大连理工大学
Dalian University Of Technology

系统持续交付设计

大连理工大学　　软件学院

马瑞新

目录

1. 系统部署服务
2. 系统发布服务
3. 软件构件
4. Maven工具依赖管理
5. 华为云（Devcloud）发布管理服务
6. 持续交付流水线

系统部署服务

自动化部署介绍

- 部署方案的发展
- 云端DevOps平台
- 蓝绿部署
- 灰度发布

手工部署到自动部署

部署方案比较：（以部署60台机器为例）

手工部署
60台x(安装2.5小时+手工配置0.5小时)/台
= 180小时

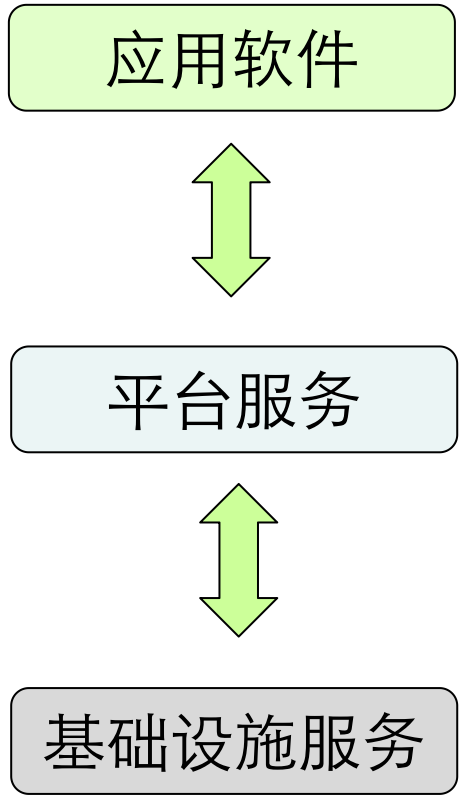
需要人工干预的半自动部署
磁盘复制0.5小时+ 60台x(手工配置0.5小时)/台
= 35小时

免接触的自动部署
自动部署1小时（同时自动配置）<
1小时

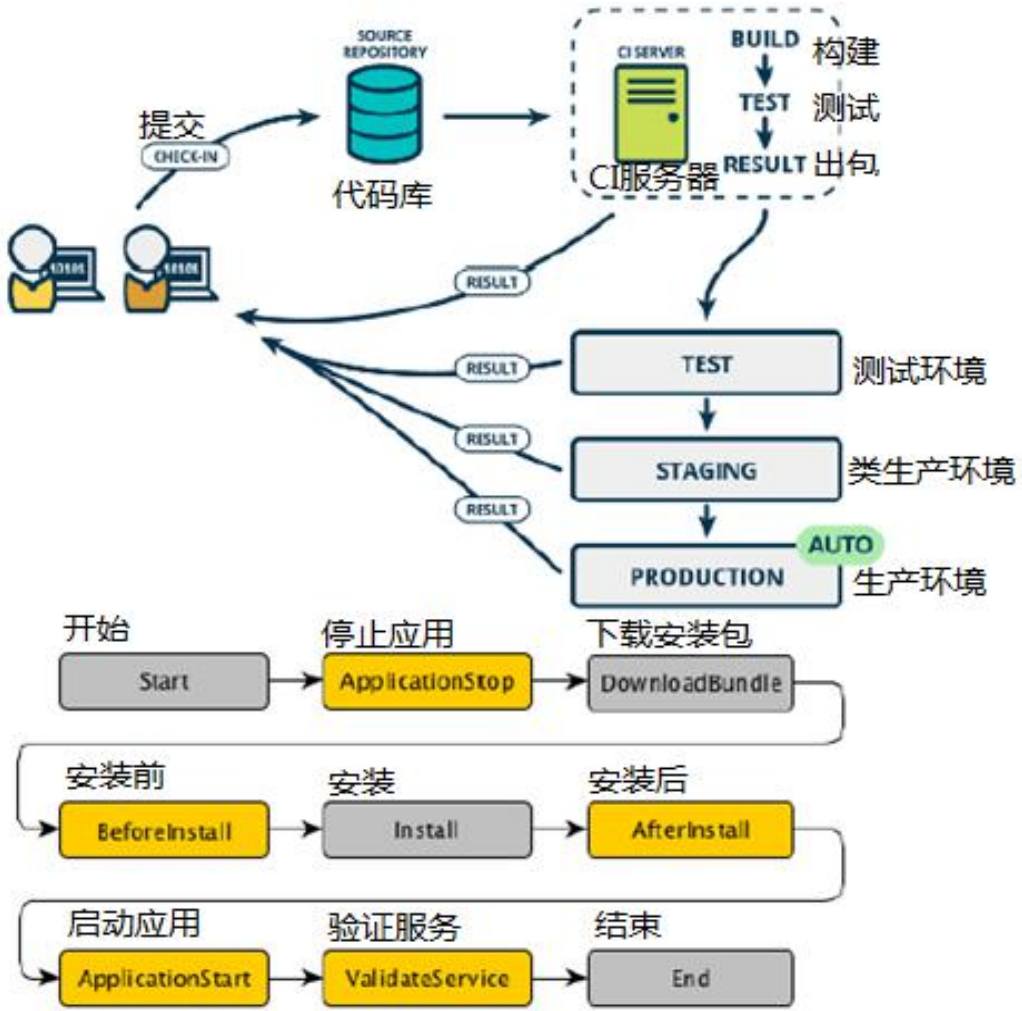
保证了部署的质量，实现了标准化，就降低了系统维护的工作量。

云环境中的部署 – 自动安装、配置操作系统、基础软件与应用软件

- 应用软件
  - 基础软件安装
  - 中间件安装和配置
  - 应用软件包安装和配置
- 基础组件
  - 操作系统安装
  - 网卡配置和安装
  - 硬盘和内存动态调整



