# 七牛持续交付平台SPOCK

李倩

- 七牛的场景与痛点
- SPOCK的业务架构
- SPOCK的技术架构
- 最佳实践与思考

# 七牛的场景

简单·可信赖

**海量的产品与代码**
6大核心产品线
500+组件和微服务
产品+内部平台+实施项目

**复杂的编译和运行环境**
Linux+Mac+Windows
后端+前端+移动端
语言版本 基础库版本
基础组件版本

**质量难控制**
高效率与高质量并重
5000+的自动化测试用例
多种场景的测试

**发布要求高**
平均每天要有6次发布
需要在极短时间内发布
按时发布 按需发布

**协作难度大**
100+研发同时写代码
代码分支多 合并频度高
issue状态难跟踪

## 为解决这些问题而产生的工具链

简单·可信赖

- 代码库管理：github + gitlab
- CI/CD工具：jenkins、travis
- 自动化测试框架：qtest（ginkgo+gomega）+ appium
- 流程管理/协作工具：jira + confluence

松散的工具 ≠ 效率的提升

软件工程的生命周期从未被完整地信息化

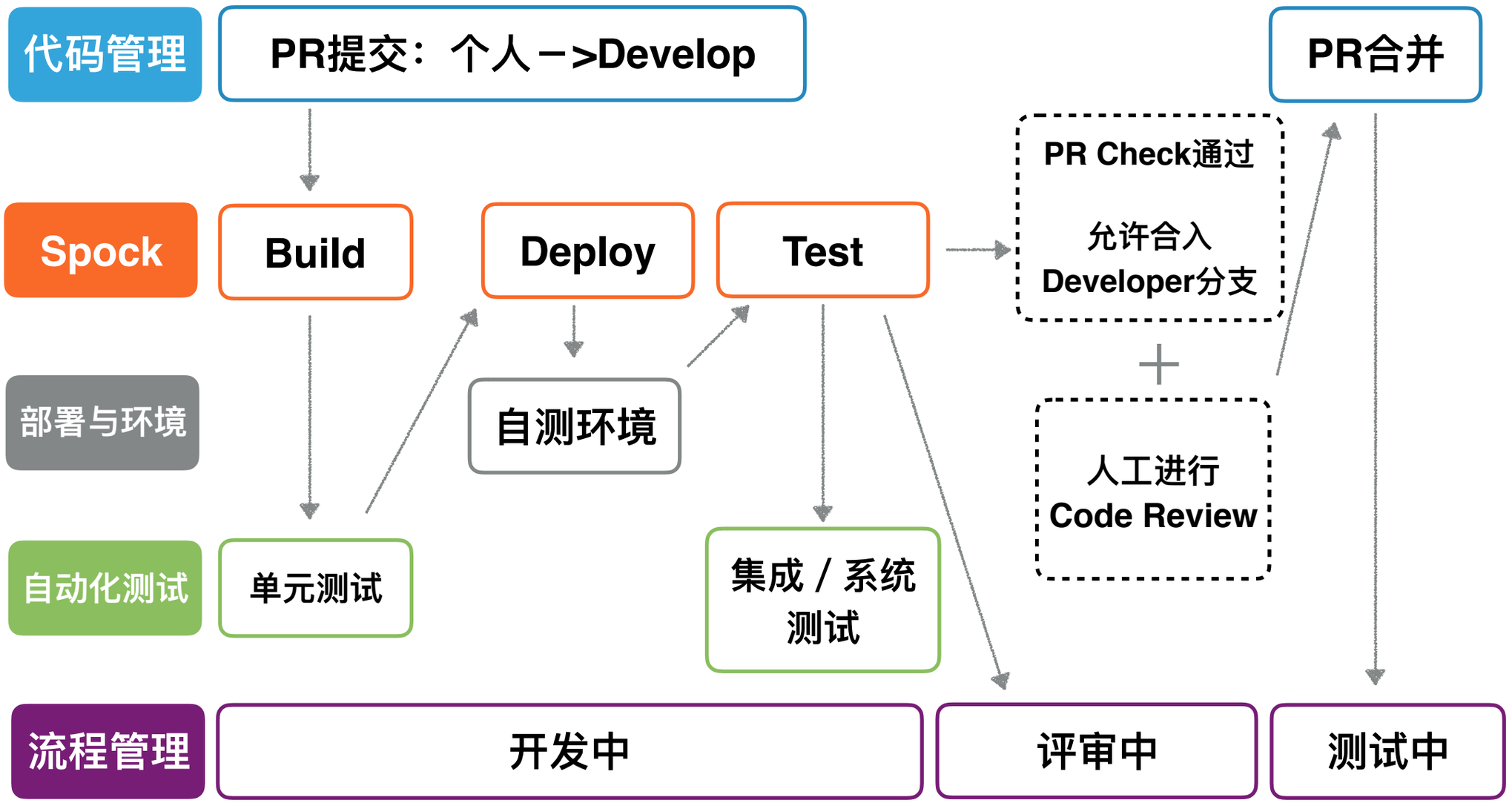
## SPOCK的PIPELINES

简单·可信赖

| Spock Pipelines | 角色 | 触发 | 输出 |
| --- | --- | --- | --- |
| 自测 | 开发人员 | 从个人分支<br>提交PR<br>至Developer分支 | 这段代码没问题<br>允许进入测试流程 |
| 集成 | 测试人员 | 从Developer分支<br>提交PR<br>至Master分支 | 这些代码没问题<br>允许进入发布流程 |
| 发布 | 发布人员 | 人工/自动触发 | 模拟生产环境测试没问题<br>提供可发布的包/或发布生产 |

