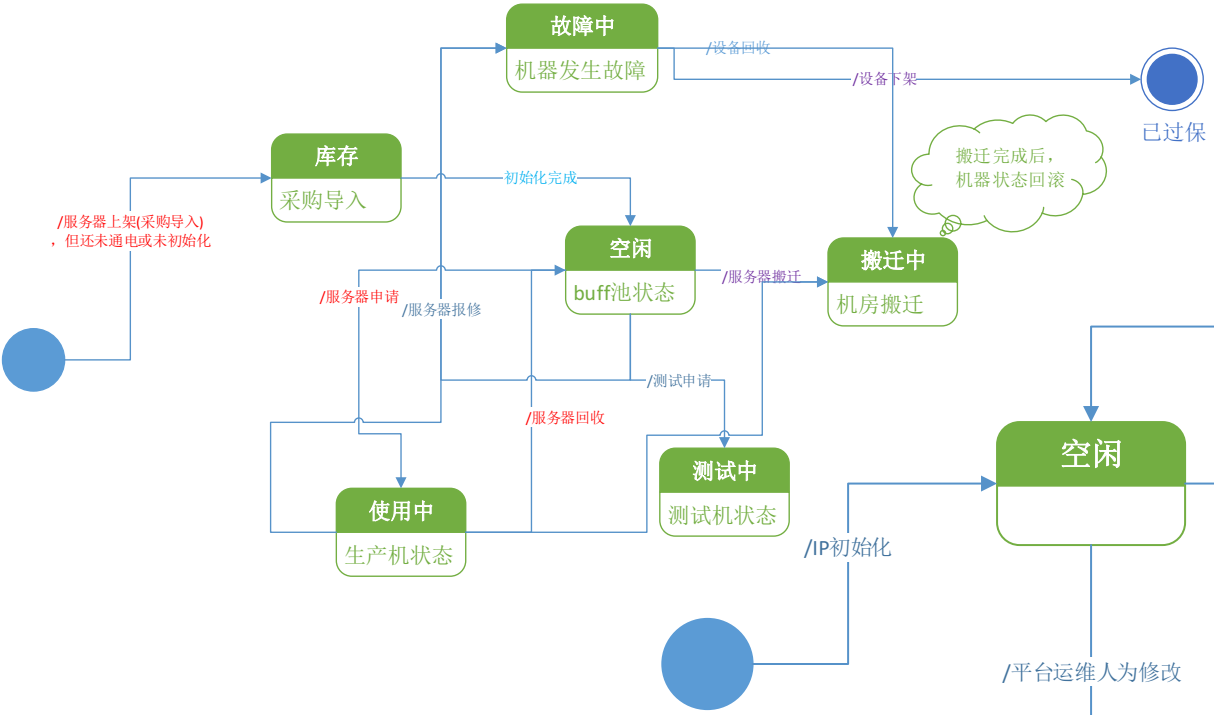

- 物理对象的识别是从机房-》链路-》机柜-》机柜上的设备（网络、安全、负载均衡、路由、波峰）-》IP地址
- 物理资源的关系有些是位置关系决定的，比如说机柜上的设备
- 物理资源的关系有些是连接产生的，比如说服务器和交换机；机房和链路等等
- 基础资源的管理可以通过自动发现和人工维护两种手段

# CMDB之对象与对象属性表

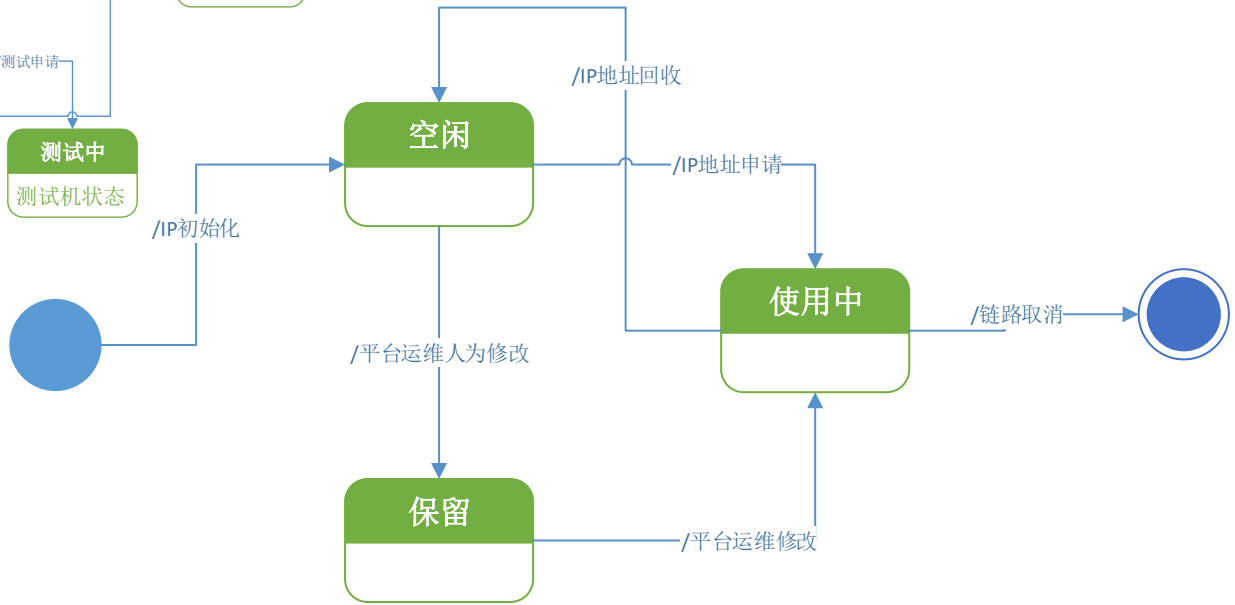
| 属性名称 | 属性定义 | 备注 |
| --- | --- | --- |
| 机房编码 | 由自己的编码规则产生 | |
| 机房名称 | | |
| 状态 | 分“测试中”、“使用中”、“下线”状态，枚举型，示数据 | |
| 机房地址 | | |
| 机房供应商 | | |
| 省份 | | |
| 机房区域 | | |
| 机房商务联系人 | | |
| 机房技术联系人 | | |
| 机房收货联系人 | | |
| 机房值班电话 | | |
| 机房运维负责人 | | |
| 备注 | | |

| 属性名称 | 属性定义 | 备注 |
| --- | --- | --- |
| 域名 | 域名具体的名称比如说 | |
| 域名功能说明 | 详细说明该域名功能、作用 | |
| 域名所属业务 | 最好能把域名和业务关联上 | 可以为空 |
| 域名申请人 | 是哪个开发提出这个域名需求的。和 HR 系统保持一致。 | |
| 域名业务接口人 | 是由哪个运维管理的。和 HR 系统保持一致。 | |
| 域名申请时间 | 什么时候申请的 | |
| 备注 | 注意我们不做域名指向的记录，动态记录，不沉淀到这个系统中，可以做跳转进入到域名系统。 | |

# CMDB之状态图

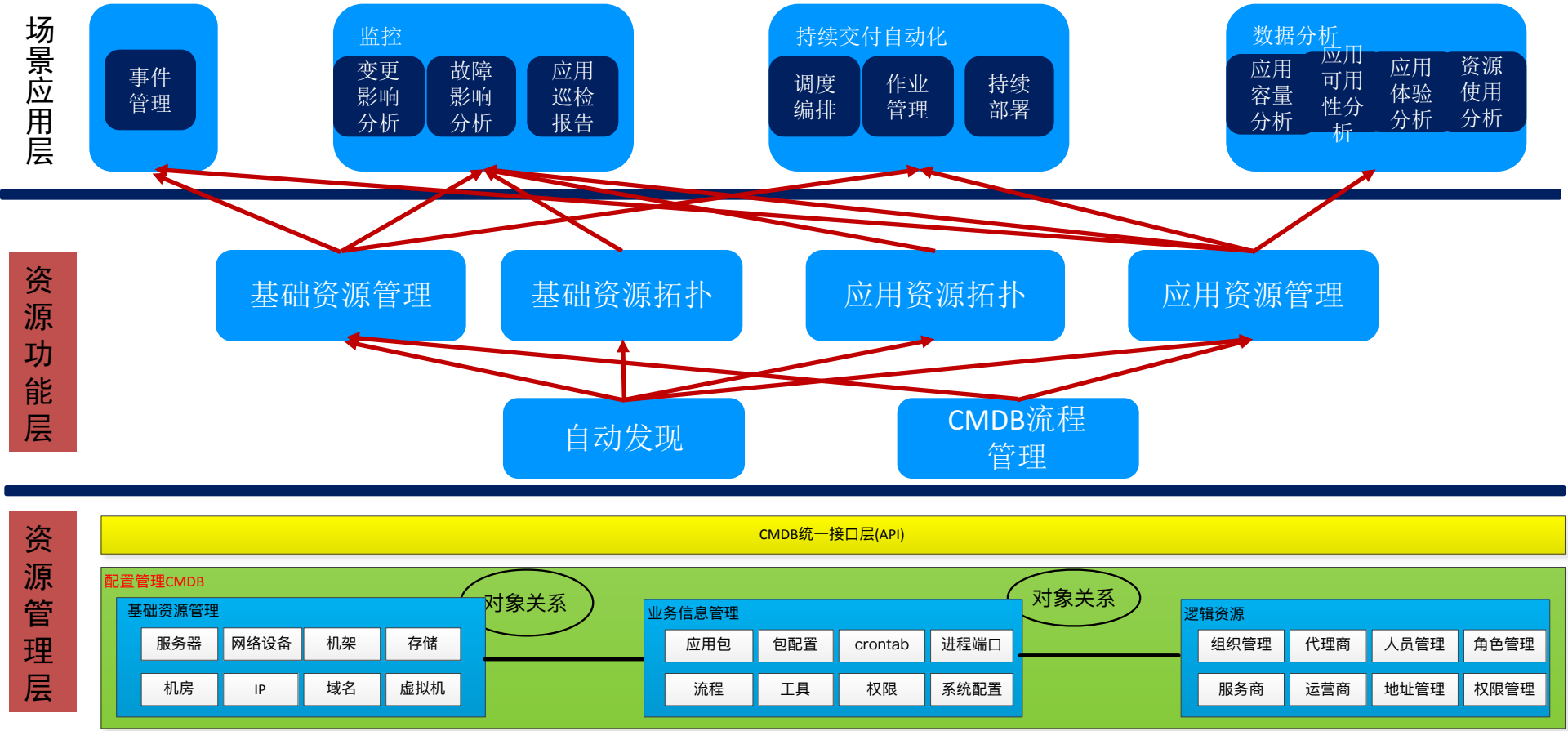



服务器状态转换图

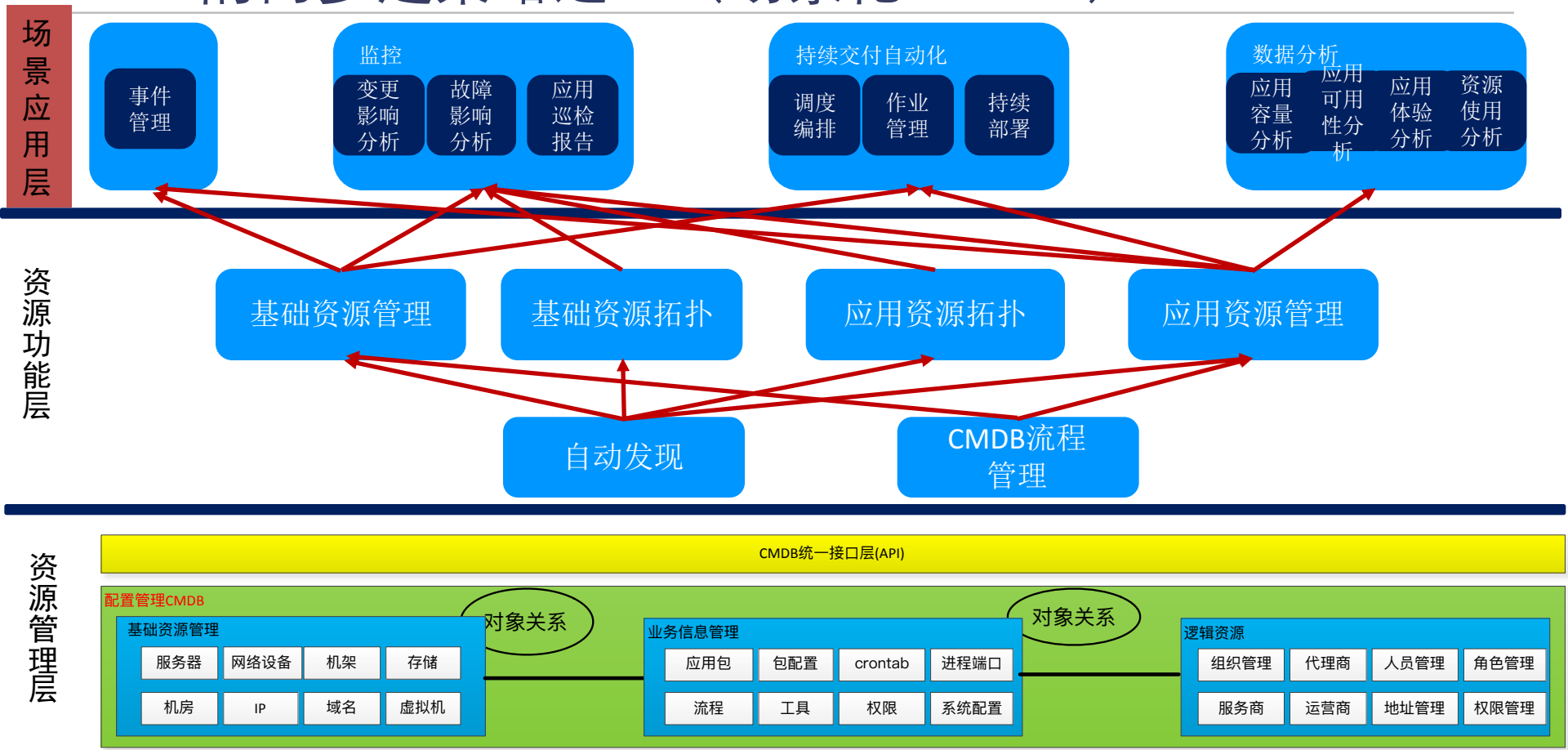



IP地址转换图

# CMDB的两步走策略之一（信息管理）



# CMDB的两步走策略之二（场景化CMDB）



# CMDB的建设讨论

1、CMDB名字应该改一下，叫IT资源管理

2、CMDB分核心模型和扩展模型

3、CMDB对象关系要简化

4、不要太迷信自动发现

5、CMDB要领导支持，团队理解一致

6、云计算的概念层次就是CMDB的层次

7、CMDB是你的资源及组织管理的快照

# DevOps精益工程实践——持续交付

# 什么是持续交付

Continuous Delivery is the ability to get changes of all types—including new features, configuration changes, bug fixes and experiments—into production, or into the hands of users, safely and quickly in a sustainable way.