应用的部署管理



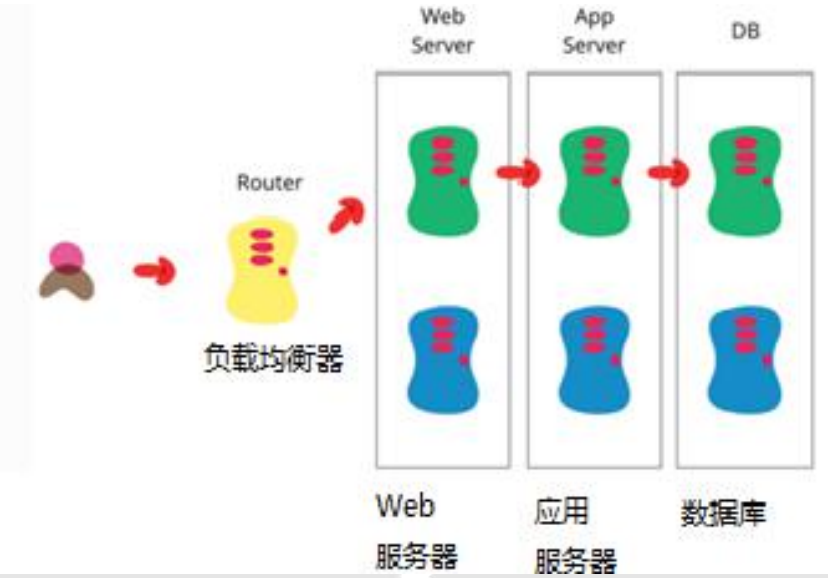
- 发布是针对业务的，部署是针对环境的。
- 部署可以频繁、安全、可持续的进行。
- 部署的版本来自于构建库。
- 使用相同的脚步、相同的部署方式对所有环境进行部署，确保一致性。
- 为了确保安全性和可用性、部署可以采用蓝绿部署、灰度部署等能力。

持续交付框架


内建质量、持续改进、变更看板
阶段
环境
动作
角色
Dev
Test
Staging
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

蓝绿部署

- 一种以可预测的方式发布应用的技术，目的是减少发布过程中服务停止的时间
- 需要准备两个相同的环境（基础架构）
  - 平时在蓝、绿环境通过负载均衡运行业务
  - 升级前把负载切到绿环境
  - 升级时在蓝环境中部署新版本，并进行测试。测试完成，把负载切回蓝环境，然后升级绿环境
  - 绿化升级测试完成后，系统恢复蓝绿环境共同承担负载
- 问题
  - 考虑数据库与应用部署同步迁移/回滚的问题
  - 需要妥当处理未完成的业务和新的业务



灰度发布/金丝雀发布

- 灰度发布是在原有版本可用的情况下，同时部署一个新版本/金丝雀，测试新版本的性能和表现，以保障整体系统稳定的情况下，尽早发现、调整问题。
- 步骤
  - 准备好工件，包括：构建工件，测试脚本，配置文件和部署清单文件
  - 从负载均衡列表中移除掉“金丝雀”服务器
  - 升级“金丝雀”应用（排掉原有流量并进行部署）
  - 对应用进行自动化测试
  - 将“金丝雀”服务器重新添加到负载均衡列表中（连通性和健康检查）
  - 如果“金丝雀”在线使用测试成功，升级剩余的其他服务器（否则就回滚）

开源部署工具

主流配置管理工具

工具	语言	架构	协议
Puppet	Ruby	C/S	HTTP
Chef	Ruby	C/S	HTTP
Ansible	Python	无Client	SSH
Saltstack	Python	C/S（可无Client）	SSH/ZMQ/RAET

主要选型因素

- 是否需要每台机器部署agent（客户端）
- 大规模并发的能力
- 二次开发扩展的能力
- 开源社区的对接
- 学习的门槛
- 第三方工具的丰富程度
- 现有用户使用的规模

开源部署工具 - Ansible （1/2）

简单

- 类自然语言的自动化脚本
- 不需要编程技巧
- 按顺序执行任务
- 快速获得生产力

强大

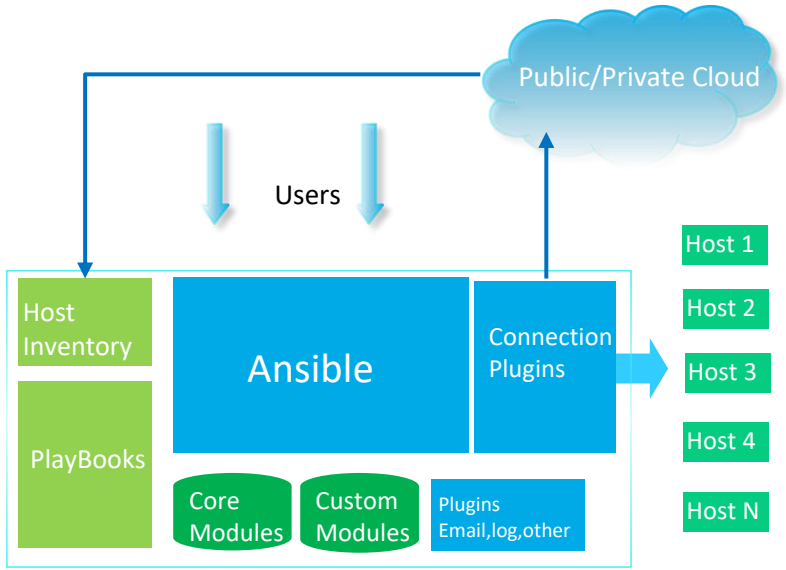
- 应用部署
- 配置管理
- 工作流编排
- 应用生命周期编排

无需代理

- 架构上无需代理
- 使用OpenSSH和WinRM
- 无需开发代理和升级代理
- 更加安全有效

开源部署工具 - Ansible（2/2）

Ansible 核心架构



- Ansible：核心模块
- Connection plugins：Ansible基于连接插件连接到各个主机上，ansible默认使用ssh连接到各个主机，但是也支持其它连接方式
- Host inventory：定义可管控的主机列表
- Playbooks：按照所设定编排的顺序执行完成安排任务
- Core Modules：自带模块
- Custom Modules：自定义模块
- Plugins：完成模块功能的补充