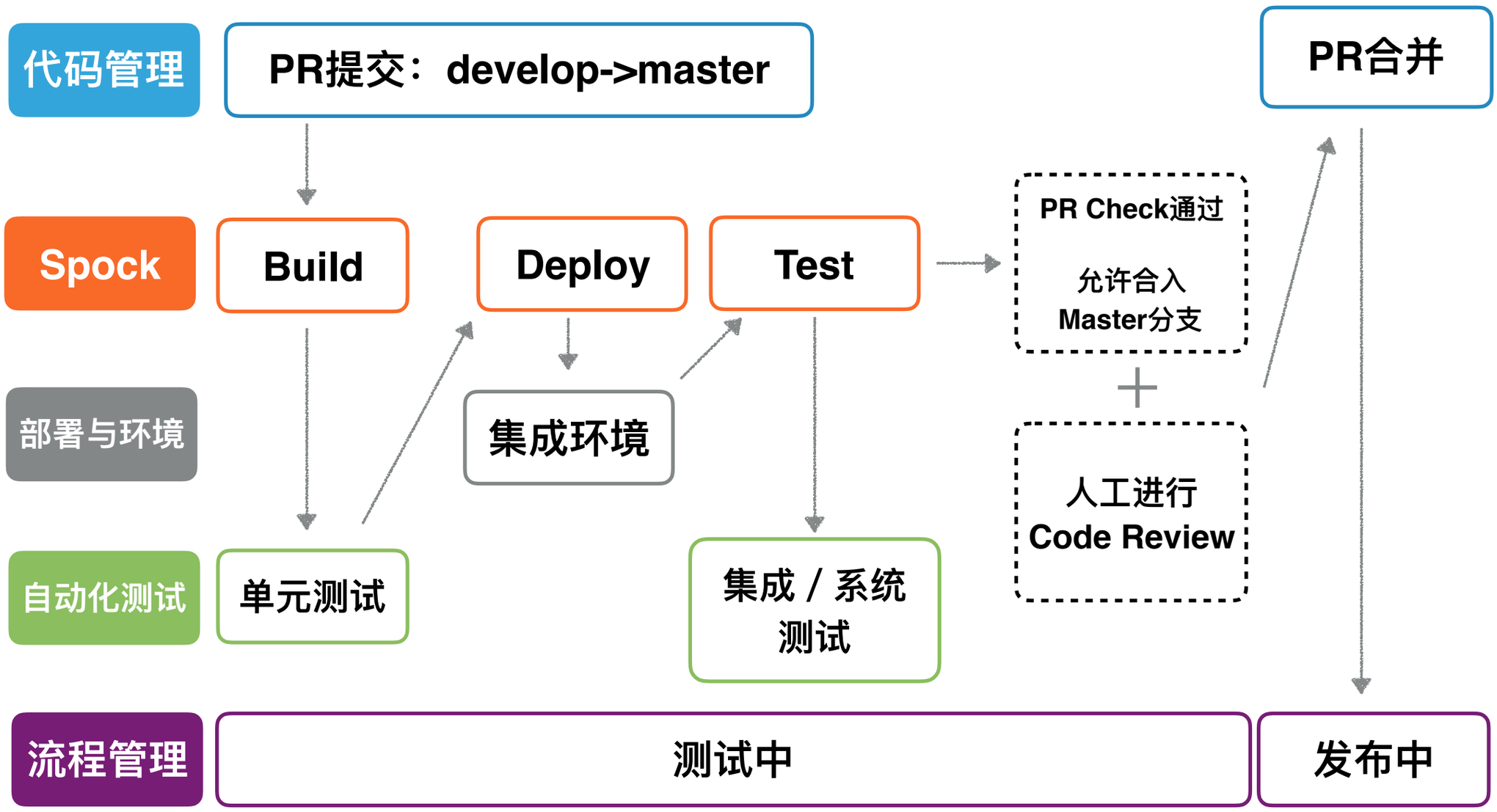
# 互动的迭代过程 – 自测PIPELINE

简单·可信赖



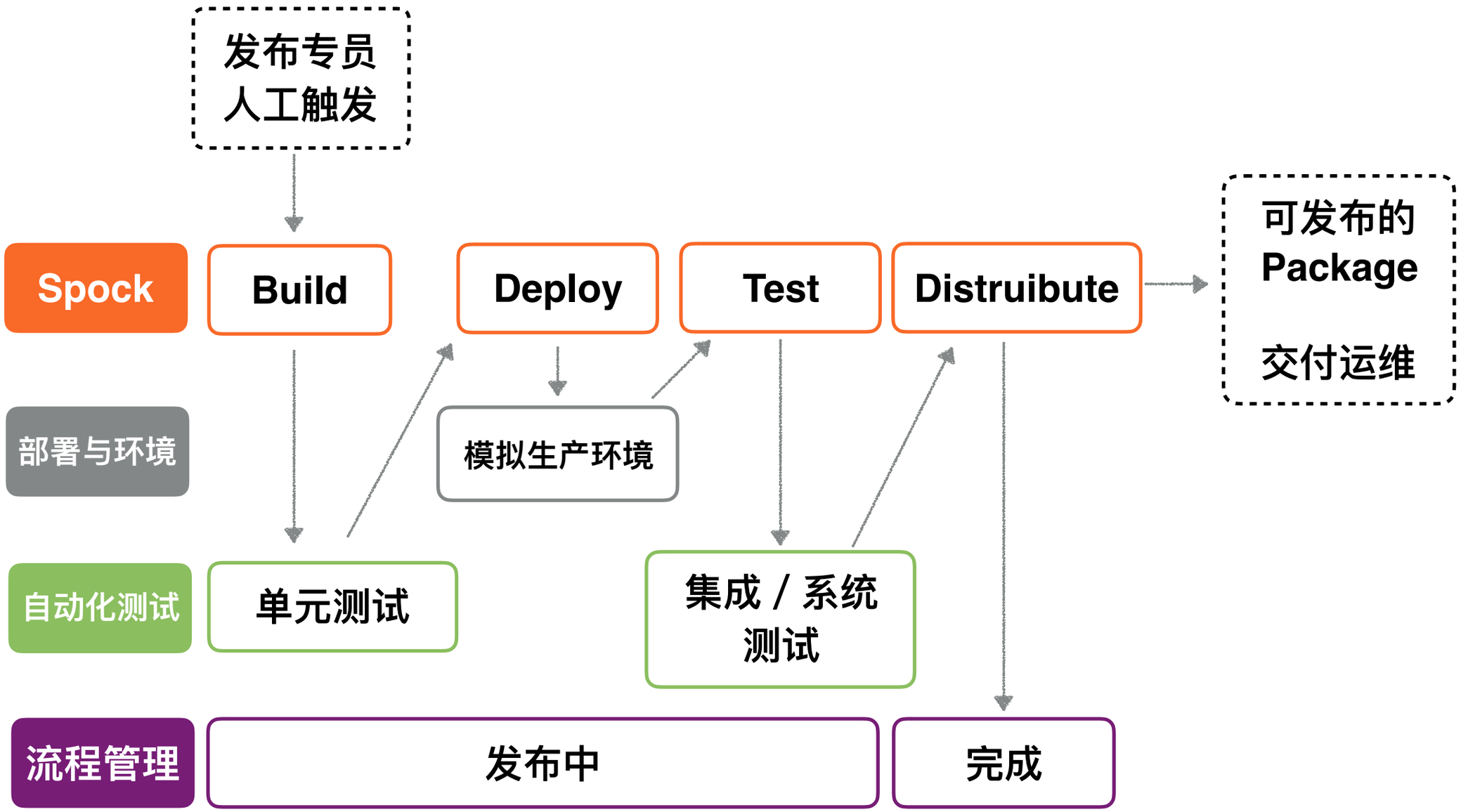
# 互动的迭代过程 – 集成PIPELINE

简单·可信赖



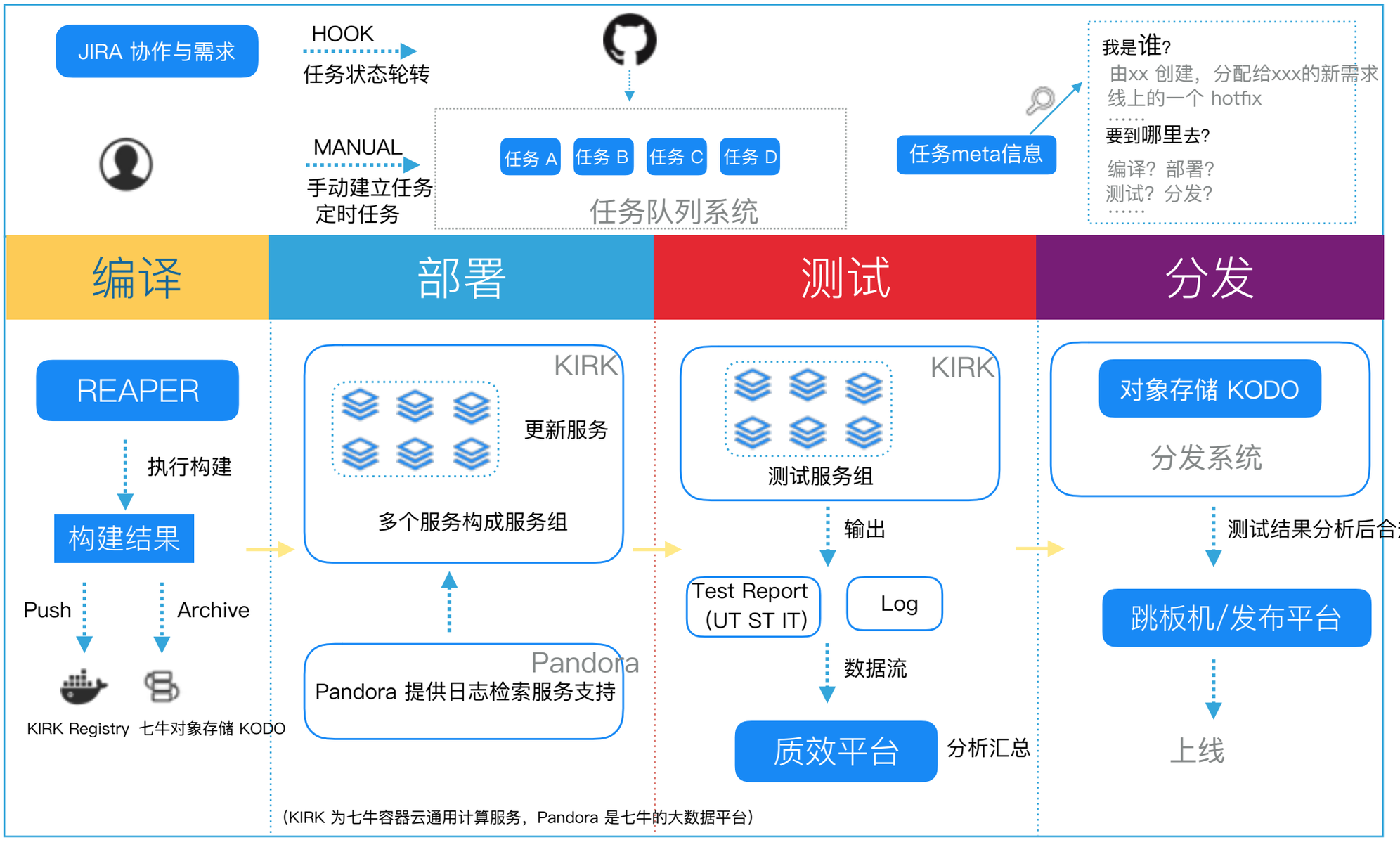
## 互动的迭代过程 – 生产PIPELINE

简单·可信赖



# SPOCK技术架构概览

简单·可信赖



# 自动化与信息化一切

甚至包括流程管理与代码管理的自动化

自动化的关键是体系完整

自动化的前提是稳定性与测试用例覆盖

自动化是快速发布的保障

在QA资源短缺时自动化更有意义

# 自测环境与Docker

自测时采用集成环境，才能发现集成问题

在微服务大环境下，单侧远远不如集成测试有意义

单测帮助迅速验证，减少打回率，在早期就过滤掉大部分问题