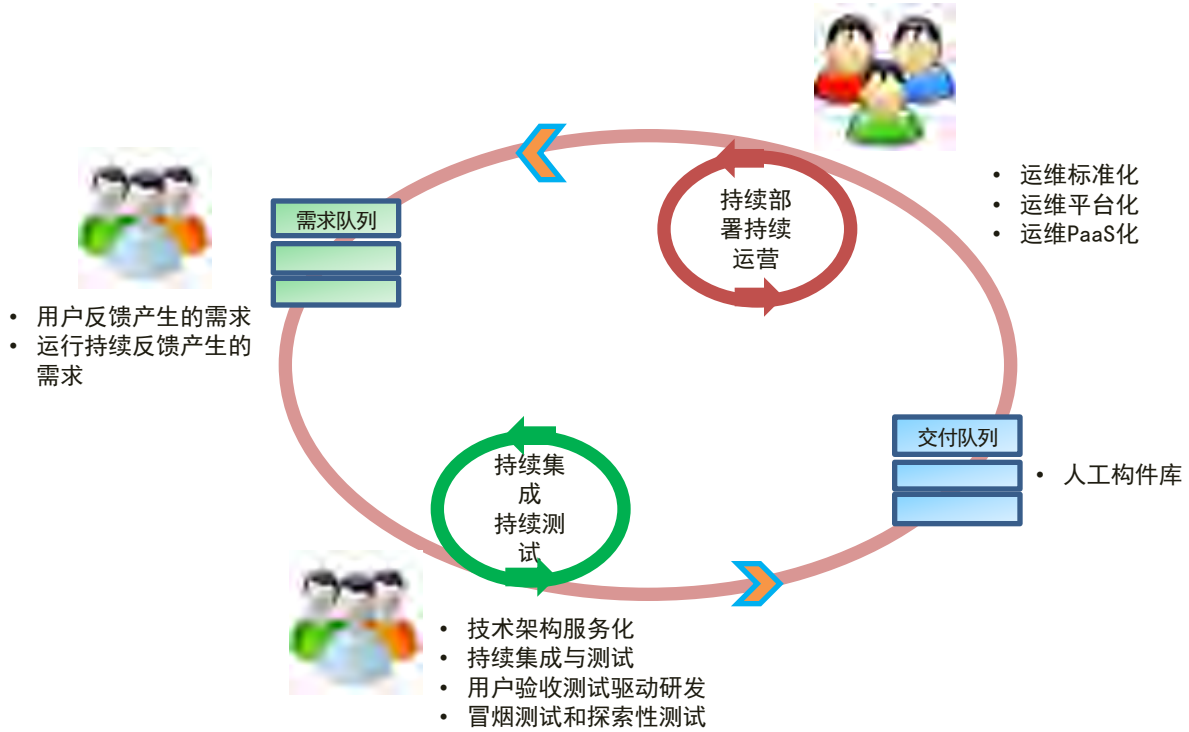
--Jez Humble

- 持续交付是DevOps的最佳工程实践
- 持续交付是IT企业的最佳工程实践

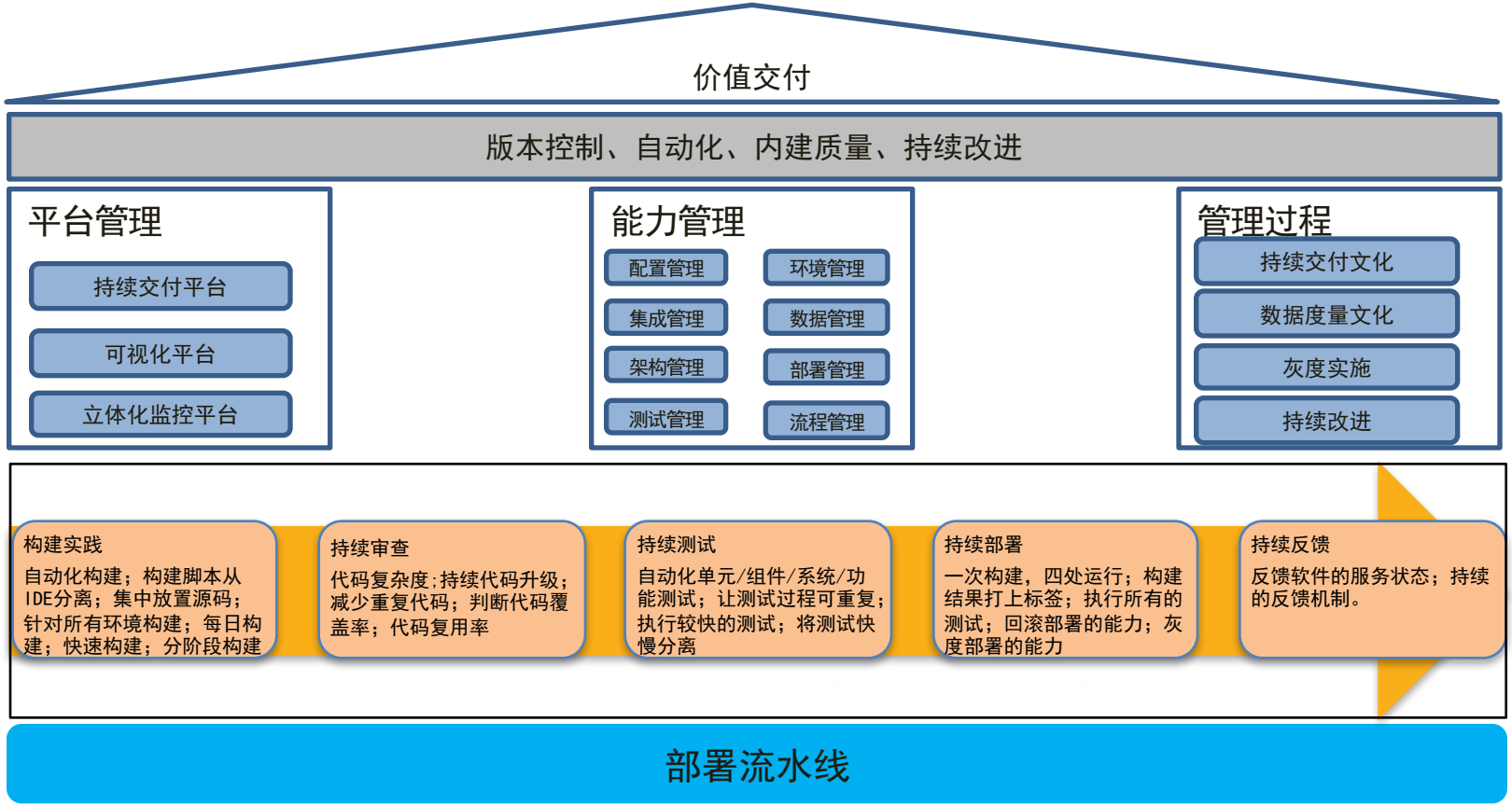
# 持续交付原则

- 为软件的发布创建一个可重复且可靠的过程
- 将几乎所有事情自动化
- 把所有的东西都纳入版本控制
- 提前并频繁地做让你感到痛苦的事情
- 内建质量
- “DONE”意味着“已发布”
- 交付过程是每个成员的责任
- 持续改进

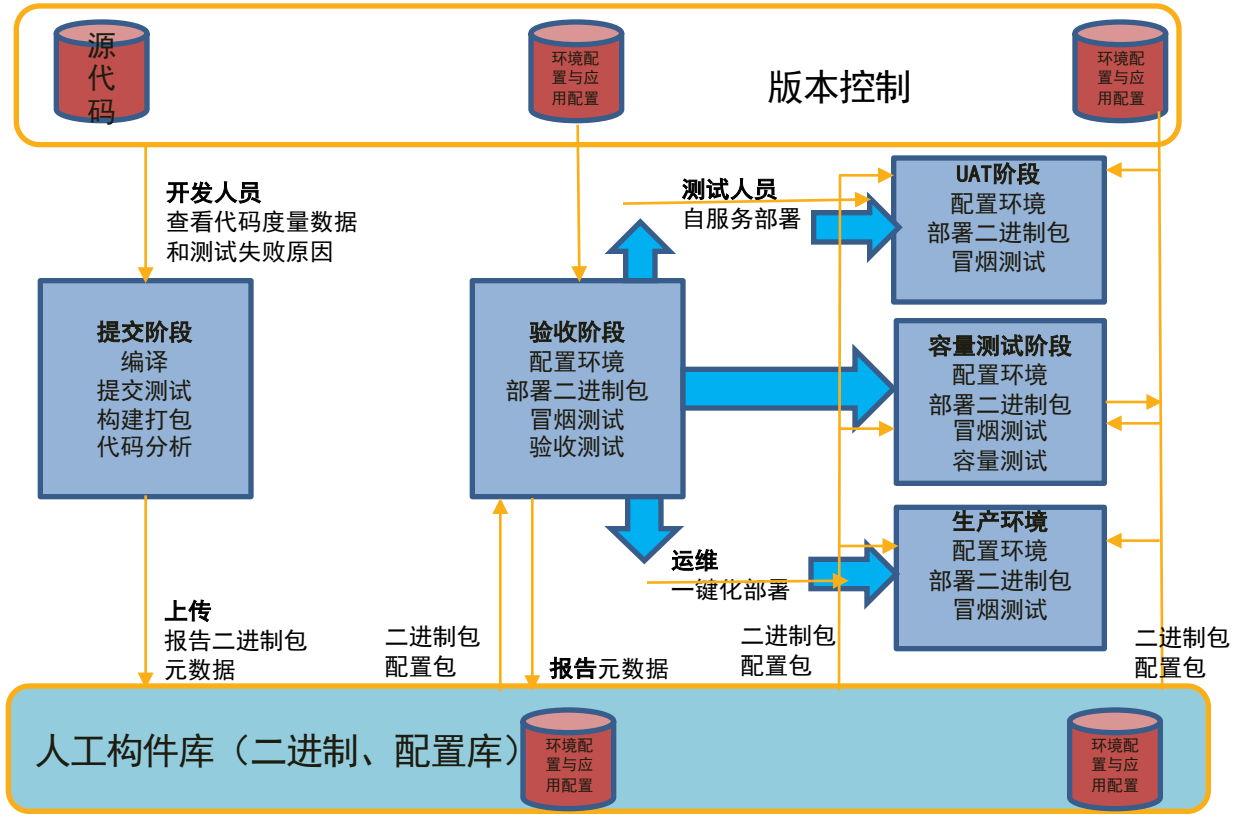
# 持续交付链=IT价值链


需求队列
持续部署持续运营
交付队列
持续集成持续测试
• 用户反馈产生的需求
• 运行持续反馈产生的需求
• 运维标准化
• 运维平台化
• 运维PaaS化
• 人工构件库
• 技术架构服务化
• 持续集成与测试
• 用户验收测试驱动研发
• 冒烟测试和探索性测试

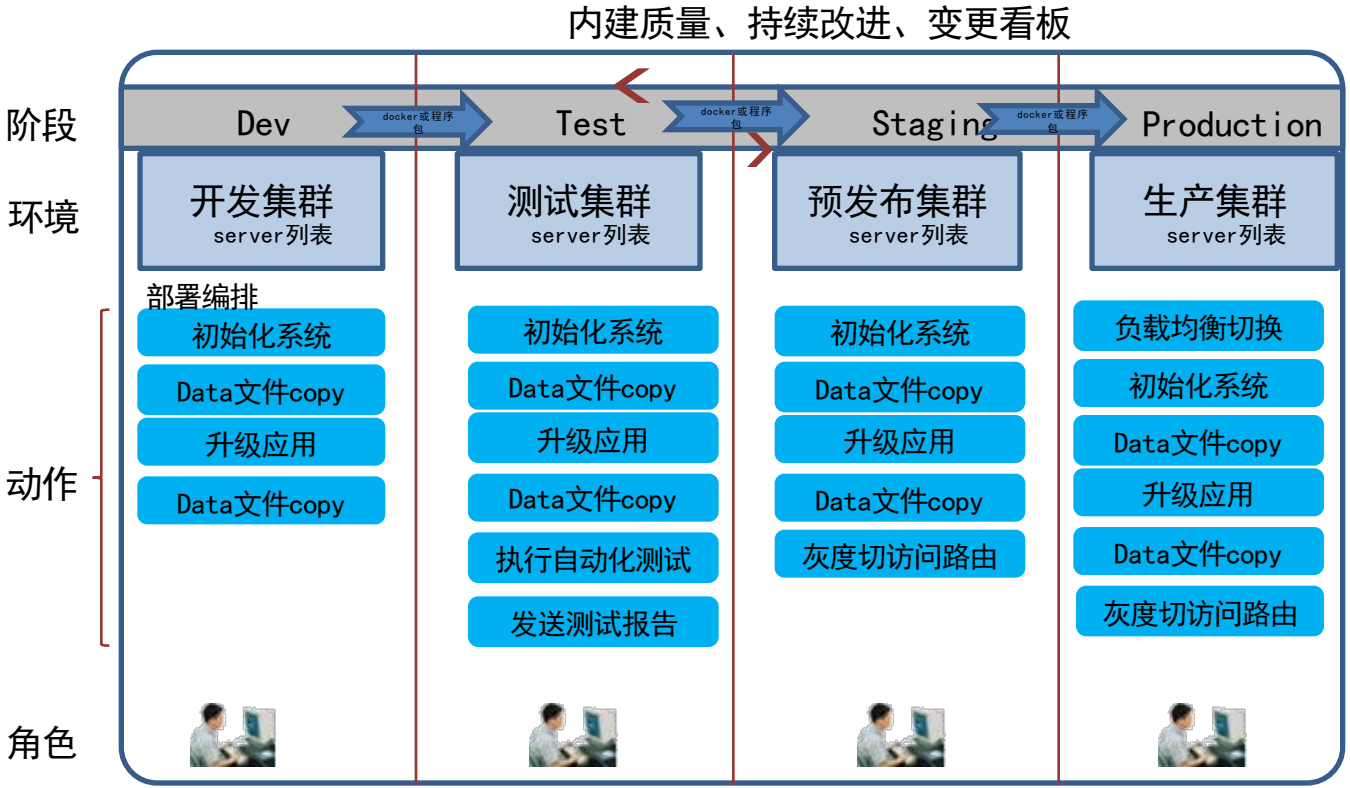
# 持续交付屋


价值交付
版本控制、自动化、内建质量、持续改进
平台管理
持续交付平台
可视化平台
立体化监控平台
能力管理
配置管理
环境管理
集成管理
数据管理
架构管理
部署管理
测试管理
流程管理
管理过程
持续交付文化
数据度量文化
灰度实施
持续改进
构建实践
自动化构建；构建脚本从IDE分离；集中放置源码；针对所有环境构建；每日构建；快速构建；分阶段构建
持续审查
代码复杂度;持续代码升级；减少重复代码；判断代码覆盖率；代码复用率
持续测试
自动化单元/组件/系统/功能测试；让测试过程可重复；执行较快的测试；将测试快慢分离
持续部署
一次构建，四处运行；构建结果打上标签；执行所有的测试；回滚部署的能力；灰度部署的能力
持续反馈
反馈软件的服务状态；持续的反馈机制。
部署流水线

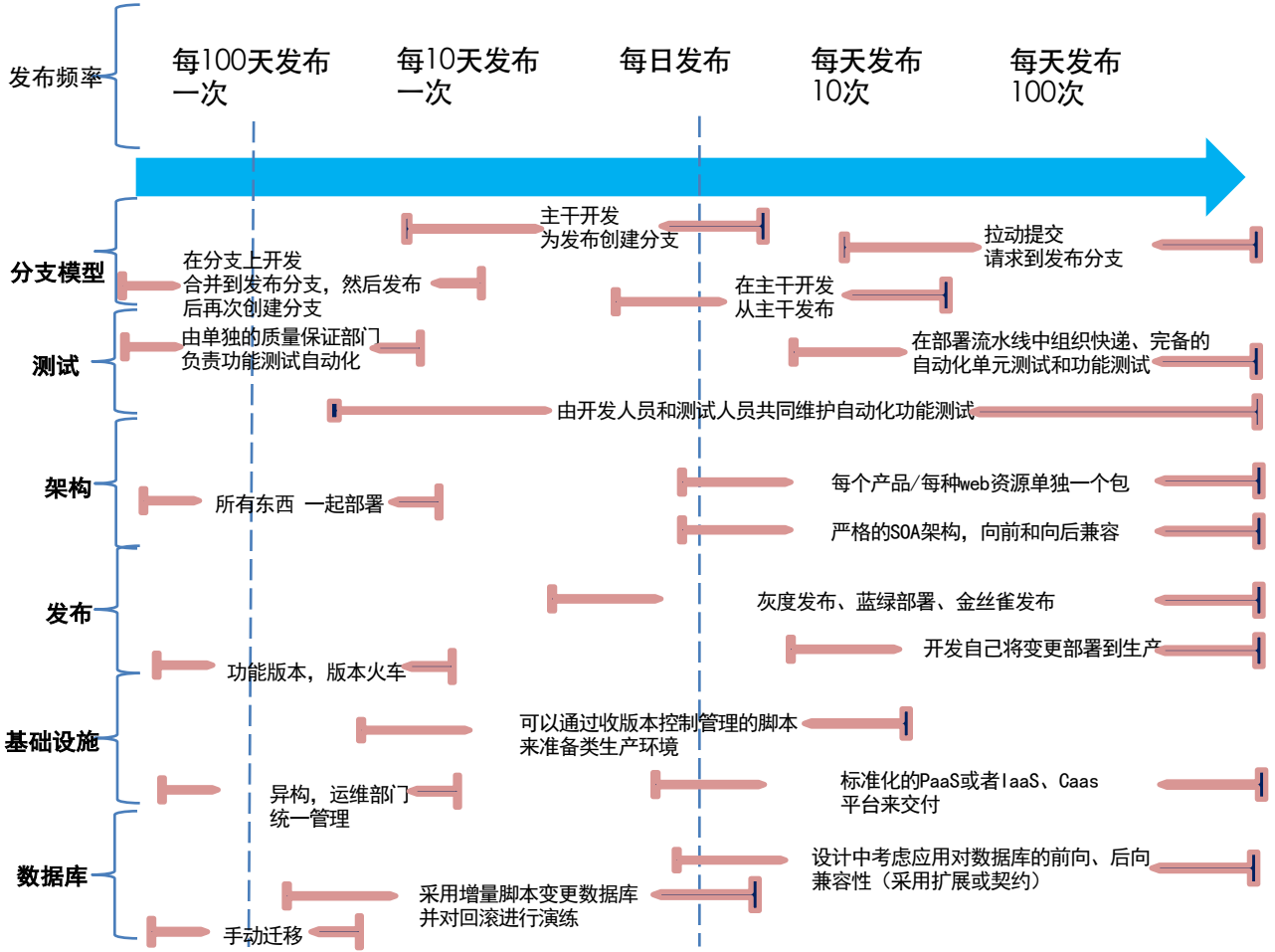
# 交付流水线整体架构


源代码
环境配置与应用配置
版本控制
环境配置与应用配置
开发人员
查看代码度量数据
和测试失败原因
测试人员
自服务部署
UAT阶段
配置环境
部署二进制包
冒烟测试
提交阶段
编译
提交测试
构建打包
代码分析
验收阶段
配置环境
部署二进制包
冒烟测试
验收测试
容量测试阶段
配置环境
部署二进制包
冒烟测试
容量测试
运维
一键化部署
生产环境
配置环境
部署二进制包
冒烟测试
上传
报告二进制包
元数据
二进制包
配置包
报告元数据
二进制包
配置包
二进制包
配置包
人工构件库（二进制、配置库）
环境配置与应用配置
环境配置与应用配置