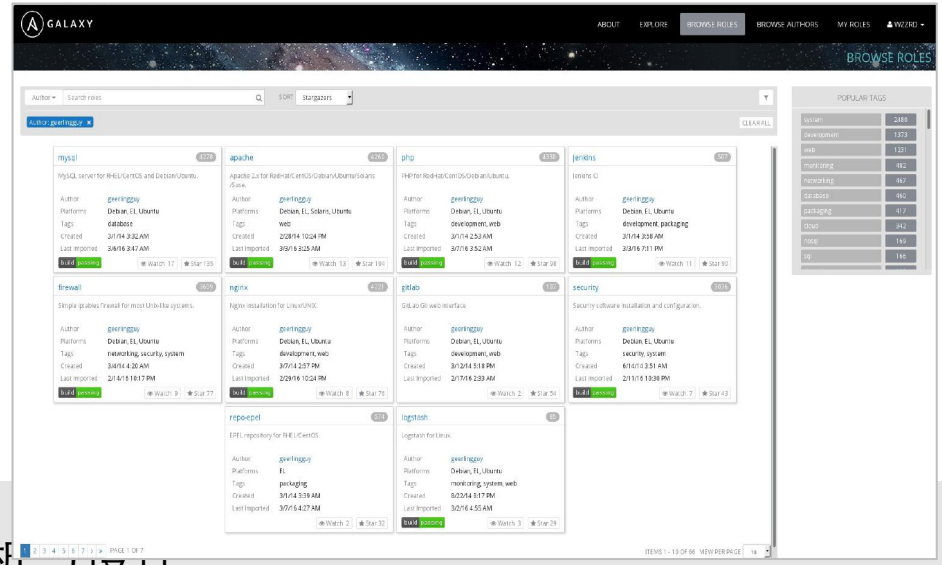
Ansible Playbook

```
---
- name: install and start apache
  hosts: all
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: install httpd
    yum: pkg=httpd state=latest
  - name: write the apache
configuration file
    template: src=/srv/httpd.j2
       dest=/etc/httpd/conf/httpd.conf
  - name: start httpd
    service: name=httpd state=running
```

- Playbook是ansible的配置、部署和编排语言。描述了管理远程主机的策略和常见IT过程的步骤。是一组IT程序运行的命令集合，可以简单理解为组合了多条ad-hoc操作的配置文件。
- Playbook用YAML格式表示，是一种配置或者过程的模型。用YAML语法把多个模块堆起来的一个文件。
- 一个play一般由4部分组成：
  - Target section：定义将要执行play的远程主机组
  - Variable section：定义play运行时需要使用的变量
  - Task section：定义将要在远程主机上执行的任务列表
  - Handler section：定义task执行完成以后需要调用的任务
- YAML文件以“---”作为文件的开始
- 列表中的所有成员都开始于相同的缩进级别，并且使用一个“- ”（一个横杠和一个空格）作为开头
- 运行： ansible-playbook playbook.yml -f 10 #设置10个并发 ansible-playbook playbook.yml --list-hosts #查看影响到的主机