容器化部署，一人一套独立的测试环境，一键起环境，开箱即用

# 微服务化的价值

将需求化整为零，分工明确

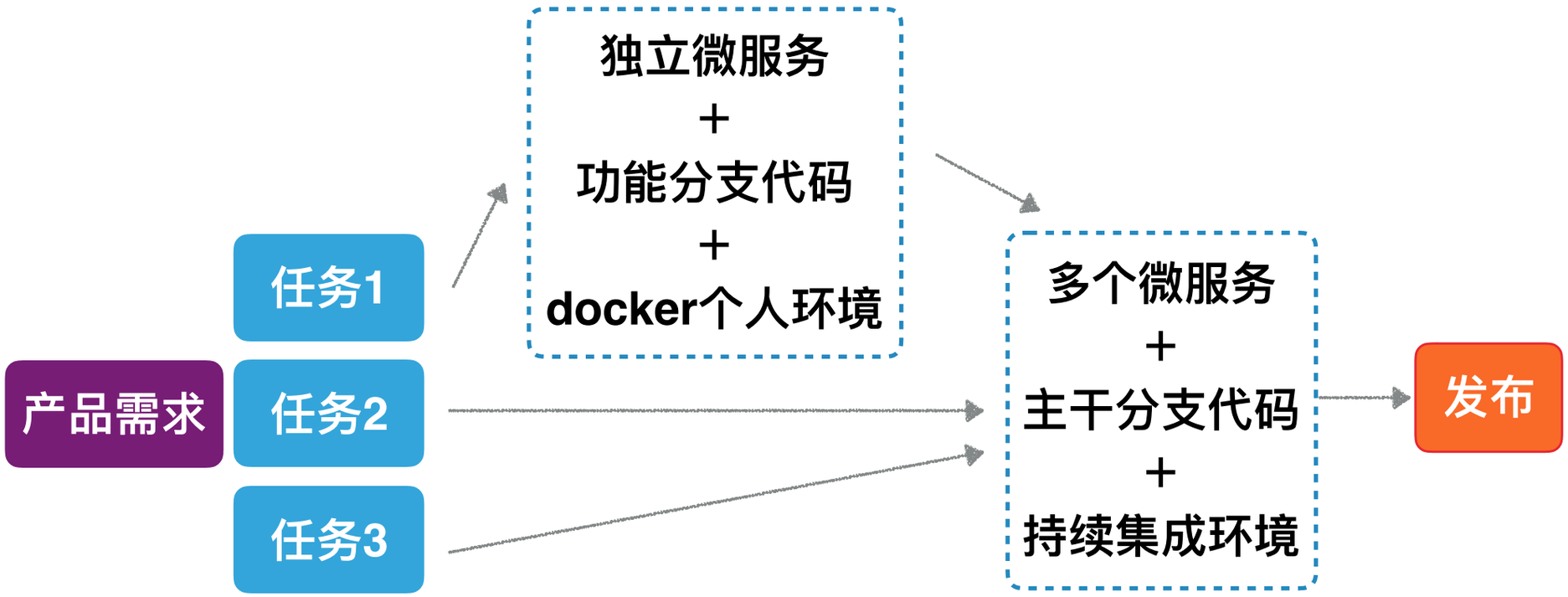
将代码和服务化零为整，持续集成并发布

微服务间相互隔离，独立部署，运维和排查问题

微服务间通过接口沟通，易于管理

# 最佳实践 - 微服务化与挑战

简单·可信赖




通过Pipeline加强管控能力

化整为零 → 化零为整

通过自动化提升效率

# 应对微服务化的挑战 = 容器 + 配置中心

容器的优势：弹性伸缩 一个镜像 多处部署

容器镜像+配置可以在任意环境部署出你想要的环境

配置还需要描述微服务依赖，以及启动顺序

使用模板简化配置的复杂度

# 发布透明化

通过Spock平台获知每次发布中包含的内容（jira）

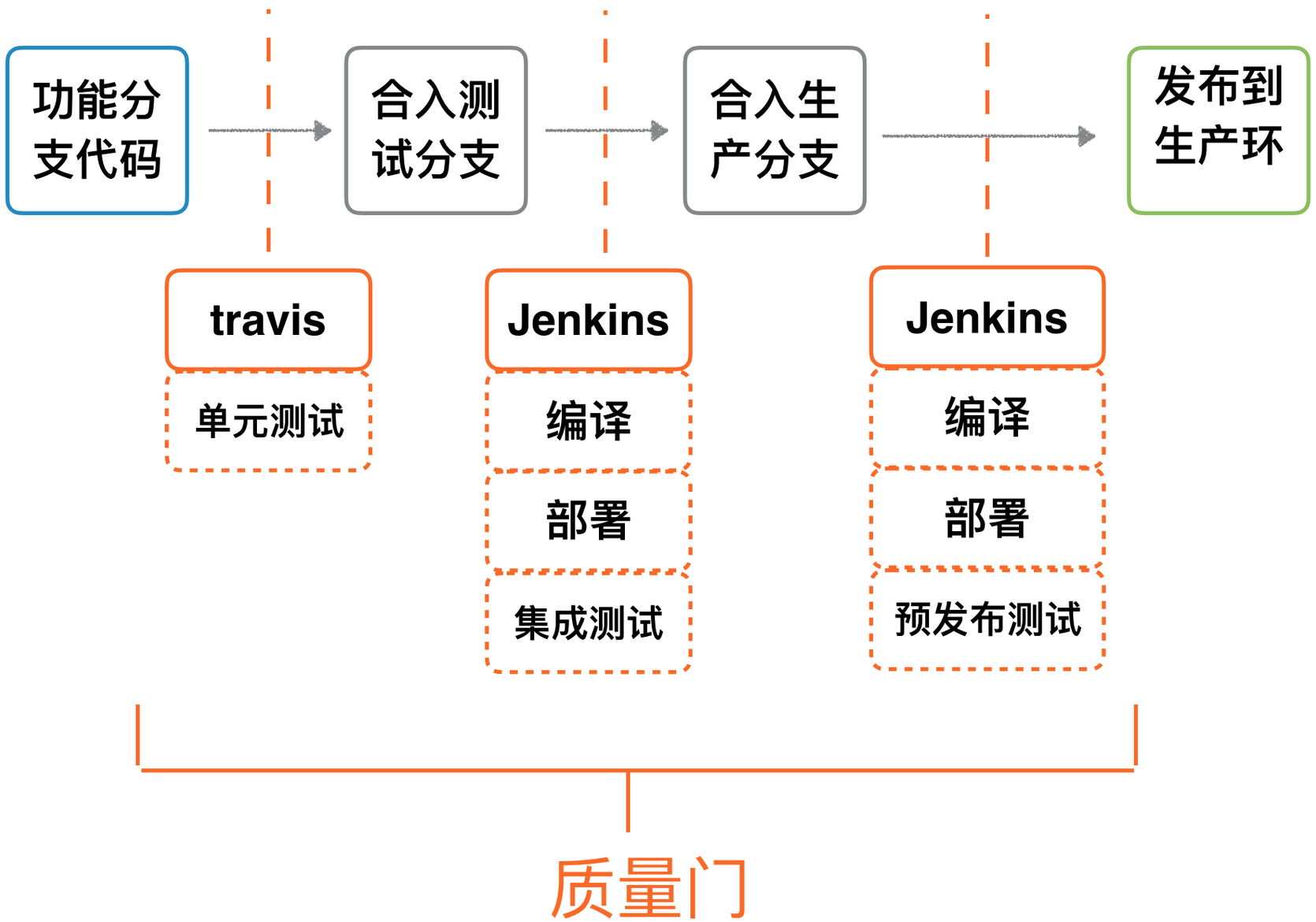
发布内容透明化，方便回滚和迅速排查问题

通过Spock平台获知当前发布的状态