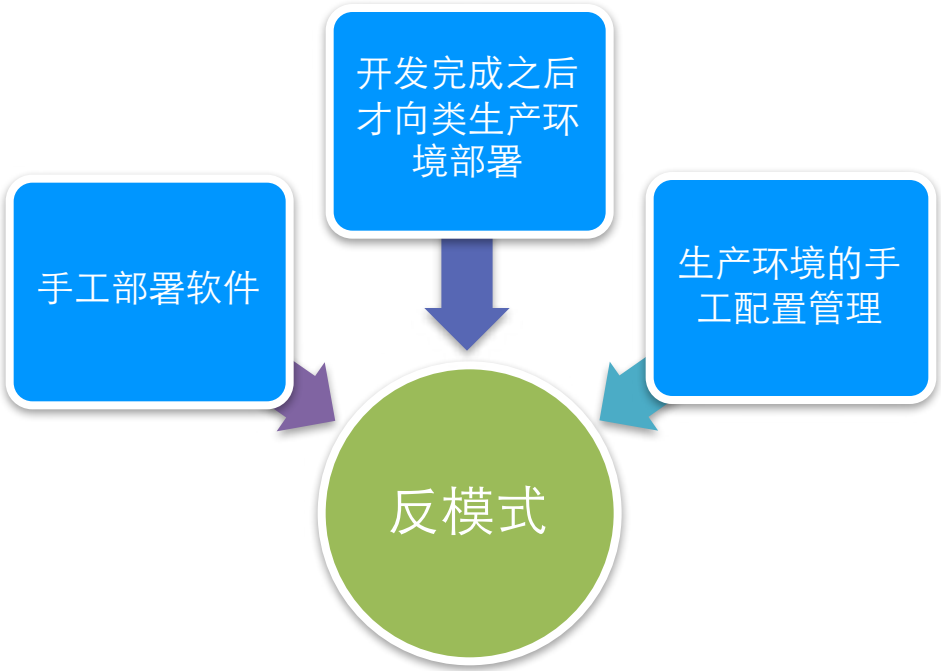
# 持续交付流水线的实现


内建质量、持续改进、变更看板
阶段
环境
动作
角色
Dev
docker或程序包
Test
docker或程序包
Staging
docker或程序包
Production
开发集群
server列表
测试集群
server列表
预发布集群
server列表
生产集群
server列表
部署编排
初始化系统
Data文件copy
升级应用
Data文件copy
初始化系统
Data文件copy
升级应用
Data文件copy
执行自动化测试
发送测试报告
初始化系统
Data文件copy
升级应用
Data文件copy
灰度切访问路由
负载均衡切换
初始化系统
Data文件copy
升级应用
Data文件copy
灰度切访问路由

# 持续交付频率与能力对照表（优化版）

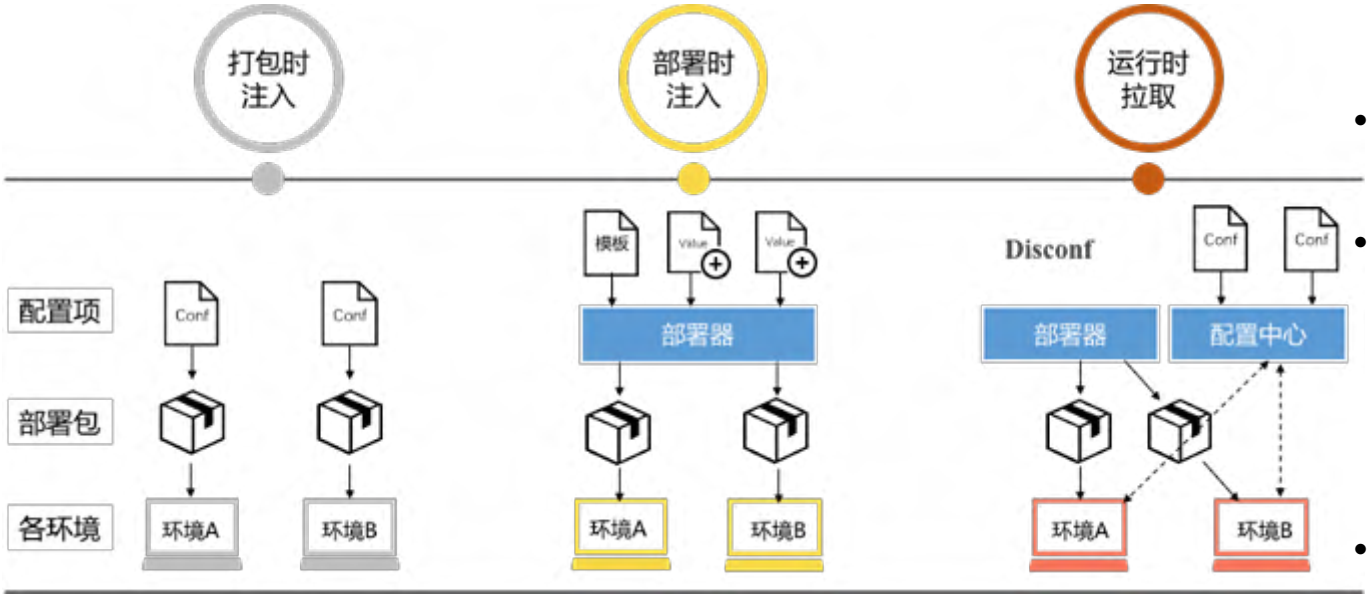


# 持续部署反模式




- 错误的做法限制了IT对业务的能力支撑
- 错误的路上也可能越走越远

# 部署配置管理




- 原始文件管理（File模式）
- 配置项管理（KV模式）
  - 环境变量
  - 数据库集中管理
  - 配置项文件统一管理
- 分布式配置中心管理

# 持续部署之层级标准化视图

| 应用/服务/组件 | 应用标准化 |
| --- | --- |
| 中间件 | 中间件标准化 |
| 操作系统 | 操作系统标准化 |
| 硬件 | |

# 持续部署之应用标准化XY模型

X轴属性 

| | 部署路径 | 属主 | 权限(目录、文件) | 代码 | 配置 | 日志 | ... |
|---|---|---|---|---|---|---|---|
| **JDK** | √ | √ | √ | × | × | × | |
| 组件层 | √ | √ | √ | × | × | × | |
| 公共库和配置 | √ | √ | √ | √ | √ | × | |
| 业务应用层 | √ | √ | √ | √ | √ | √ | |

# 持续部署应用规范之5条原则

## 启动脚本

统一的启动脚本，通过传入参数来匹配不同的业务组件

## 实体隔离

不同的实体部署上必须隔离

## 数据隔离

数据需要写到数据目录或者数据卷上

## 日志实践

日志写到数据或者日志卷上；

规范的输出级别、内容格式、日志种类、轮替周期和定期清理

## 代码和配置

代码包无状态，一次打包，多环境流转；

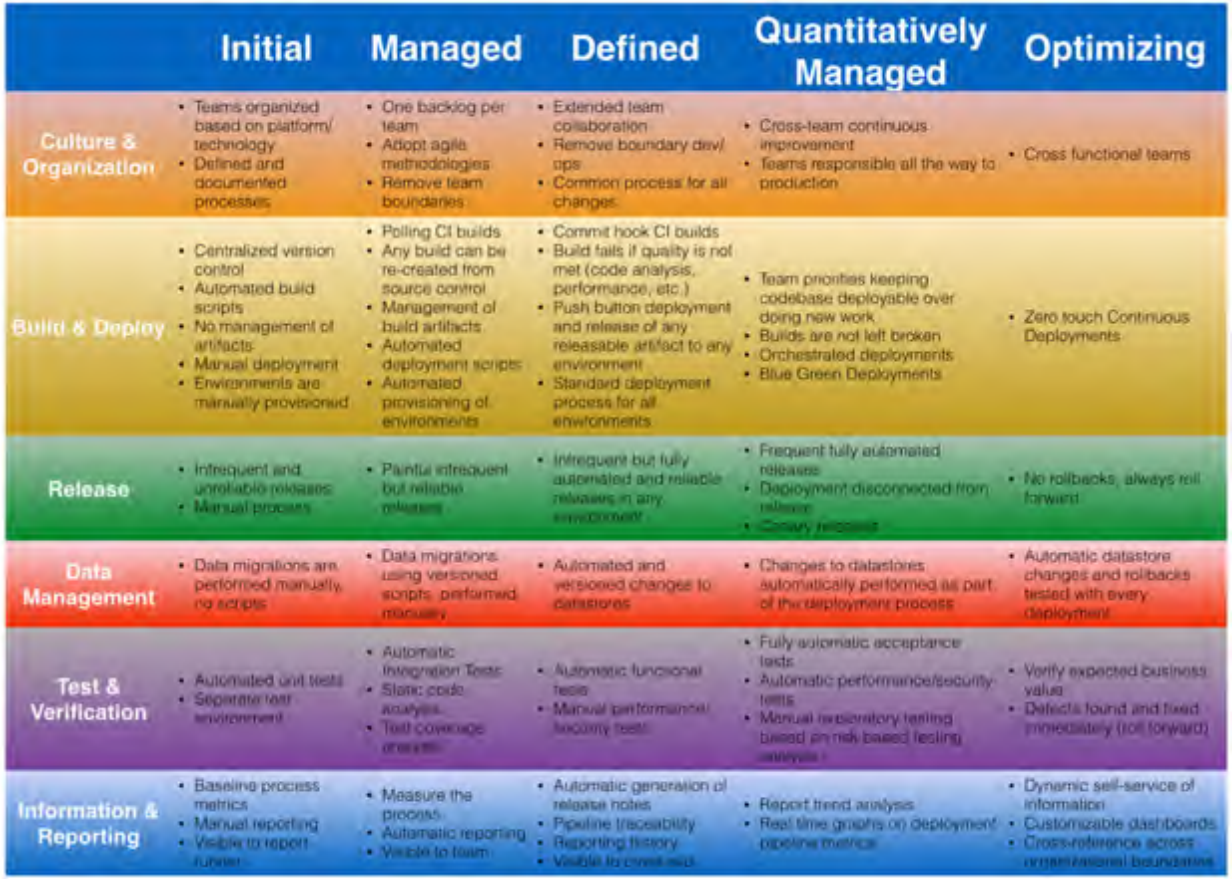
配置包环境相关，甚至可以实现配置中心

# 持续交付成熟参考1（业界）

- 组织&文化
- 设计&架构
- 构建&部署
- 测试&验证
- 信息&报告

From UrbanCode

# 持续交付成熟参考1（业界）



| | Initial | Managed | Defined | Quantitatively Managed | Optimizing |
|---|---|---|---|---|---|
| Culture & Organization | • Teams organized based on platform/ technology<br>• Defined and documented processes | • One backlog per team<br>• Adopt agile methodologies<br>• Remove team boundaries | • Extended team collaboration<br>• Remove boundary dev/ ops<br>• Common process for all changes | • Cross-team continuous improvement<br>• Teams responsible all the way to production | • Cross functional teams |
| Build & Deploy | • Centralized version control<br>• Automated build scripts<br>• No management of artifacts<br>• Manual deployment<br>• Environments are manually provisioned | • Polling CI builds<br>• Any build can be re-created from source control<br>• Management of build artifacts<br>• Automated deployment scripts<br>• Automated provisioning of environments | • Commit hook CI builds<br>• Build fails if quality is not met (code analysis, performance, etc.)<br>• Push button deployment and release of any releasable artifact to any environment<br>• Standard deployment process for all environments | • Team priorities keeping codebase deployable over doing new work<br>• Builds are not left broken<br>• Orchestrated deployments<br>• Blue Green Deployments | • Zero touch Continuous Deployments |
| Release | • Infrequent and unreliable releases<br>• Manual process | • Painful infrequent but reliable releases | • Infrequent but fully automated and reliable releases in any environment | • Frequent fully automated releases<br>• Deployment disconnected from release<br>• Canary releases | • No rollbacks, always roll forward |
| Data Management | • Data migrations are performed manually, no scripts | • Data migrations using versioned scripts, performed manually | • Automated and versioned changes to datastores | • Changes to datastores automatically performed as part of the deployment process | • Automatic datastore changes and rollbacks tested with every deployment |
| Test & Verification | • Automated unit tests<br>• Separate test environment | • Automatic Integration Tests<br>• Static code analysis<br>• Test coverage analysis | • Automatic functional tests<br>• Manual performance/ security tests | • Fully automatic acceptance tests<br>• Automatic performance/security tests<br>• Manual exploratory testing based on risk based testing analysis | • Verify expected business value<br>• Defects found and fixed immediately (roll forward) |
| Information & Reporting | • Baseline process metrics<br>• Manual reporting<br>• Visible to report runner | • Measure the process<br>• Automatic reporting<br>• Visible to team | • Automatic generation of release notes<br>• Pipeline traceability<br>• Reporting history<br>• Visible to cross-team | • Report trend analysis<br>• Real time graphs on deployment pipeline metrics | • Dynamic self-service of information<br>• Customizable dashboards<br>• Cross-reference across organizational boundaries |

# 持续交付成熟参考2（业界）

- 初始级（人工发布）
- 管理级（计划发布）
- 定义级（有规则发布）
- 数据驱动级（按需发布）
- 优化级（商业驱动/创新）

# 持续交付成熟模型（3）

初始级

管理级

已定义级

量化管理级

优化级

# 持续交付成熟模型的8个能力维度

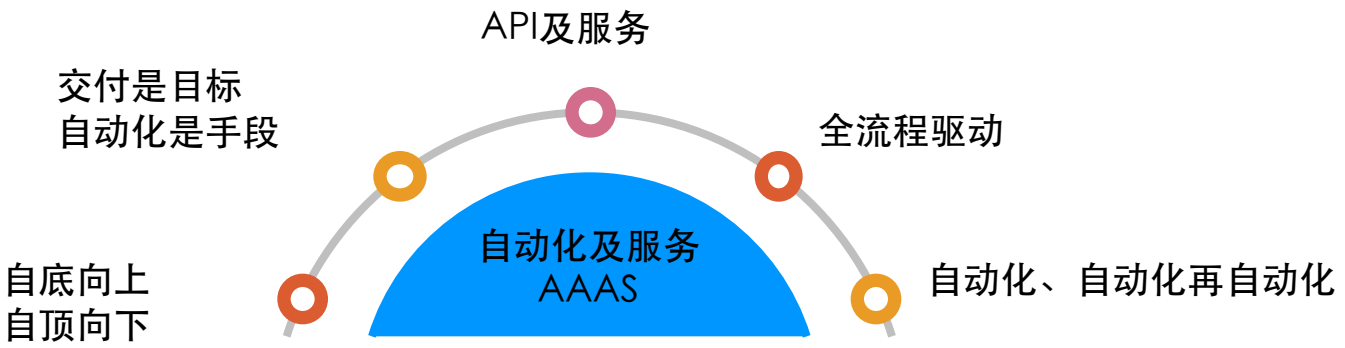
- 持续集成
- 环境管理
- 部署管理
- 发布管理
- 测试管理
- 数据管理
- 配置管理
- 架构管理