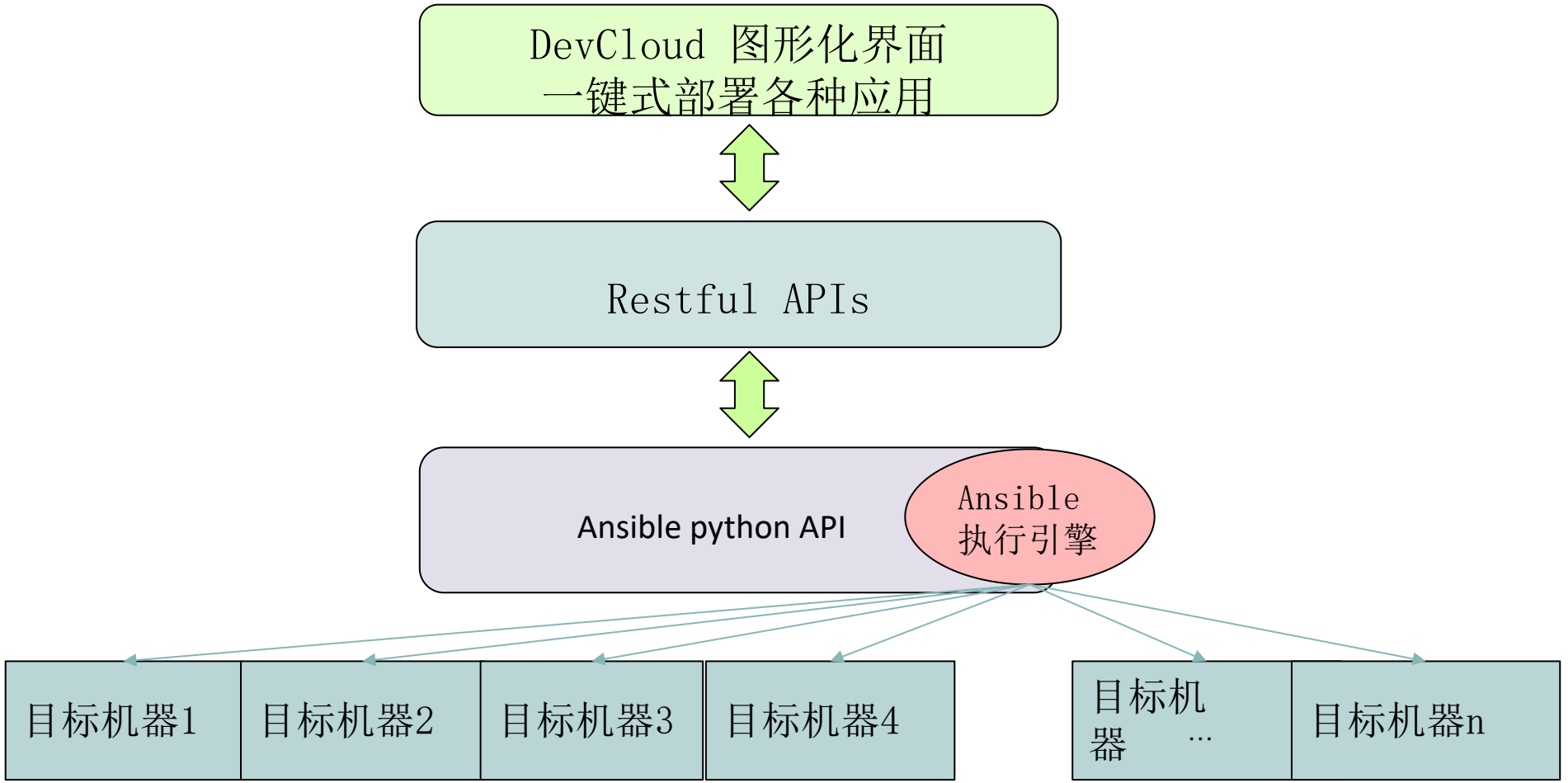
Ansible Galaxy

- 千千万万个系统和应用管理现成的、内容充实的角色，精确匹配你的环境
- 网站搜索或者使用命令工具
- 也可以通过Ansible安装
- 很容易找到流行的角色：有Github星、观察者和下载数量的评级
- 已经有5000以上的角色可供各种平台使用