发布过程透明化，方便了解进度

# 最佳实践 - 质量门

简单 · 可信赖



# 质量是持续集成驱动开发的杠杆

忽略质量，单纯强调持续集成的稳定性和效率是没有意义的

质量门就是质量保障的体系化建设

质量数据可以反过来作用于开发过程，构成回环

云计算行业更需要重视质量

# 七牛－质效看板

NE | 质效看板 | 单元测试 | 系统测试 | GO REPORT | 事故统计 | 缺陷分析 | 性能测试 | 李倩

截止上周 | 截止上月 | 截止上季度

| 产品线 | 单测覆盖率 | 代码合规 | 测试通过率 | 返工率 | 事故提交报告时长 | 事故分数 (本季度) | 外部缺陷 (本季度) |
|---|---|---|---|---|---|---|---|
| H D ) | 60.0 0.10 | 7 .35 | 7 .28 -0.16 | 4.44 0.25 | 2 ).73 | 3 ) | 6 |
| r a ra | 3.9( -0.25 | 8 .58 -0.03 | 6( .86 | .24 0.10 | 1 ).67 -0.92 | 3 ) | 4 |
| D | .7.7 9.25 | 76.15 0.39 | 9 .90 | .05 -0.05 | 1 ).53 -0.75 | 1 | 7 |
| N | 4 .2 7.52 | 78 .3 | 8 .49 -0.03 | .08 0.02 | 1 .28 | 3 | 2 |
| _A | 4 2 3.04 | 75. 6 0.01 | 7( 28 -0.19 | 1.25 -0.04 | 1 ).09 | 3 | 25 |
| ( | 41.9 7.06 | 74.71 0.17 | 97 55 0.32 | 2.75 -0.20 | 3 .76 | 2 | 9 |

# 质量驱动开发的关键解读

收集多方维度数据：开发，测试，运维，服务质量

使用大数据分析找到关键性指标，确定KPI

KPI可以直接刺激和推动某个角色

数据需要清洗，数据需要长线收集

## 成果

简单·可信赖