# 持续交付最佳实践

| 能力维度 / 成熟度 | 持续管理和持续集成 | 环境管理 | 部署管理 | 发布管理 | 测试管理 | 数据管理（DBA） | 配置管理 | 架构管理 |
|---|---|---|---|---|---|---|---|---|
| 5级 - 优化管理级 | 团队定期召开会议，讨论集成问题，并利用自动化 / 更快地反馈和更好的可视化来解决问题。 | 更有效地管理所有环境，可以使用更新的技术来准备环境，比如docker。 | 准备工作全面自动化。如果适当的话，使用虚拟化和容器技术来完成环境的快速部署。 | 运营和交付团队定期沟通来管理风险，缩短发布周期。 | 生产环境回滚很少，如回滚可以立即发现并修复。 | 在两次发布之间建立了数据库变更和回滚流程的反馈回路。 | 定期验证配置管理是否之间高效的合作，快速开发和可审计的变更管理流程 | **所有技术架构中涉及到的组件都是服务化的。技术架构中的接口也都是服务化封装的。可以无感知变更** |
| 4级 - 量化管理级 | 构建度量收集，可视化并平衡行动。构建不能长时间处于失败状态 | 所有环境的管理都纳入了监控监控能力范围，提供了容量 / 可用性 / 一致性等多种监控手段。 | 精心计划的部署管理，对发布和回滚流程进行测试。对于变更后对重要有指标的监控能力，自动变更过程异常。 | 环境与应用程序的健康性得到监控，并有能量性管理：变更前置时间得到关注和监控。对于变更后对重要不能指标的监控能力，自动变更过程异常。 | 质量度量和趋势指标，对于非功能需求进行了定义和度量 | 每次部署都进行数据库的升级和回滚测试。对数据库进行监控和优化。 | 开发人员每天至少提交到主干一次。只在需要发布时才会创分支。 | 架构的服务化能力可以从监控 / 管理 / 调用能力 / 调用状态等多个角度衡量。 |
| 3级 - 已定义级 | 每次代码提交都进行自动化构建和测试。依赖关系被很好的管理。脚本和工具得到很好的重用。 | 开发 / 测试 / 生产甚至其它各种环境的标准化和规范化能力是一致的。建立了标准的环境管理过程。 | 软件部署使用全自动自助服务一键式过程。使用相同的过程向各种环境部署。 | 定义并执行变更管理和审批过程。监管及合规的条件得到满足。 | 自动化的单元测试盒验收测试。后者是测试人员写的。测试是开发过程一部分 | 数据库变更作为部署流程的一部分自动执行。 | 库和依赖被很好的管理。变更过程决定了版本控制的使用规则。 | 对关联的技术架构服务变更可以通过其提供的API接口来调用完成 |
| 2级 - 可管理级 | 定期的自动化构建与测试。任意一个构建都可以自动化过程重新从源版本控制库上构建 | 对应用或及当的环境是有清晰的职责定义和owner定义，并在统一可视化管理。 | 自动化部署到几种环境中。新环境的创建成本非常低。所有的配置都放置在外部，并做版本控制。 | 痛苦且不频繁，但能可靠地发布。从需求到发布可以看到部分可追踪 | 自动化测试是团队协作开发的一部分 | 对数据库的变更使用自动化脚本完成，而这些脚本与应用的版本相对应 | 重建软件所需的内容都进行了版本控制，包括：源代码 / 配置 / 构建和部署脚本 / 数据迁移 | 所有服务的变更是可以通过webconsoie完成的。可视化完成 |
| 1级 - 初始级 | 软件构建是手工过程。没有对产物和报告进行管理 | 几乎没有环境管理的能力，应用的环境分布是混乱且不可知的。 | 软件部署是手工过程。针对具体环境生成二进制包。环境准备是手工的。 | 不频繁且不可重复的发布 | 开发完之后，才做手工测试 | 数据库变更没有版本化，且手工进行 | 没有使用版本控制，或者提交不频繁 | 毫无架构设计准则，架构设计是无序和混乱的 |