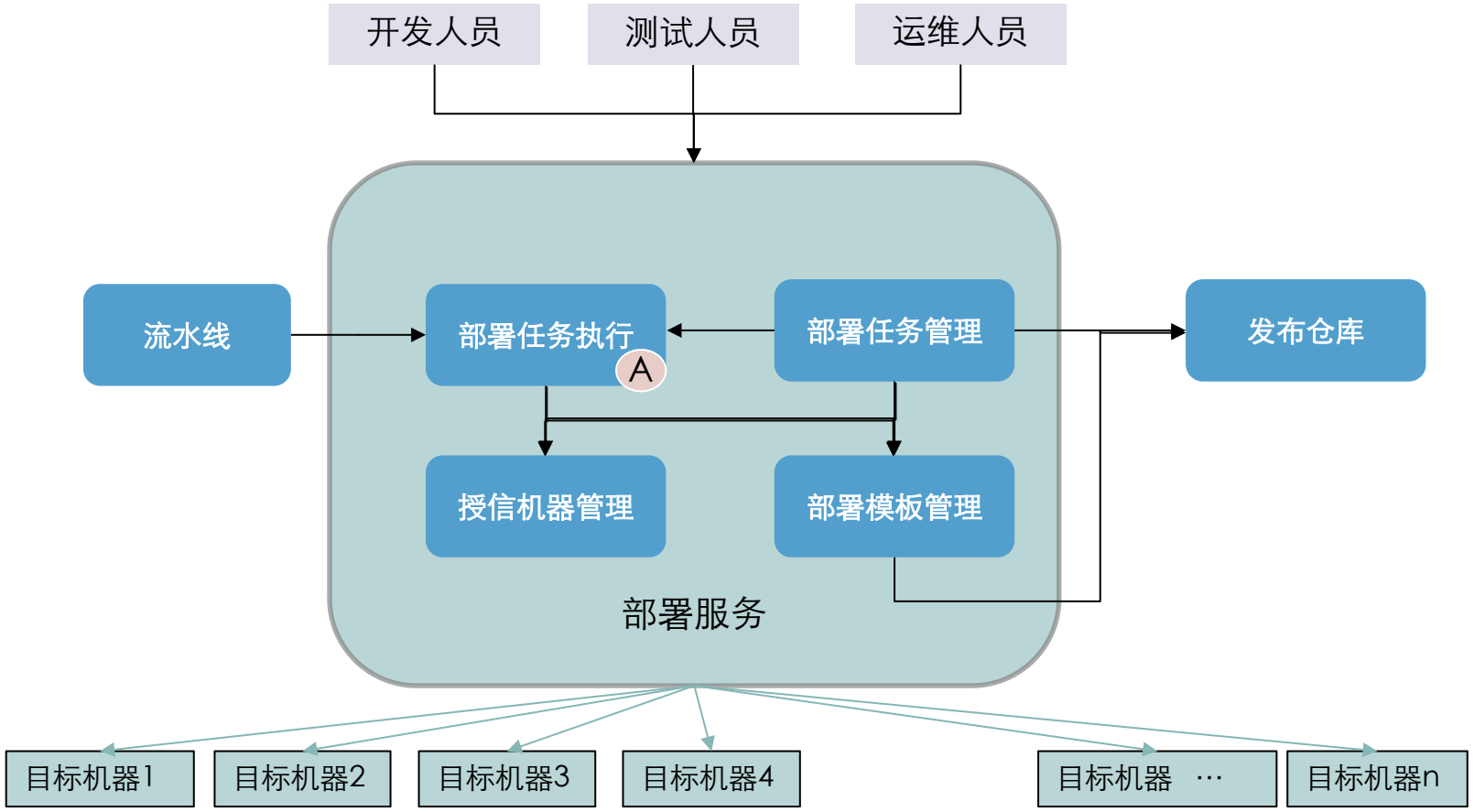


角色：galaxy.ansible.com网站上提供的可供重用的自动化脚本

DevCloud应用自动化部署



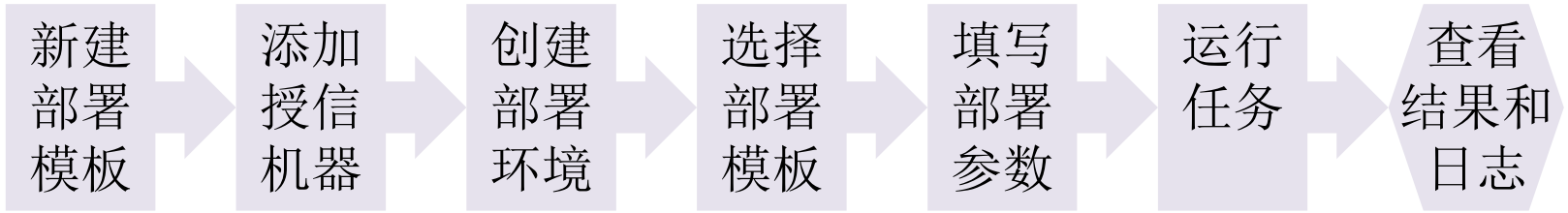
DevCloud部署服务功能模块



DevCloud应用自动化部署

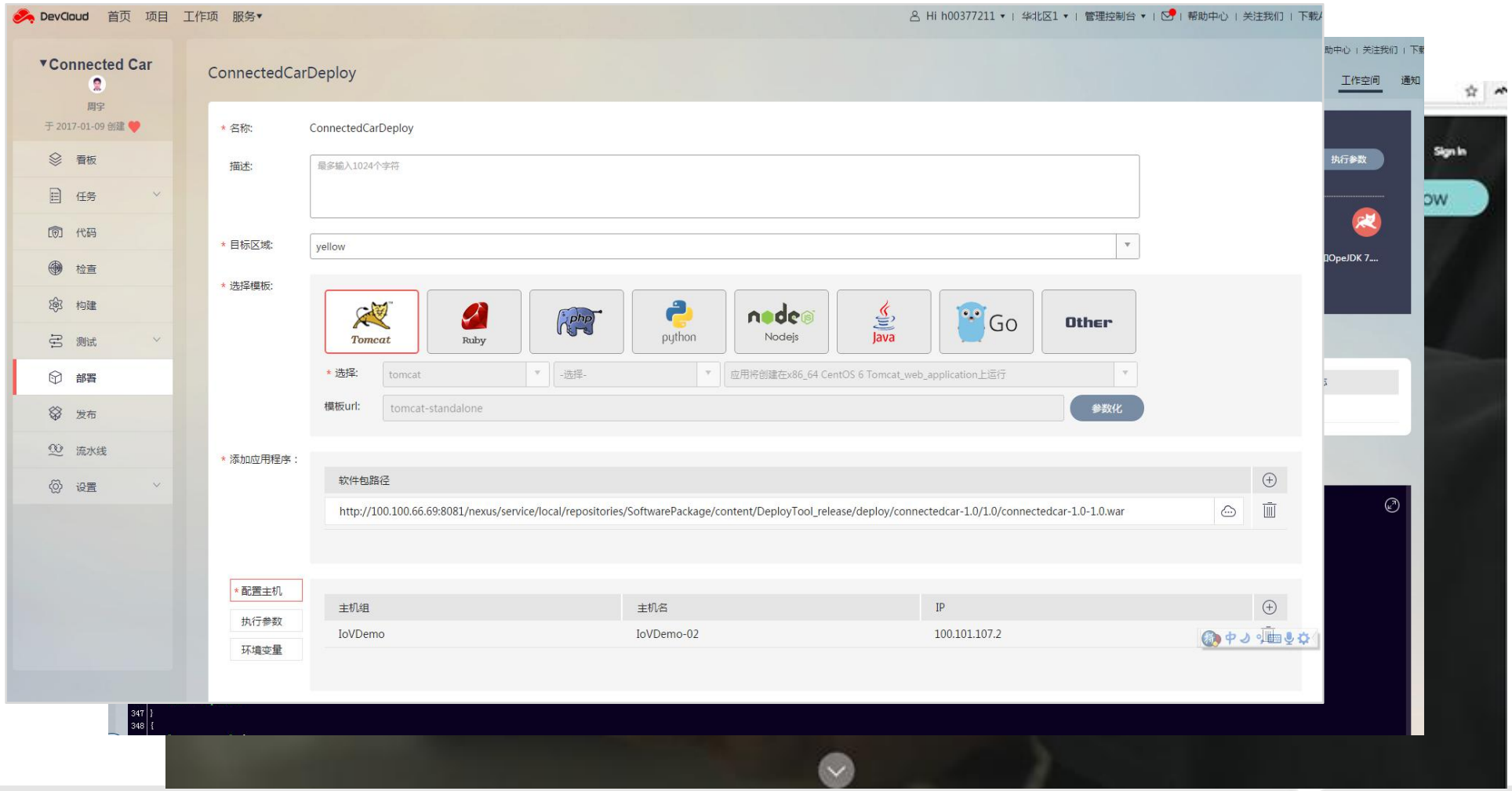
- 一键式部署主流应用
  - 支持Java、PHP、NodeJS、Python、Go和Tomcat应用
- 支持并行部署
- 和流水线无缝集成
- 目标机器无需安装Agent
- 部署操作可以多次高效执行
  - Ansible的幂等性操作
  - 优于传统的Shell脚本