**质量**
(Quality)

- 核心产品单测覆盖率 60%+
- 代码合规 80%+
- PIPELINE通过率 85%+
- 集测覆盖率 E2E + API 35%+

**效率**
(Efficiency)

- 构建效率 2-10分钟
- 缺陷解决率 82.97%
- 发布频度 每周50+
- 核心产品MTTR 1.5小时

**Table 2: 2017 IT performance by cluster**

| Survey questions | High IT performers | Medium IT performers | Low IT performers |
|---|---|---|---|
| **部署频率**<br>针对主要应用和服务，你的组织多久部署一次代码? | On demand (multiple deploys per day) | Between once per week and once per month | Between once per week and once per month* |
| **变更前置时间**<br>针对主要应用和服务，变更的前置时间是多长（从代码提交到代码在生产环境中运行成功的时间）? | Less than one hour | Between one week and one month | Between one week and one month* |
| **故障恢复时间（MTTR）**<br>针对主要应用和服务，如果发生服务故障，一般多久能恢复服务（比如：计划外宕机，服务损害）? | Less than one hour | Less than one day | Between one day and one week |
| **变更失败率**<br>针对主要应用和服务，变更结果是回滚或随后修复的比例是多少（比如：导致服务损害，服务中断，需要紧急修复，回滚，向前修复，补丁）? | 0-15% | 0-15% | 31-45% |


谢谢!