8个能力管理的最佳实践，详细【中国应用运维规范】

# 关于交付/自动化的一点观点




所有敲命令完成的运维管理都应该转换成鼠标动作

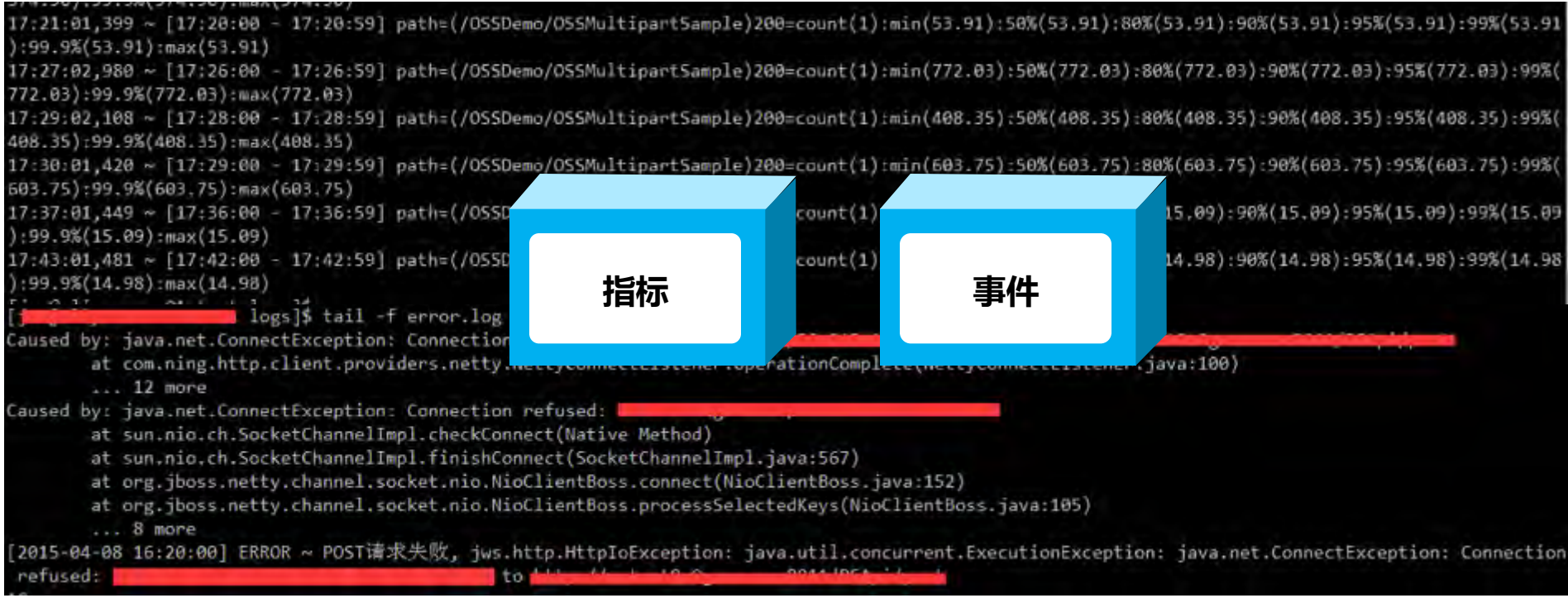
# 从监控到数据化运营的蜕变

# 为什么需要数据

除了上帝,任何人都必须用数据说话

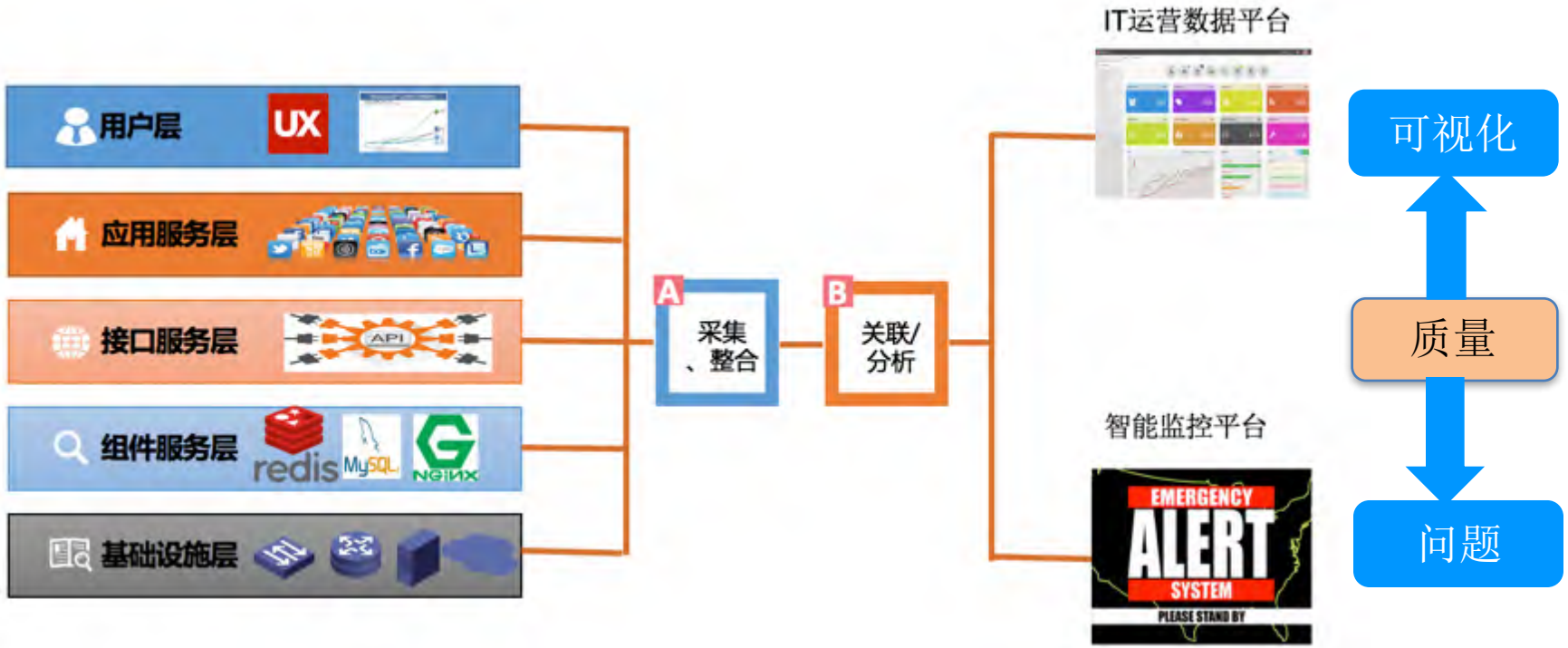
**爱德华.戴明**

# IT运营数据的本质




数据的技术本质

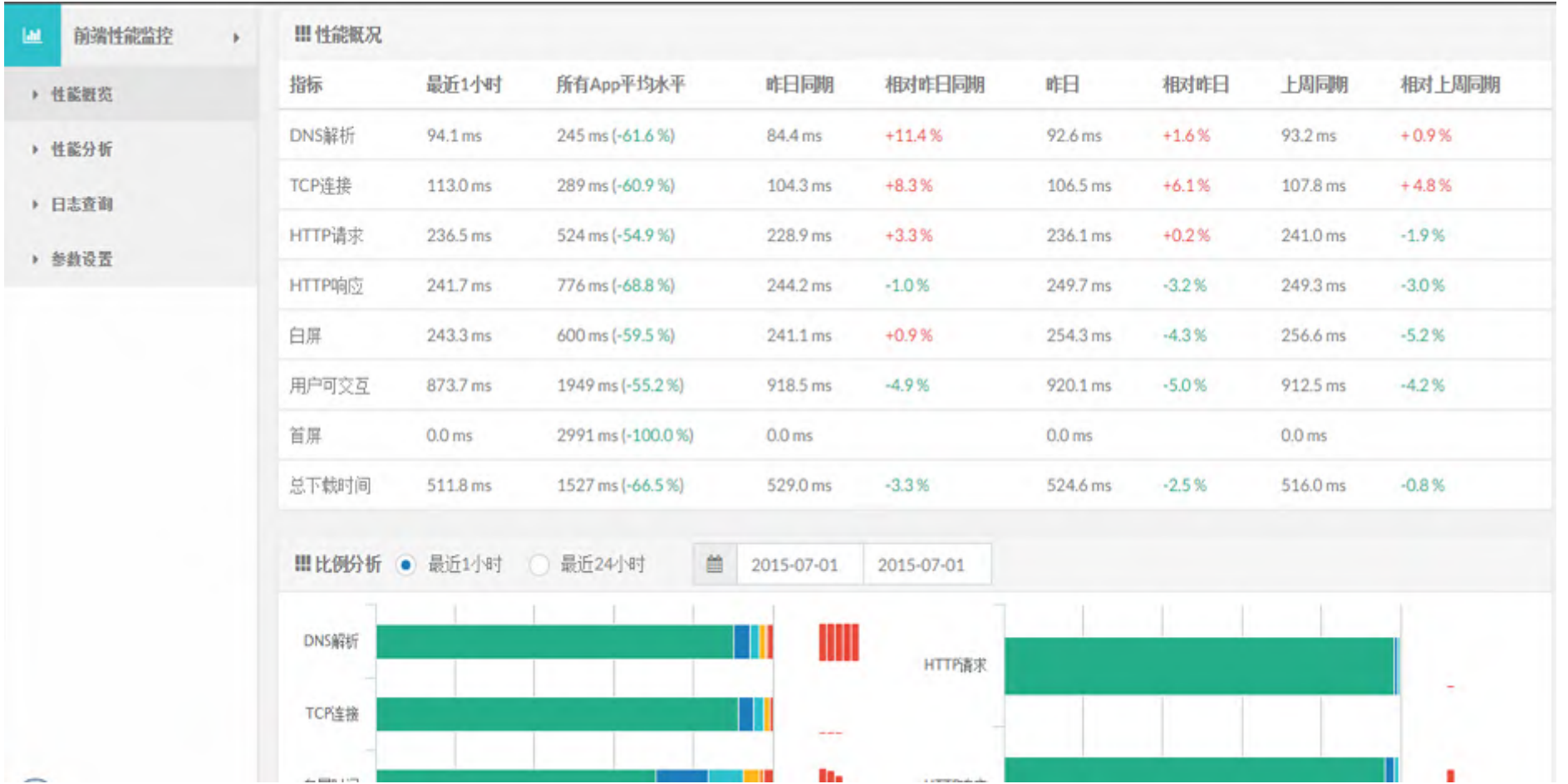
# 数据运营的两种路径




- 数据化平台是裂变了告警监控和运营分析两个平台
- 智能监控负责问题处理能力闭环
- 运营分析负责数据化驱动决策和优化闭环

# IT运营数据的分层体系

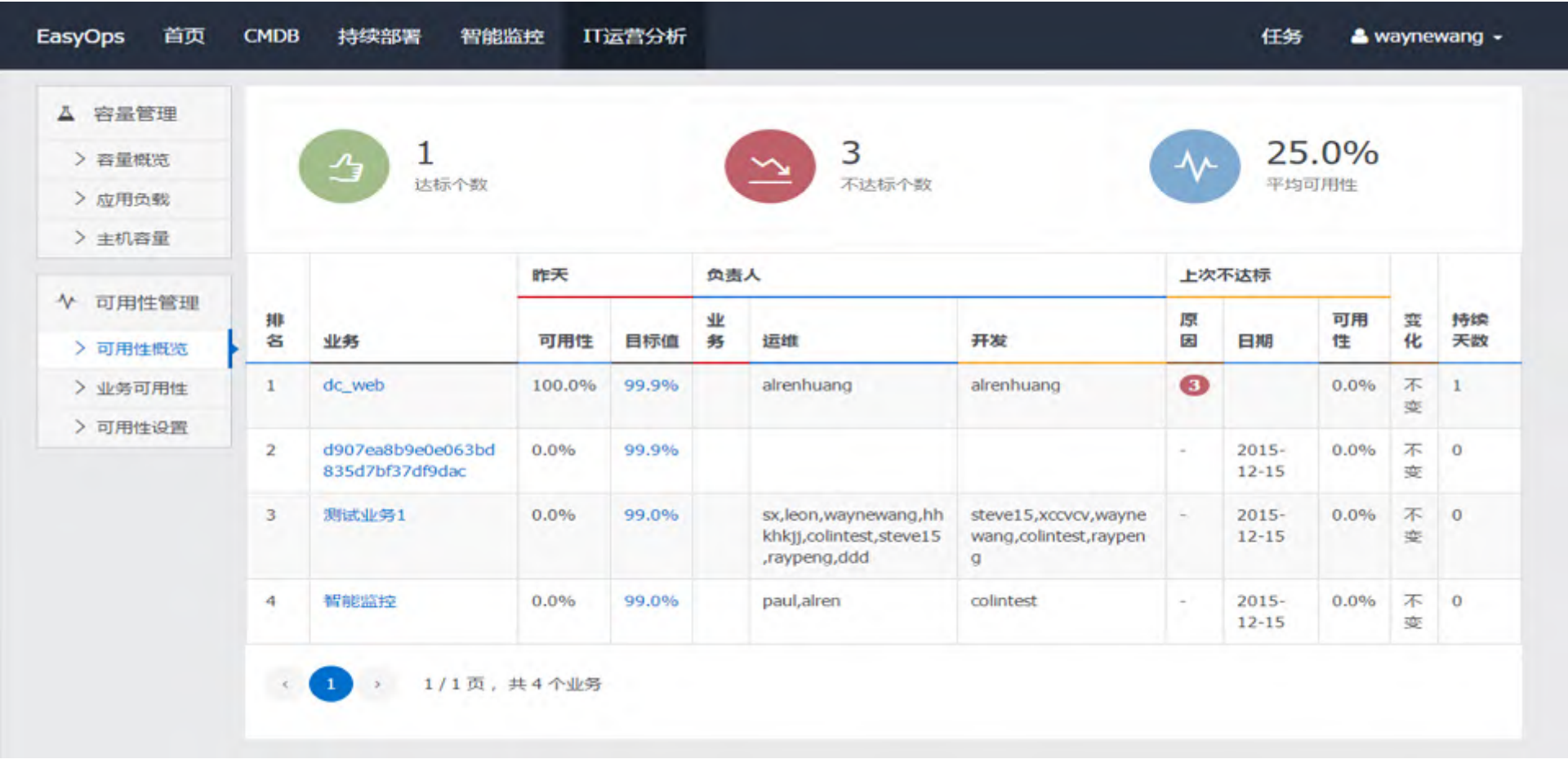
# IT运营之用户体验管理



# IT运营之容量管理

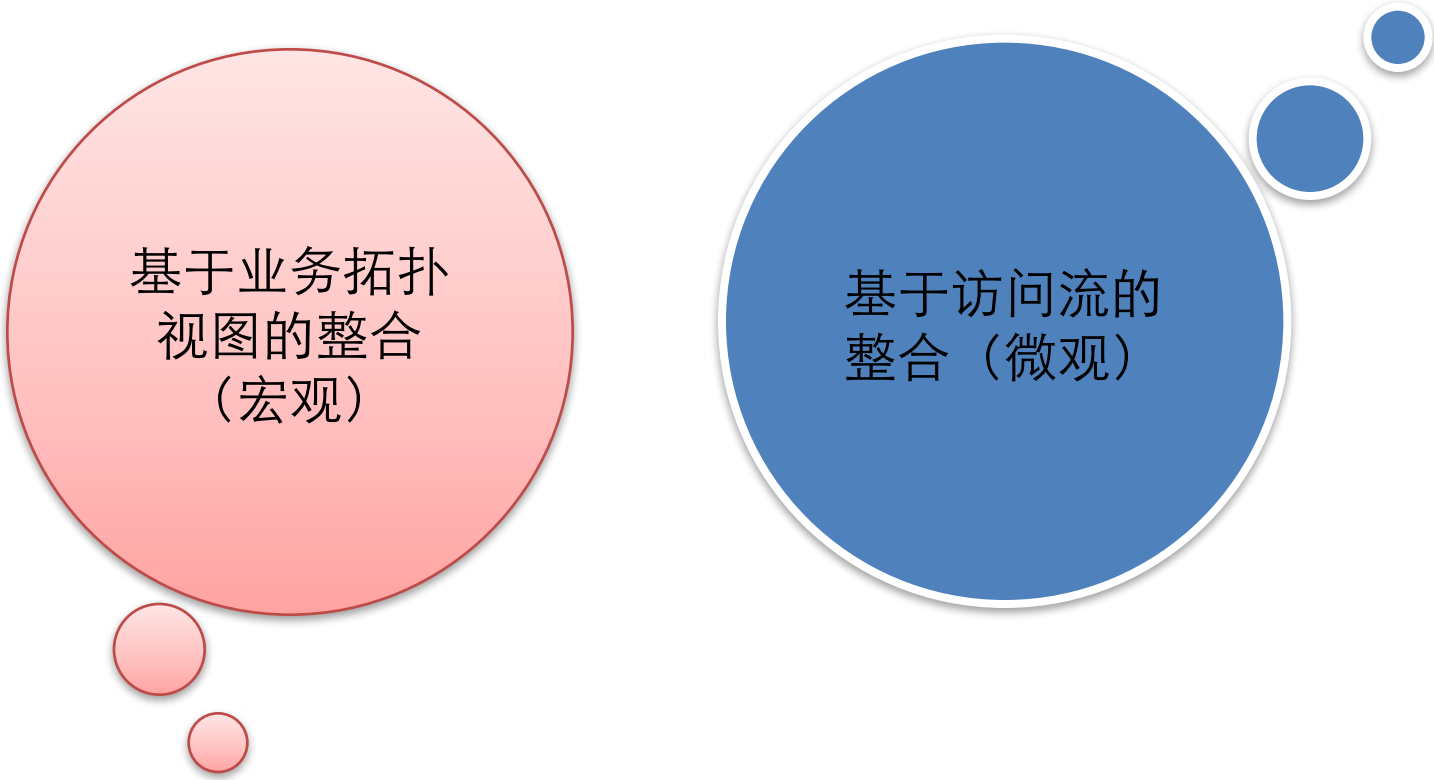

| 应用 | 最高负载 | 平均负载 | 设备总数 | 负载率 | | | 负责人 |
|---|---|---|---|---|---|---|---|
| | | | | 高(80~)% | 中(30~80)% | 低(~30)% | |
| im | 92.4% | 43.0% | 100 | 8.0% | 84.0% | 8.0% | colin |
| sdk-app | 99.9% | 43.0% | 100 | 13.0% | 79.0% | 8.0% | leon |
| biz-db | 65.6% | 42.3% | 100 | 0.0% | 87.0% | 13.0% | daliang |
| sdk-web | 99.9% | 34.4% | 100 | 10.0% | 83.0% | 7.0% | david |

# IT运营之业务可用性




| 排名 | 业务 | 昨天 | | 负责人 | | | 上次不达标 | | | 变化 | 持续天数 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 可用性 | 目标值 | 业务 | 运维 | 开发 | 原因 | 日期 | 可用性 | | |
| 1 | dc_web | 100.0% | 99.9% | | alrenhuang | alrenhuang | 3 | | 0.0% | 不变 | 1 |
| 2 | d907ea8b9e0e063bd835d7bf37df9dac | 0.0% | 99.9% | | | | - | 2015-12-15 | 0.0% | 不变 | 0 |
| 3 | 测试业务1 | 0.0% | 99.0% | | sx,leon,waynewang,hhkhkjj,colintest,steve15,raypeng,ddd | steve15,xccvcv,waynewang,colintest,raypeng | - | 2015-12-15 | 0.0% | 不变 | 0 |
| 4 | 智能监控 | 0.0% | 99.0% | | paul,alren | colintest | - | 2015-12-15 | 0.0% | 不变 | 0 |

# IT运营之数据集成




离散的数据没有任何意义

# 基于应用架构的全局拓扑视图

# 架构服务化，从OFFLINE到ONLINE

# 为什么需要架构服务化


每个组件的可用性<1，乘积的放大效应


公共服务让业务的试错成本越来越低


简化运维管理，提高可运维性

# 架构服务化之架构失控

服务间调用：配置、DNS、LVS、链路...


**负载均衡**

LVS
F5
Haproxy+keepalive
Nginx+keepalive

**接入层**

Nginx
Tomcat
Resin
Jetty
自研

**逻辑层**

私有程序
Tomcat
Resin
Apache

**Cache服务器**

Memcache
Redis

**文件服务器**

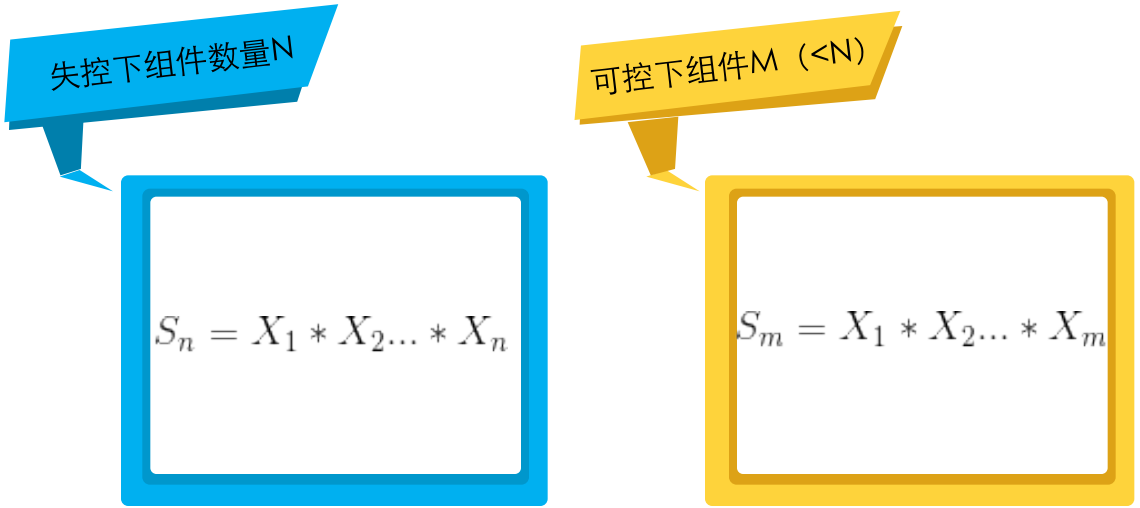
Localstorage
Ftp
Mfs
Fastdfs
Tfs

**存储服务器**

Mysql
Mongodb
Cassandra
Redis

## 架构中的点和线构成的失控

# 架构失控的统计学阐释




组件越多，意味着出问题概率越高，可用性越低(A<B)

# 架构服务化的目标


服务透明
容错性、位置透明、名字服务
可管理性
可视化管理，一切可配置，场景化
可运维性
可监控性
服务的数据采集和监控能力
自服务
服务最终自助化管理，产品化的要求

# 架构服务化的层次化理解


运维者服务和研发者服务统一，Dubbo、EDAS、Spring Cloud

# 架构服务化之组件服务化（例子）


组件服务化是PaaS化的核心能力

# 传统服务调用方式带来的质量问题

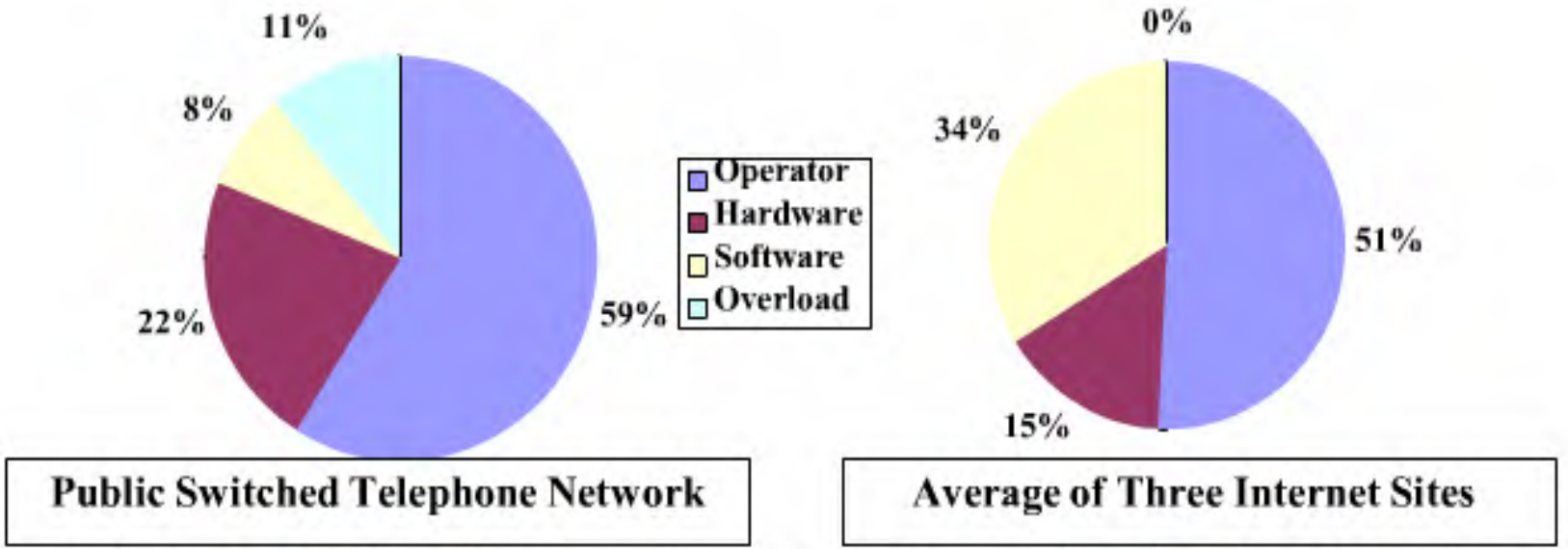
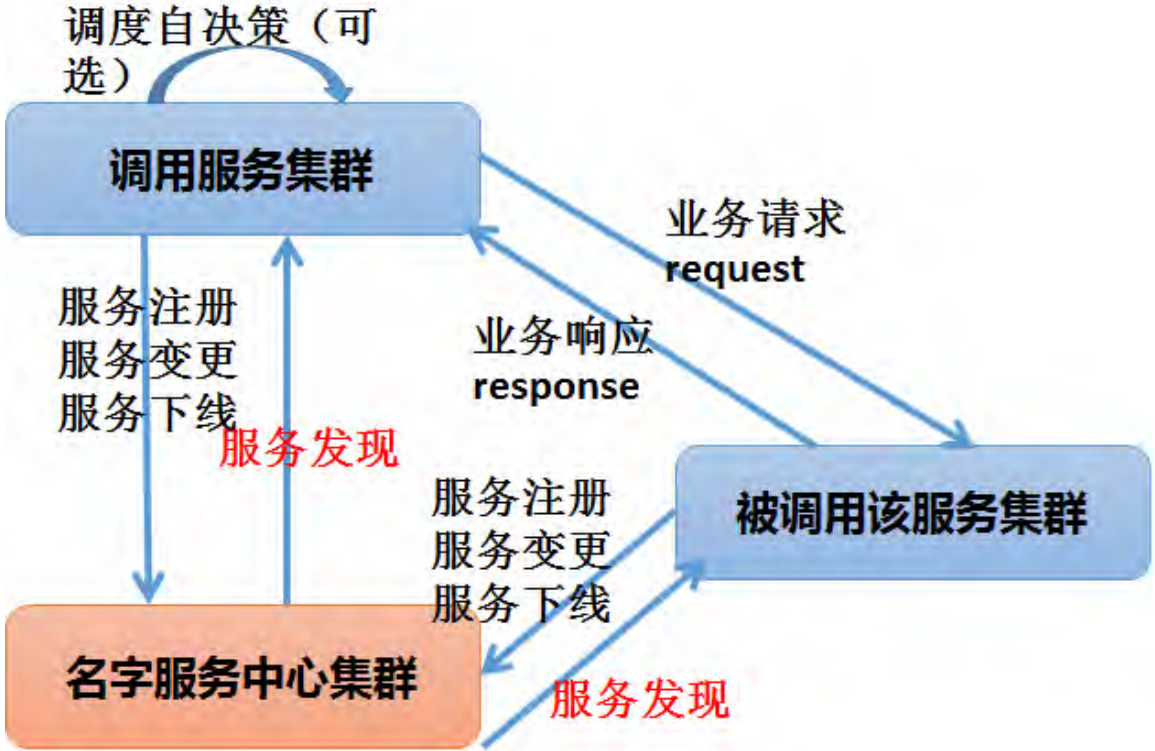



Figure 2. Percentage of failures by operator, hardware, software, and overload for PSTN and three Internet

技术架构运行时应该剔除人的因素

# 新名字服务的特性（高级）



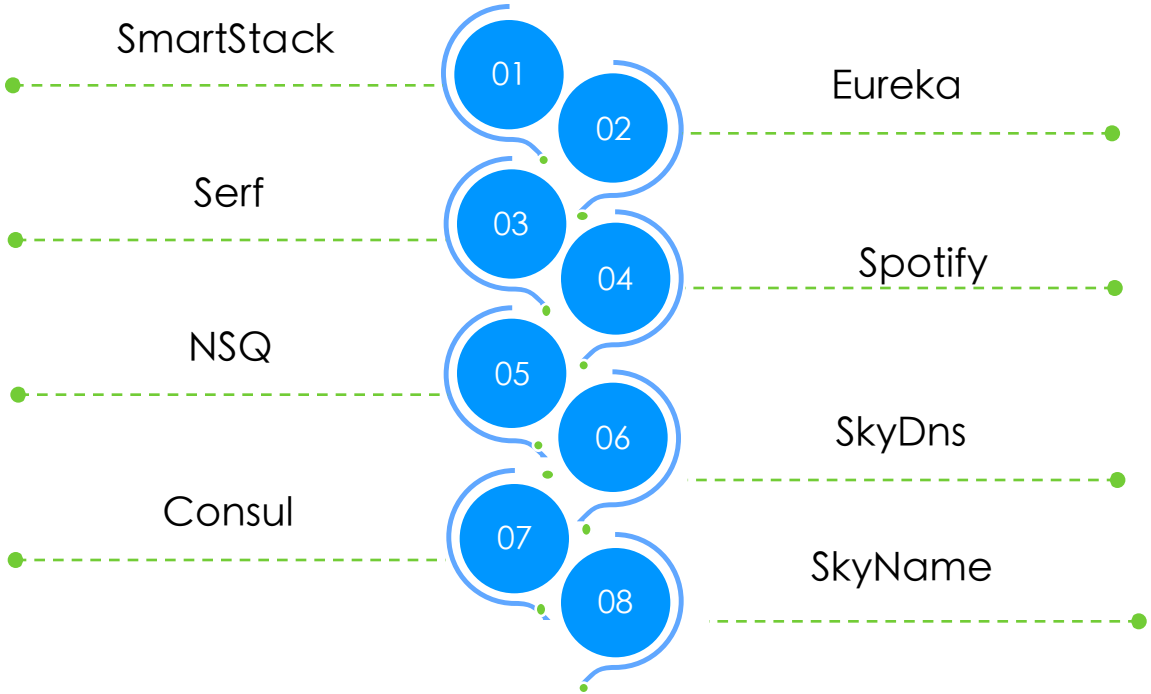

**服务注册**

能够完成服务的人工或者自动注册

**服务发现**

服务调用端能够对被调用端做自动的服务发现

# 架构服务化之名字服务中心



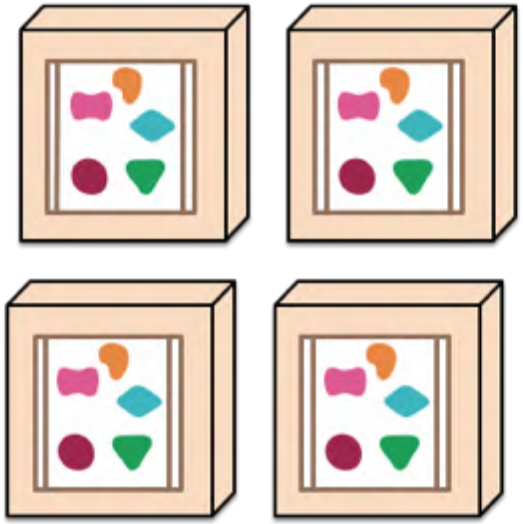
# 组件服务化架构之经验


【如何化解产品和技术之间的矛盾】

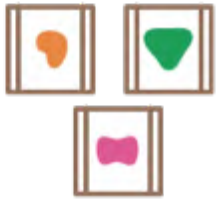
# 巨石架构 VS 微服务架构

*A monolithic application puts all its functionality into a single process...*

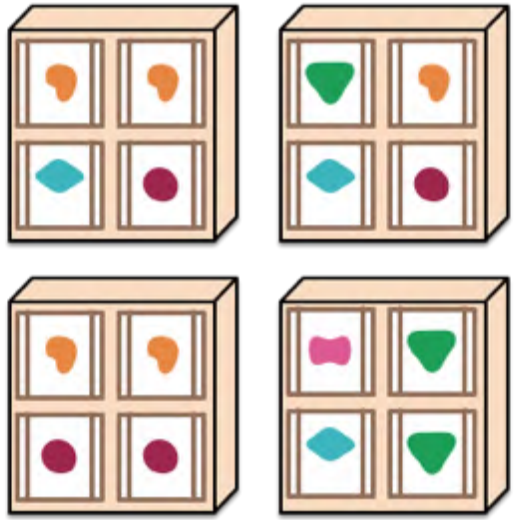
*... and scales by replicating the monolith on multiple servers*