部署使用流程（1/2）



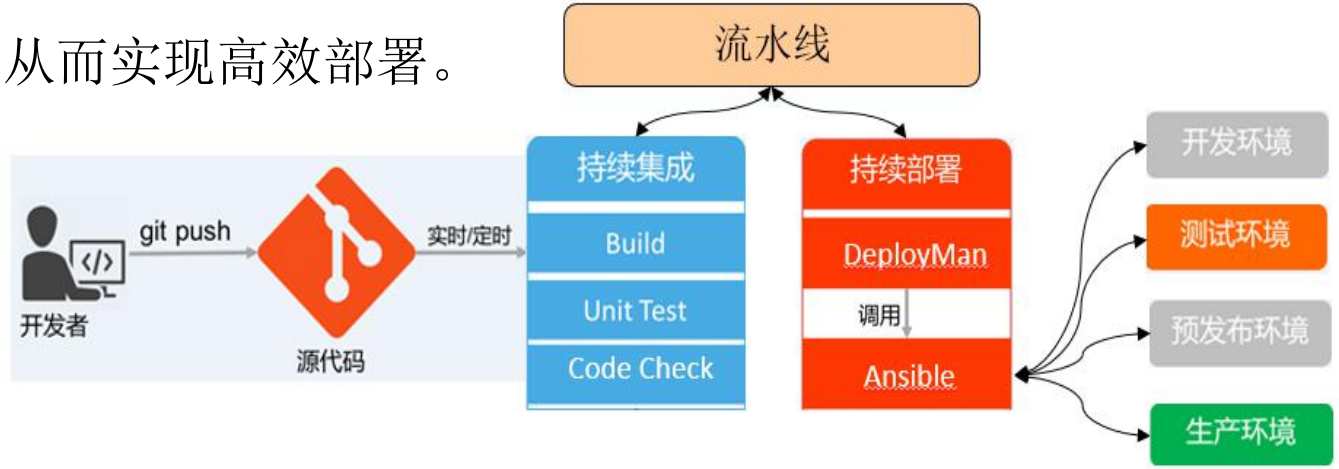
模板名称 模板类型 子类型 标签 Playbook-URL	主机名称 主机IP 用户名 口令	环境名称 别名 描述信息 应用名称 应用版本	模板名称 模板类型 过滤条件	环境变量 主机名称 下载源地址	监视部署状态 部署实例统计	查看日志 查看出错信息

部署使用流程（2/2）

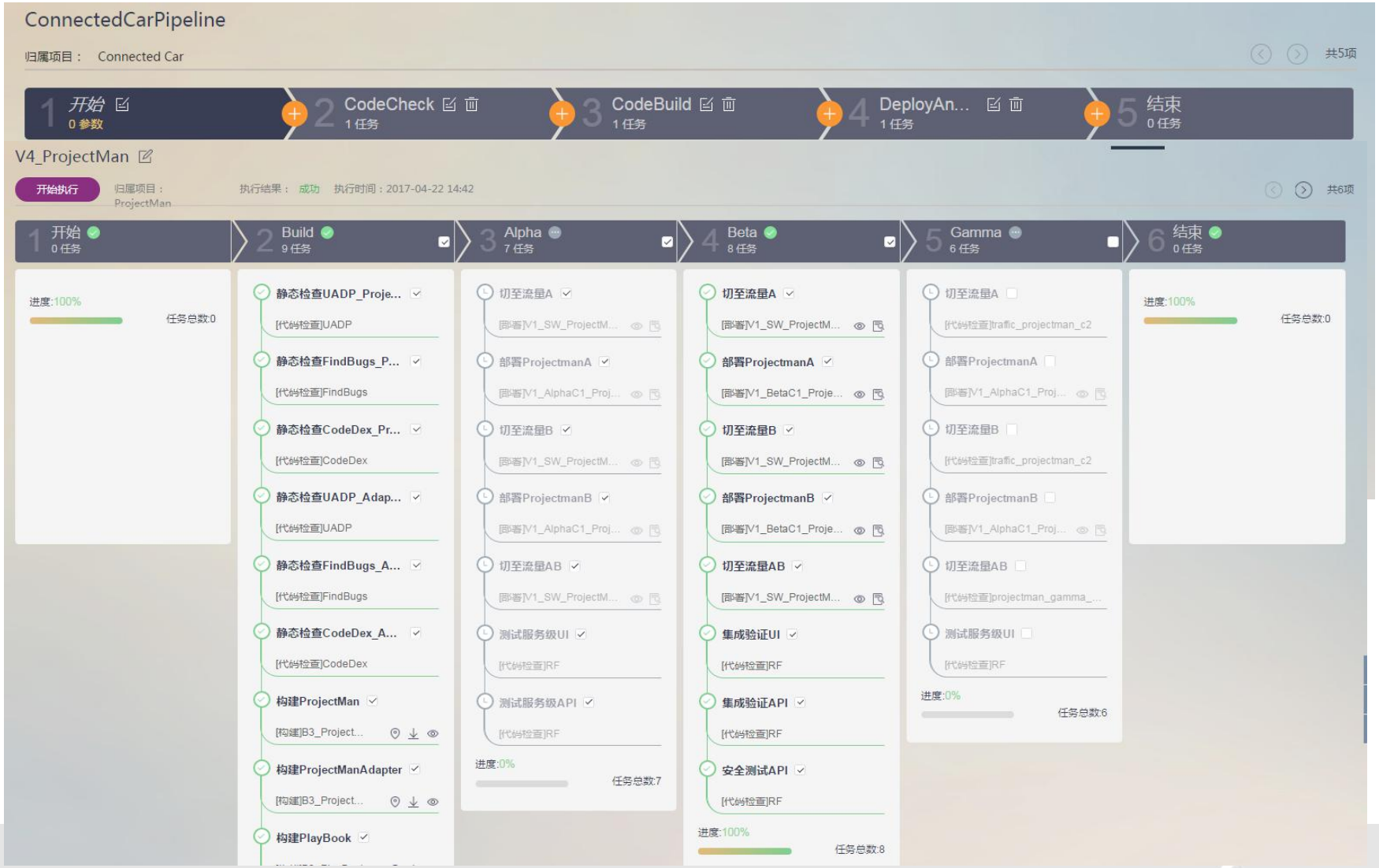

DevCloud
首页
项目
工作项
服务
Hi h00377211
华北区1
管理控制台
帮助中心
关注我们
Connected Car
周宇
于 2017-01-09 创建
看板
任务
代码
检查
构建
测试
部署
发布
流水线
设置
ConnectedCarDeploy
* 名称:
ConnectedCarDeploy
描述:
最多输入1024个字符
* 目标区域:
yellow
* 选择模板:
Tomcat
Ruby
php
python
Nodejs
Java
Go
Other
* 选择:
tomcat
-选择-
应用将创建在x86_64 CentOS 6 Tomcat_web_application上运行
模板url:
tomcat-standalone
参数化
* 添加应用程序：
软件包路径
http://100.100.66.69:8081/nexus/service/local/repositories/SoftwarePackage/content/DeployTool_release/deploy/connectedcar-1.0/1.0/connectedcar-1.0-1.0.war
* 配置主机
执行参数
环境变量
主机组
主机名
IP
IoVDemo
IoVDemo-02
100.101.107.2
工作空间
通知
执行参数
Sign In

流水线

- Ansible的playbook提供一定的服务编排能力。
- 在 playbooks 中可以编排有序的执行过程，甚至于做到在多组机器间，来回有序的执行特别指定的步骤，并且可以同步或异步的发起任务。
- 用流水线同时集成编译构建、代码检查、部署和自动化测试，从代码提交开始触发操作，完成整个DevOps的自动化流程。
- 流水线支持编排部署任务之间的依赖和顺序，通过串行任务和并行任务以及子流水线来组合、调度各个任务，从而实现高效部署。



通过流水线拉通DevOps



系统发布服务

- 一方面，软件服务变更通常在一个分布式、复杂的环境中进行。发布管理对于保障软件服务正确快速变更有种不可或缺的意义，发布管理能力除了是交付过程必备环节，随着对软件交付速度不断提升的诉求及激烈的市场竞争，它也逐渐成为一种核心竞争力。
- 另一方面，软件开发交付过程中，随着松耦合架构的推广，开源构件及框架的广泛使用，持续集成（CI）与持续交付（CD）等自动化手段的逐渐普及，软件构件的复用及依赖管理对发布准确性和及时性的影响越来越大，逐渐受到重视并日益规范。特别是在中大型项目中，良好的构件管理能带来较大的发布效率和软件质量提升。

发布管理定义（1/2）

- ITIL定义：

发布是由一项或多项经过批准的变更所组成的实施活动。一项发布是指经过测试并导入实际运作环境的一组配置项。发布管理需要确保只有那些经过测试和授权的正确的软件版本和硬件才能提供给IT基础设施。发布管理与配置管理和变更管理活动有密切的联系。

- 维基等来源的综合定义：

发布管理是IT服务管理的一部分，旨在规划、调度、控制和跟踪软件构件在开发、测试、部署、发布、维护等不同阶段和环境的版本控制和分发，确保发布正确的软件构件，保障生产环境的完整性、稳定性和及时性。

发布管理定义（2/2）

- 一种管理手段

  作为一种管理方法它会因不同的软件交付模式（如瀑布、敏捷、DevOps等）、产品类型、组织形态而衍生出不同的管理方式，没有统一的标准。

- 贯穿软件交付生命周期

  发布管理广义上说不针对某一个特定软件交付过程阶段的管理，比如开发管理、测试管理等。它的目标在于发布正确的软件构件到生产环境，让用户获得可用的软件服务。因此所有可以保障该目标达成的需求、缺陷、文档、代码、用例、软件包等管理过程都是其组成部分，紧密相关。

  狭义上主要指软件源码经过构建、测试并部署到各个环境的管理过程。

发布管理定义（2/2）

- 正确和快速发布的重要性

  软件服务作为一种商品形式，相对于质量、可靠性、性能等目标达成，更重要的是如何保障软件服务正确且快速的提供给用户使用。开发、测试、部署等阶段的管理往往仅聚焦于阶段内部而没有从整体交付使用的角度进行看护，而发布管理则是从软件服务交付可用的角度全面进行看护及跟踪，保持发布的规范、有序和可跟踪。

- 发布的复杂性

  软件项目的技术多样、依赖复杂、子服务或组件繁多、用户需求不同、交付响应快等种种因素让软件构件发布管理变得复杂而容易出错，因此需要有统一的流程、工具和组织团队来进行控制，尤其是对中大型的软件项目而言。

发布管理目标

ITIL中定义的发布管理目标包含：

- 规划、协调软件和硬件的实施（或安排实施）。
- 针对分发和安装对系统实施的变更而设计和实施有效的程序。
- 确保与变更相关的硬件和软件是可追溯和安全的（正确的、经过批准和测试的版本才能被安装）。
- 在新发布的规划和试运行期间与用户进行沟通并考虑他们的期望。
- 保障发布软件的原始拷贝被安全地存放在最终软件库，配置信息存储在配置管理数据库。

- 确保发布正确的软件服务，管理发布过程风险。

  1）软件交付是一个复杂的过程，往往涉及众多不同的组件、技术栈、质量要求、用户诉求、响应诉求、开发团队、角色等因素，发布涉及到整个交付过程端到端环节。规范统一的管理对风险识别和规避，协作效率等方面都有重大的意义。