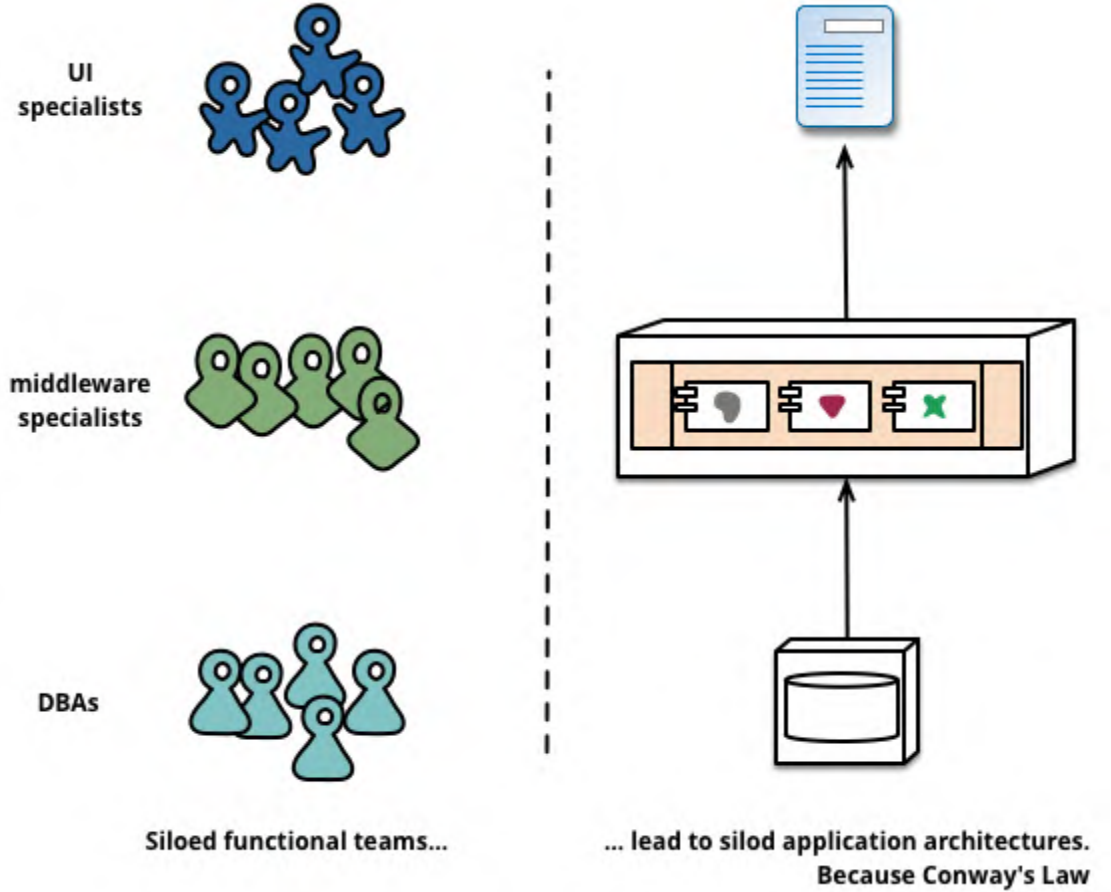
*A microservices architecture puts each element of functionality into a separate service...*



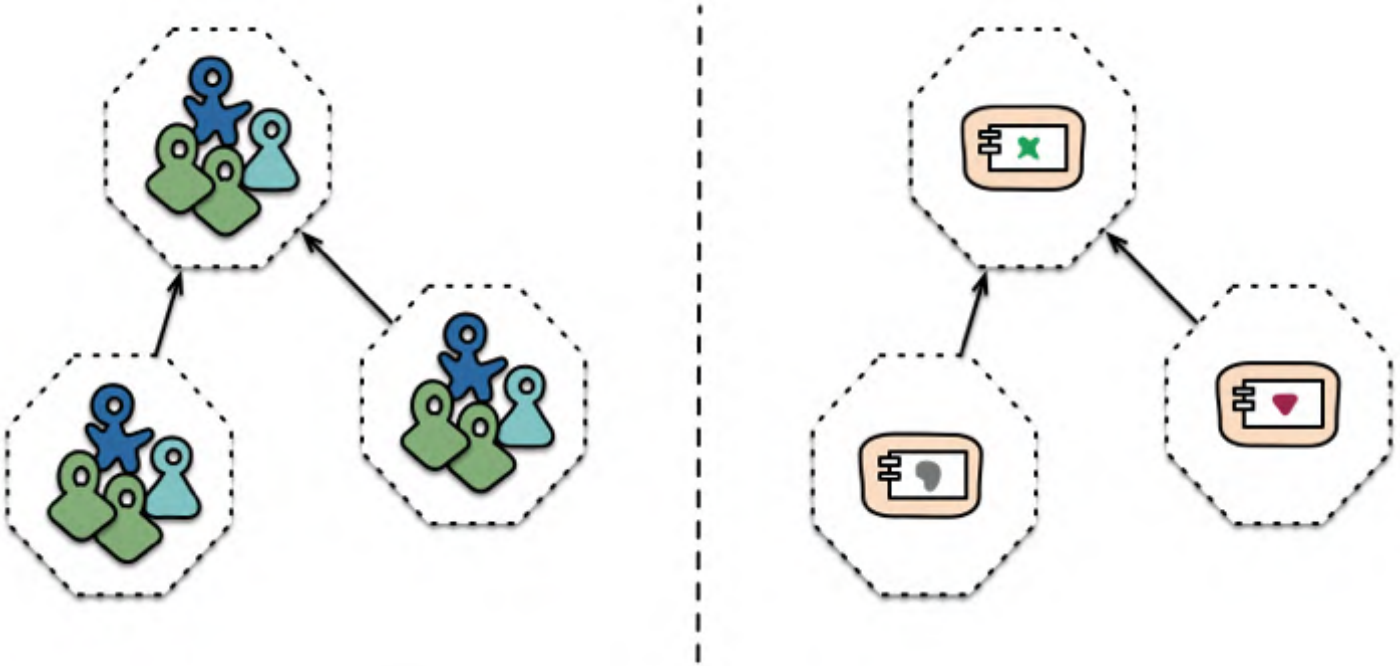
*... and scales by distributing these services across servers, replicating as needed.*



# 传统巨石架构 的组织根源



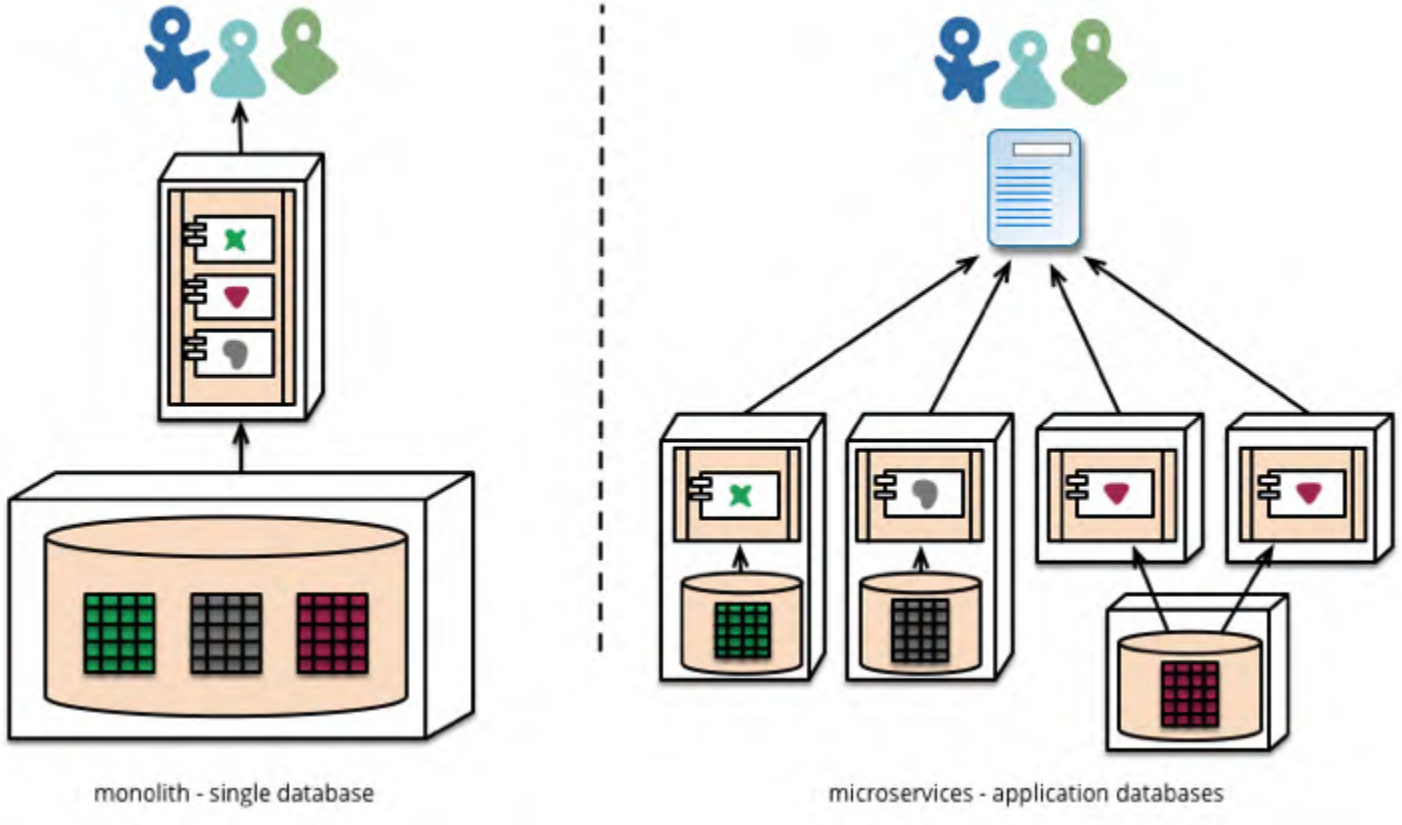
# 微服务架构 的组织根源



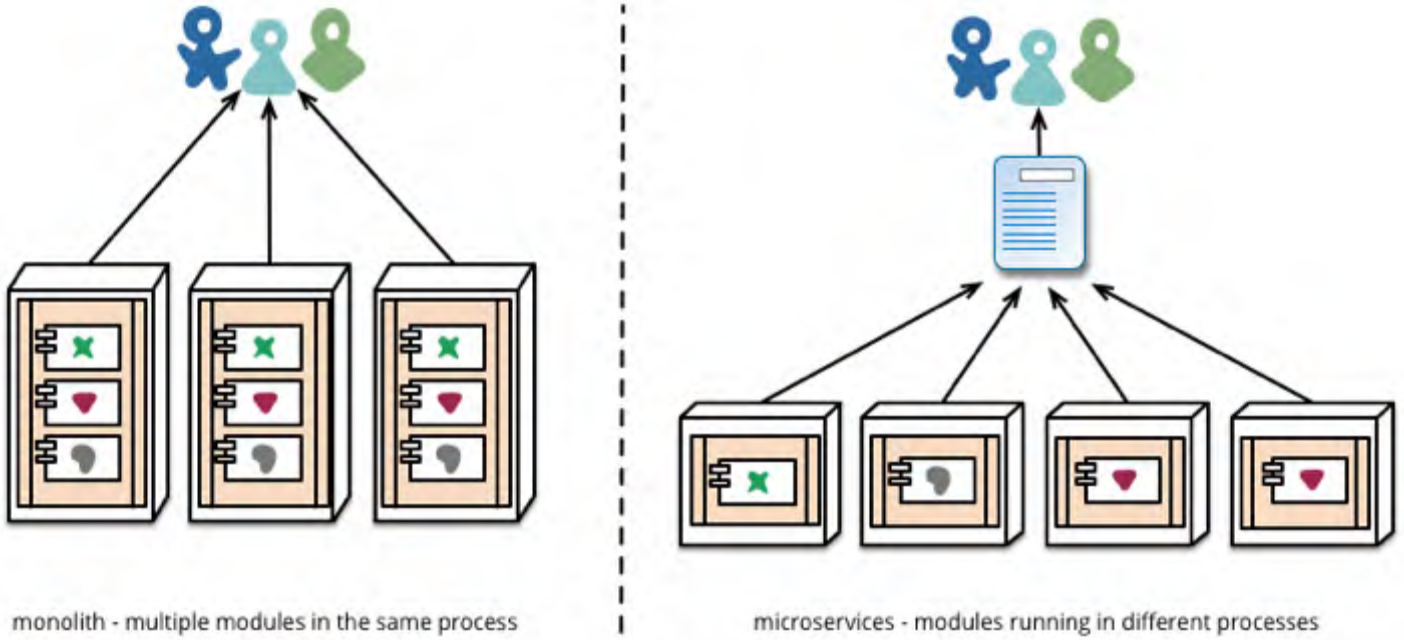
Cross-functional teams...

... organised around capabilities
Because Conway's Law

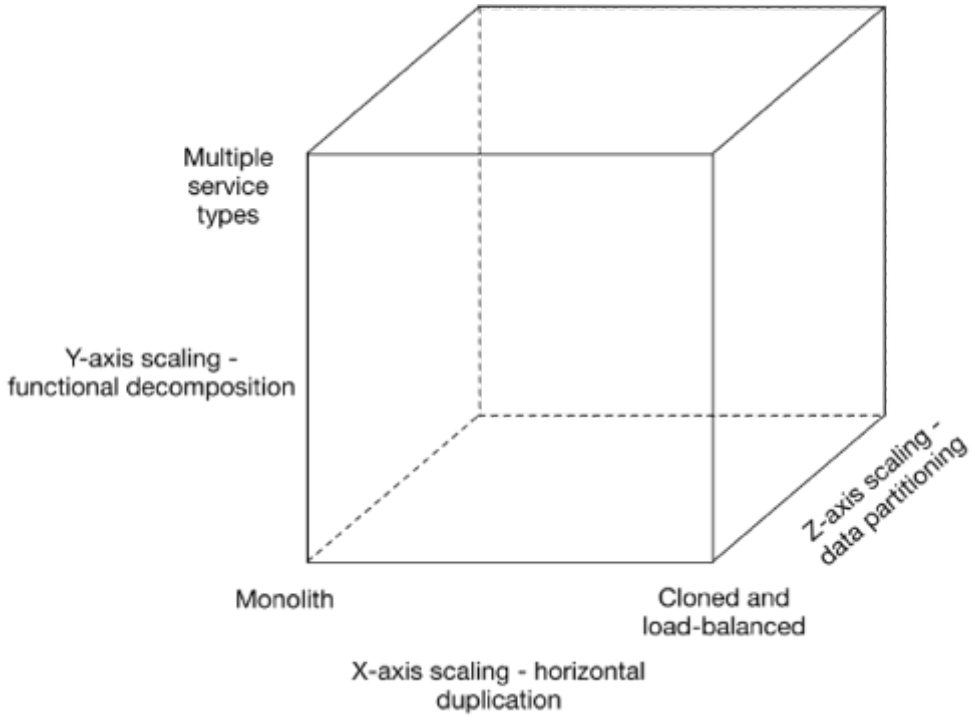
# 传统巨石架构 VS 微服务架构(数据库)



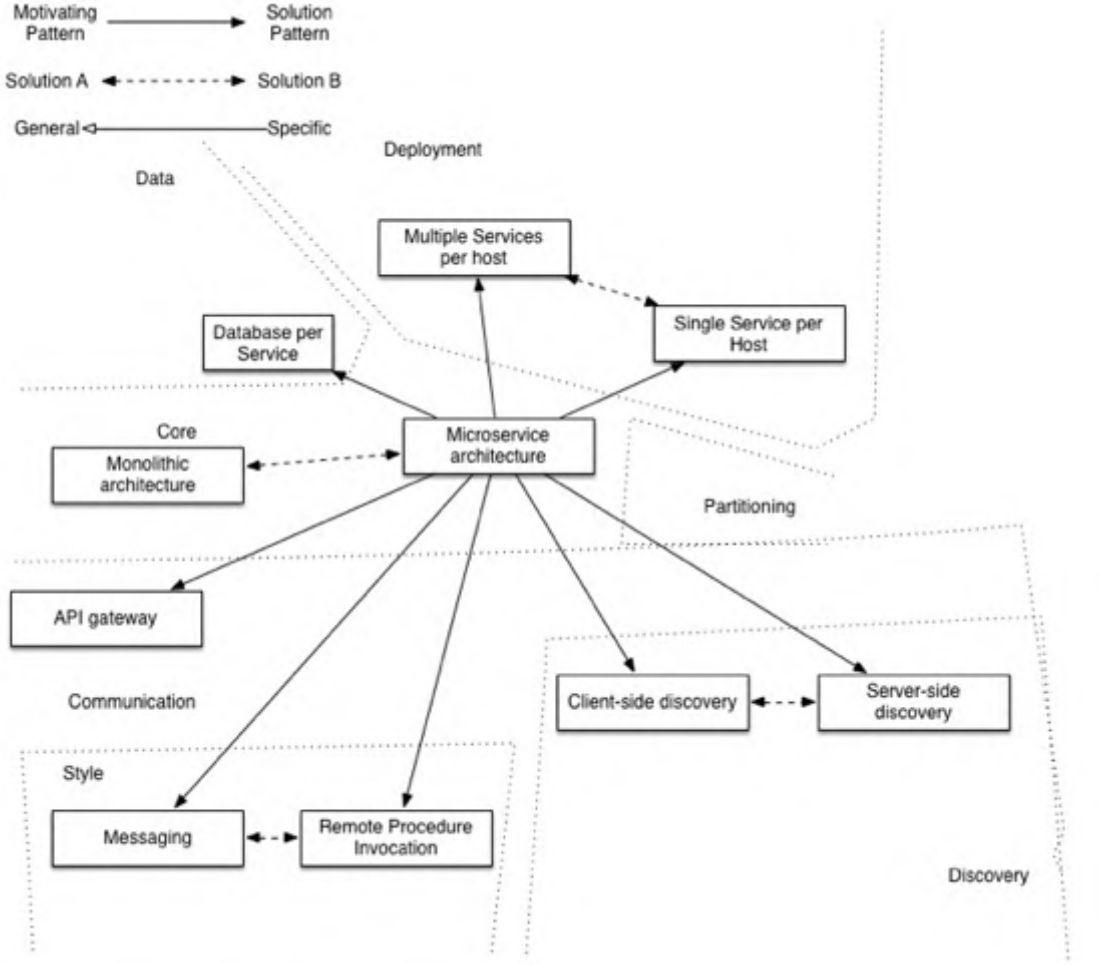
# 传统巨石架构 VS 微服务架构(进程)



# 微服务架构的三维CUBE模型

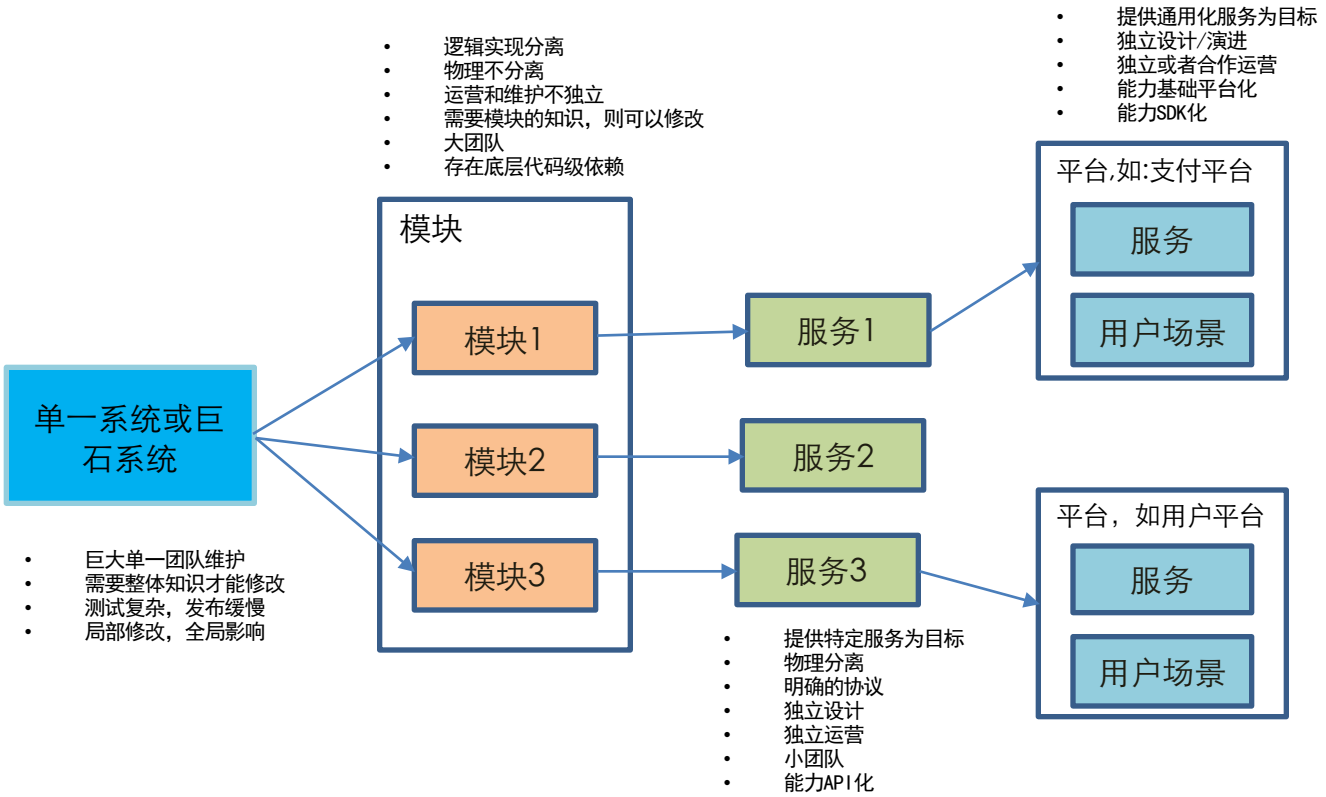


# 微服务架构的三维CUBE模型




- 数据
- 部署
- 分区
- 自动发现
- 服务间通信
- API网关

# 架构的转换路径图


单一系统或巨石系统
巨大单一团队维护
需要整体知识才能修改
测试复杂，发布缓慢
局部修改，全局影响
模块
逻辑实现分离
物理不分离
运营和维护不独立
需要模块的知识，则可以修改
大团队
存在底层代码级依赖
模块1
模块2
模块3
服务1
服务2
服务3
提供特定服务为目标
物理分离
明确的协议
独立设计
独立运营
小团队
能力API化
平台,如:支付平台
服务
用户场景
提供通用化服务为目标
独立设计/演进
独立或者合作运营
能力基础平台化
能力SDK化
平台，如用户平台
服务
用户场景

# 互联网服务架构之核心经验谈

**9个技术手段**

SET模型
全网调度
灰度升级
过载保护
立体监控
自动部署
柔性可用
数据银行
云中生长

**4个意识**

大系统做小
先抗住再优化
边重构边生活
干干净净

**2个技术价值观**

有损服务
动态运营

不仅是技术的挑战，更是方法论的挑战

# 设计可扩展的互联网化服务(James Hamilton)

- 总体服务设计 ---18条最佳实践
- 自动化配置和管理设计 ---11条最佳实践
- 依赖管理 ---6条最佳实践
- 发布周期和测试 ---12条最佳实践
- 硬件选型和标准化 ---4条最佳实践
- 运维和容量规划 ---5条最佳实践
- 审计监控和报警 ---9条最佳实践
- 优雅降级和准入控制 ---3条最佳实践
- 客户和媒体沟通计划
- 客户自配置和自助(customer self provisioning and self help)

https://www.usenix.org/legacy/event/lisa07/tech/full_papers/hamilton/hamilton_html/

# 微服务架构设计原则(12 factor)

- CodeBase. One codebase tracked in revision control, many deploys
- Dependencies. Explicitly declare and isolate dependencies
- Config. Store config in environment
- Backing services. Treat Backing services as attatched Resources
- Build,Release,Run. Strictly separate Build and Run Stages
- Processes. Execute the app as one or more stateless processes

# 微服务架构设计原则(12 factor)

- Port Binding. Export Services via port binding
- Concurrency. Scale out Via process model
- Disposability. Maximize robustness with fast startup and graceful shutdown
- Dev/Prod Parity. Keep Dev/Test/Prod as similar as possible
- Logs. Treat Logs as event streams
- Admin Processes. Run admin/management tasks as one-off processes

# DevOps之组织与文化实践

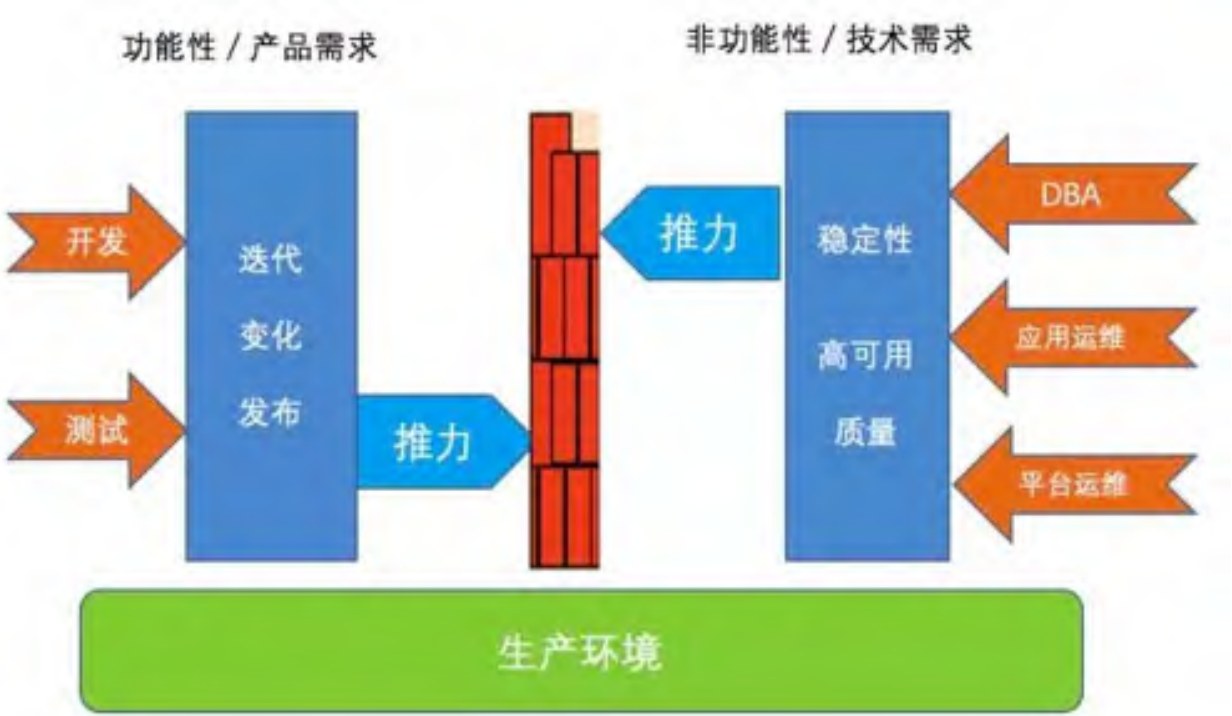
# 组织的三种类型

**Table 1** How organisations process information

| Pathological | Bureaucratic | Generative |
|---|---|---|
| **Power oriented** | **Rule oriented** | **Performance oriented** |
| Low cooperation | Modest cooperation | High cooperation |
| Messengers shot | Messengers neglected | Messengers trained |
| Responsibilities shirked | Narrow responsibilities | Risks are shared |
| Bridging discouraged | Bridging tolerated | Bridging encouraged |
| Failure→ scapegoating | Failure→ justice | Failure→ inquiry |
| Novelty crushed | Novelty→ problems | Novelty implemented |

A Westrum的A typology of organisational cultures

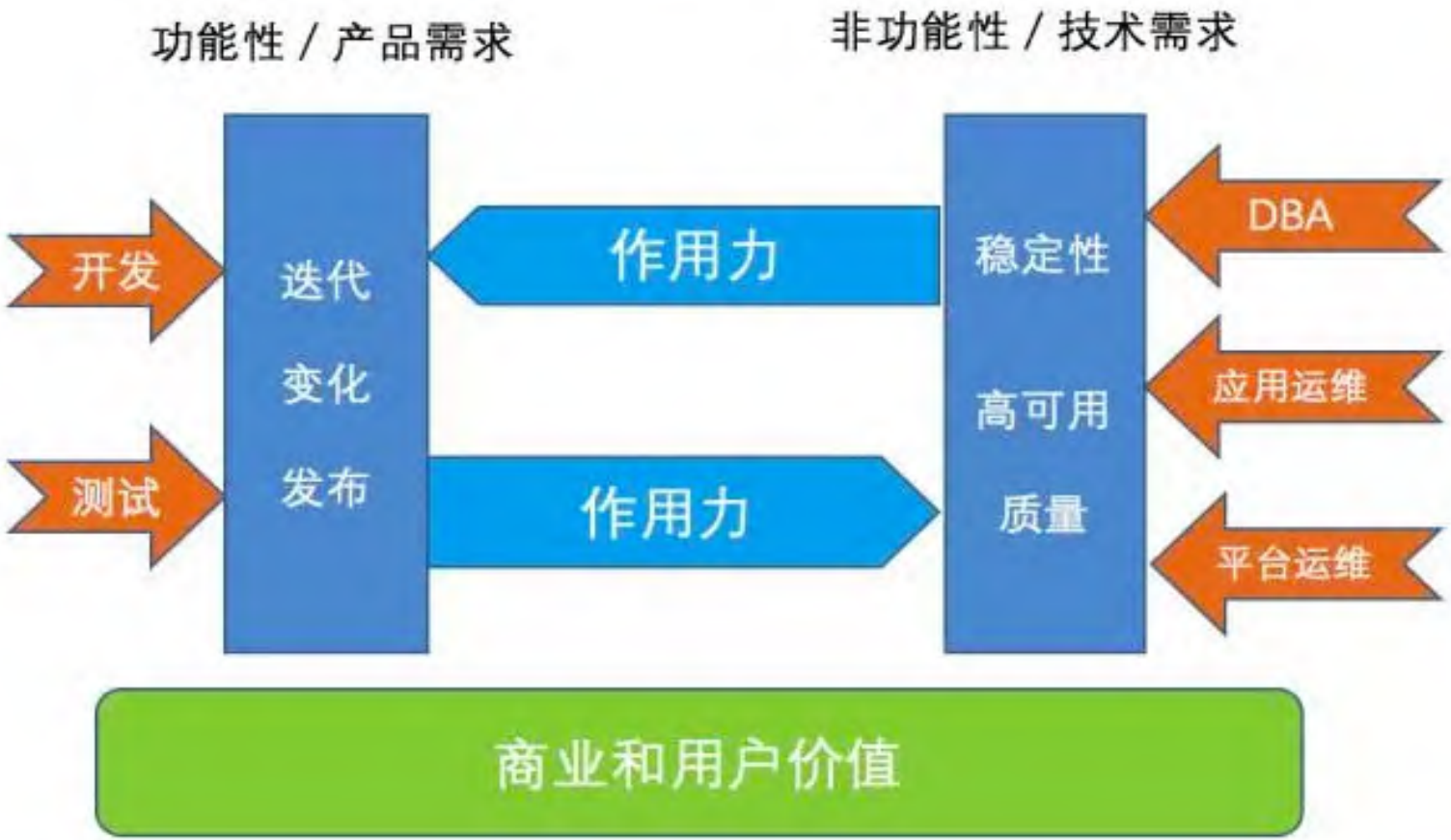
# Dev与Ops的冲突性



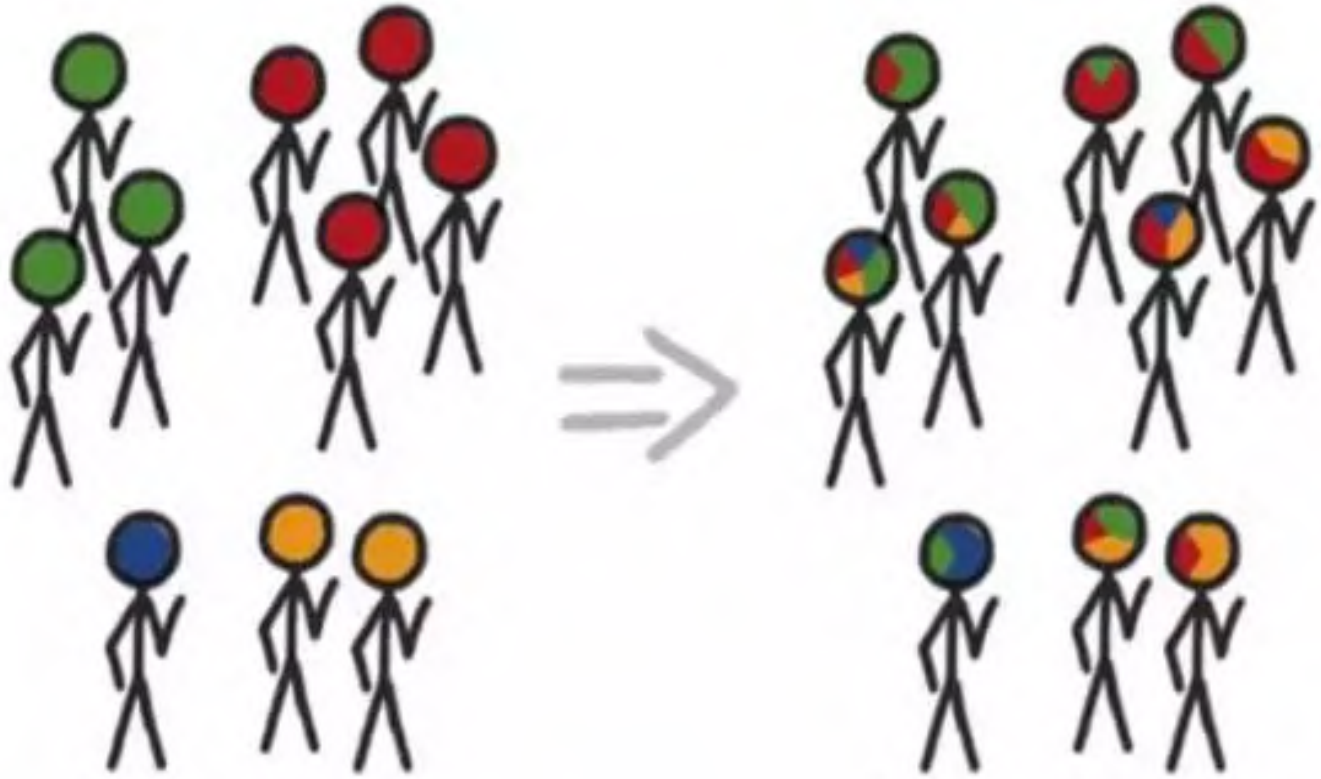

DevOps不只是开发与运维的问题

DevOps也是IT部门和业务部门之间的问题

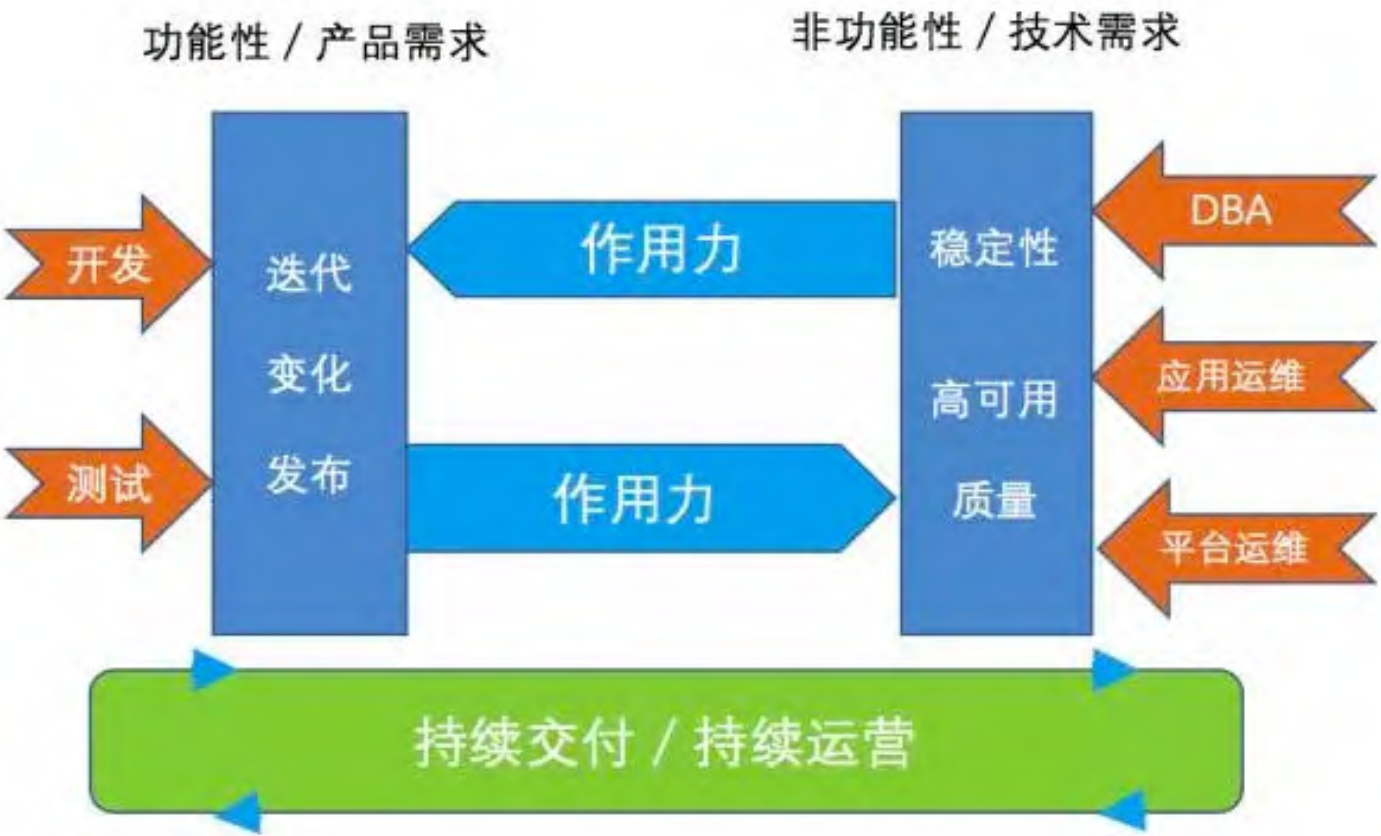
# Dev与Ops的冲突解决方案（业务）



# Dev与Ops的冲突解决方案（思维）



# Dev与Ops的冲突解决方案（技术）

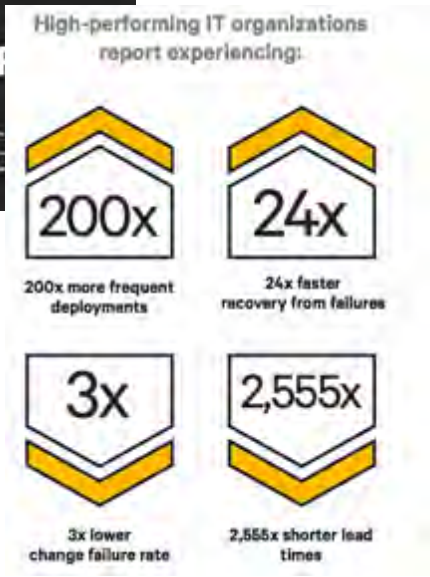


# 建立面向IT性能的核心度量体系

# IT性能有多重要


| | 2015 (Super High vs. Low) | 2014 (High vs. Low) |
|---|---|---|
| Deployment Frequency | 30x | 30x |
| Deployment Lead Time | 200x | 200x |
| Mean Time to Recover (MTTR) | 168x | 48x |
| Change Success Rate | 60x | 3x |






Firms with highperforming were twice as likely to exceed their profitability, market share and productivity goals.”

# IT性能组织的指标设定

01 lead time for changes
变更时长

02 release frequency
发布频率

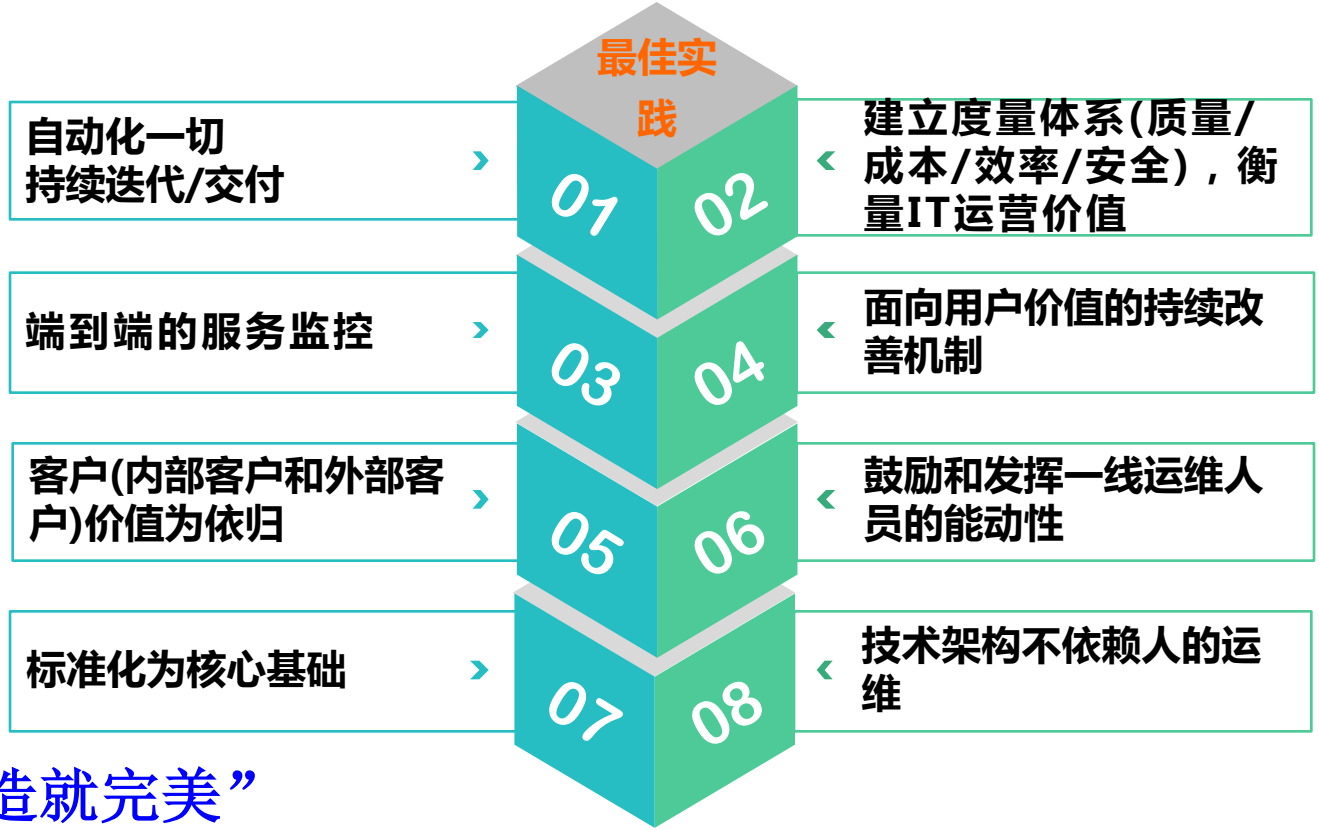
03 time to restore service
服务恢复时长

04 change fail rate
变更失败率

# DevOps之度量体系

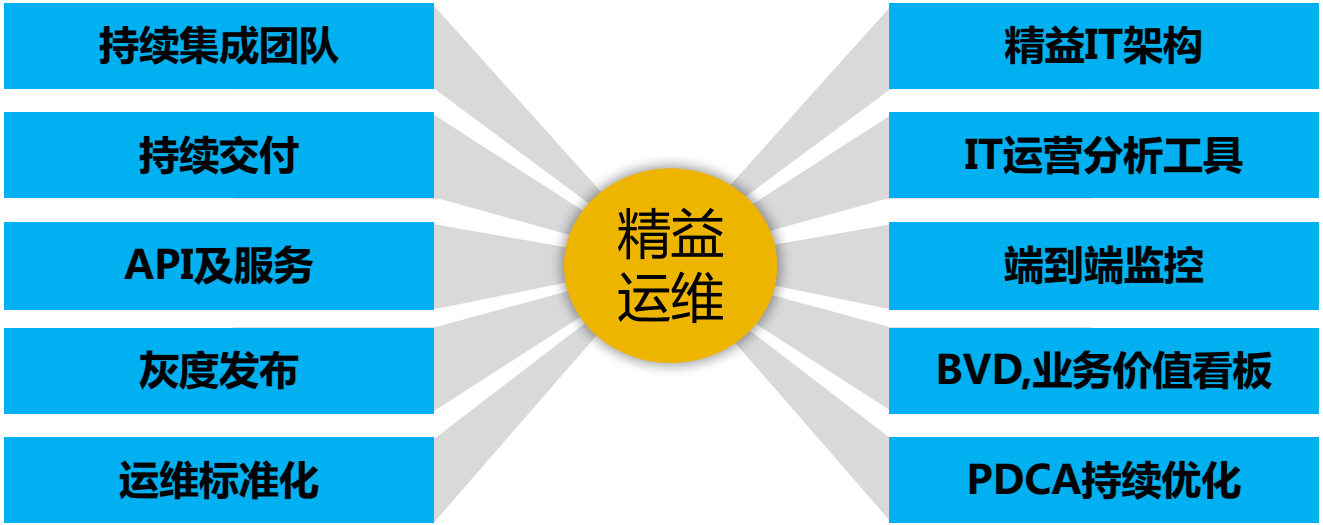
- （全局/核心）Cycle time/Lead Time。一个特性的完整交付周期
- （全局/核心）交付频率
- （全局/核心）服务恢复时长
- （全局/核心）变更失败率
- （局部/辅助）自动化测试覆盖率
- （局部/辅助）代码库特征
- （局部/辅助）缺陷数量
- （局部/辅助）交付速度
- （局部/辅助）构建情况：成功、失败、时间
- （局部/辅助）提交次数

# DevOps运维之八大实践




“实践造就完美”

# DevOps运维之十大工具




工具推动变革

# DevOps运维到底是什么？


**拒绝浪费 / 创造价值**

# 推荐阅读


BEST PRACTICE
itSMF
IT服务管理


持续交付
发布可靠软件的系统方法


A3思维
丰田PDCA
管理系统的关键要素


精益
企业
高效能组织如何规模化创新
O'REILLY


Google
软件测试之道


持续集成
软件质量改进和风险降低之道
Continuous Integration


精益思想

# 关于优维

优维科技（深圳）有限公司主要聚焦互联网公司以及需要“互联网+”转型的公司提供一站式运维服务，通过交付运维价值经验平台，帮助企业的IT能力互联网化。优维人将带着“DevOps管理专家”的使命，提供全栈的互联网化运维能力！

- 互联网运维体系构建
- 互联网运维优化和技术架构优化服务
- 互联网全栈运维平台（自动化+数据化+安全）
- 互联网DevOps相关服务咨询，如持续交付、微服务、运维

EASYOPS
优维

http://www.easyops.cn

# 关于团队

优维的核心团队是由腾讯、阿里、网易的运维专家组成，具有10年的海量运维和服务运维经验，丰富的运维开发经验。

- 优维科技CEO，中国开放运维联盟发起人，"精益运维"理论提出者
- 中国第一批DevOps Master授权讲师，持续交付专家
- 优维科技CTO黎明，中国最大的运维管理平台织云设计者
- 公司85%是研发，坚持技术与产品驱动，最佳实践落地
- 坚持IT管理过程ERP产品化，提升所有企业的IT效能，驱动业务创新
- 坚持DevOps理念，带着"重新定义运维"的使命，做中国最好的IT运维服务厂商

EASYOPS
优维

http://www.easyops.cn

# 谢谢


优维，互联网运维践行者
http://www.easyops.cn

深圳.广州

QQ：29956914
微信：waynewang
电话：18682215876