  2）发布过程需要经验总结，更需要持续优化。

- 构建快速发布能力，提升产品竞争力

  1）发布不仅是一个交付动作也是一个商业活动，快速正确的发布是重要的产品竞争力。

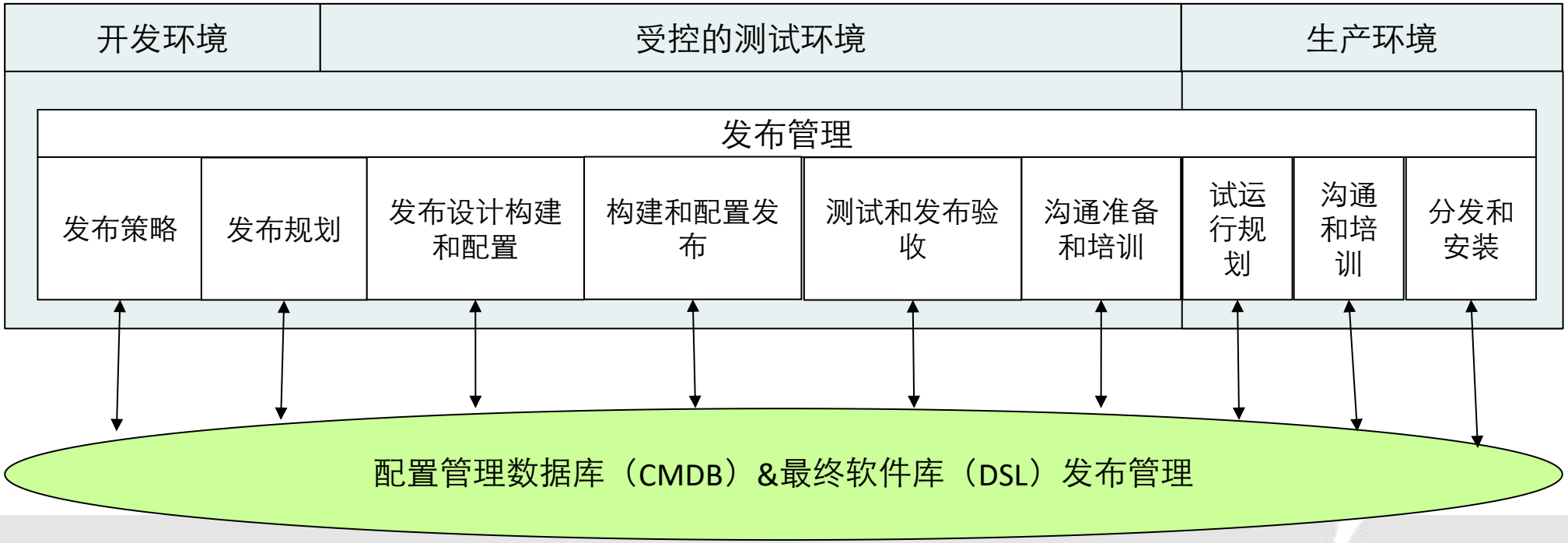
  2）快速迭代，MVP发布，及时获取反馈等敏捷和DevOps实践要求发布速度尽可能的快。

混乱随意不受控的发布无法满足软件服务交付准确性和高效性的要求

发布管理的关键活动

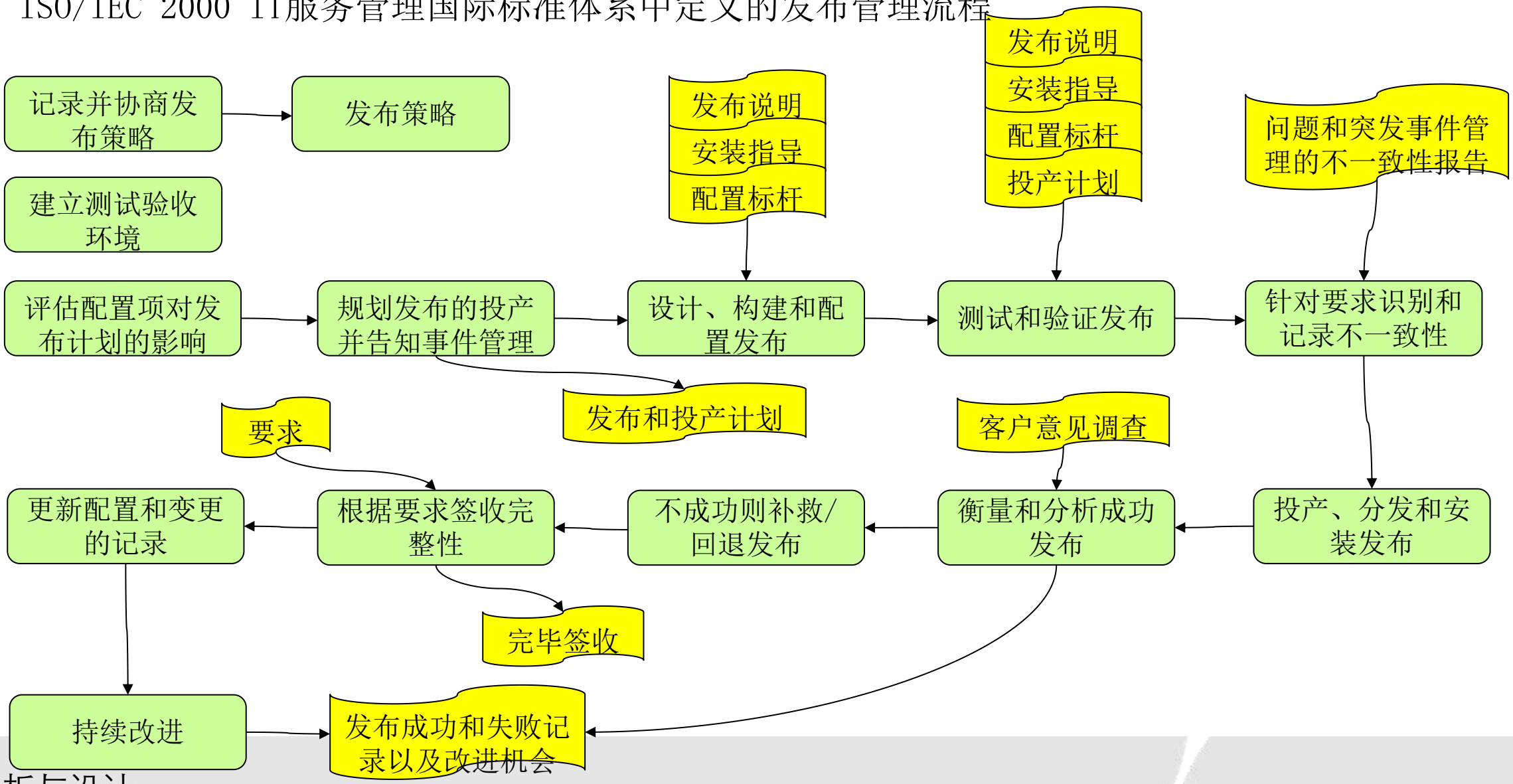
发布管理包含以下主要活动：

- 发布策略和规划制定
- 发布设计、构建和配置
- 测试及发布验收
- 试运行规划
- 沟通、准备和培训
- 发布分发和安装



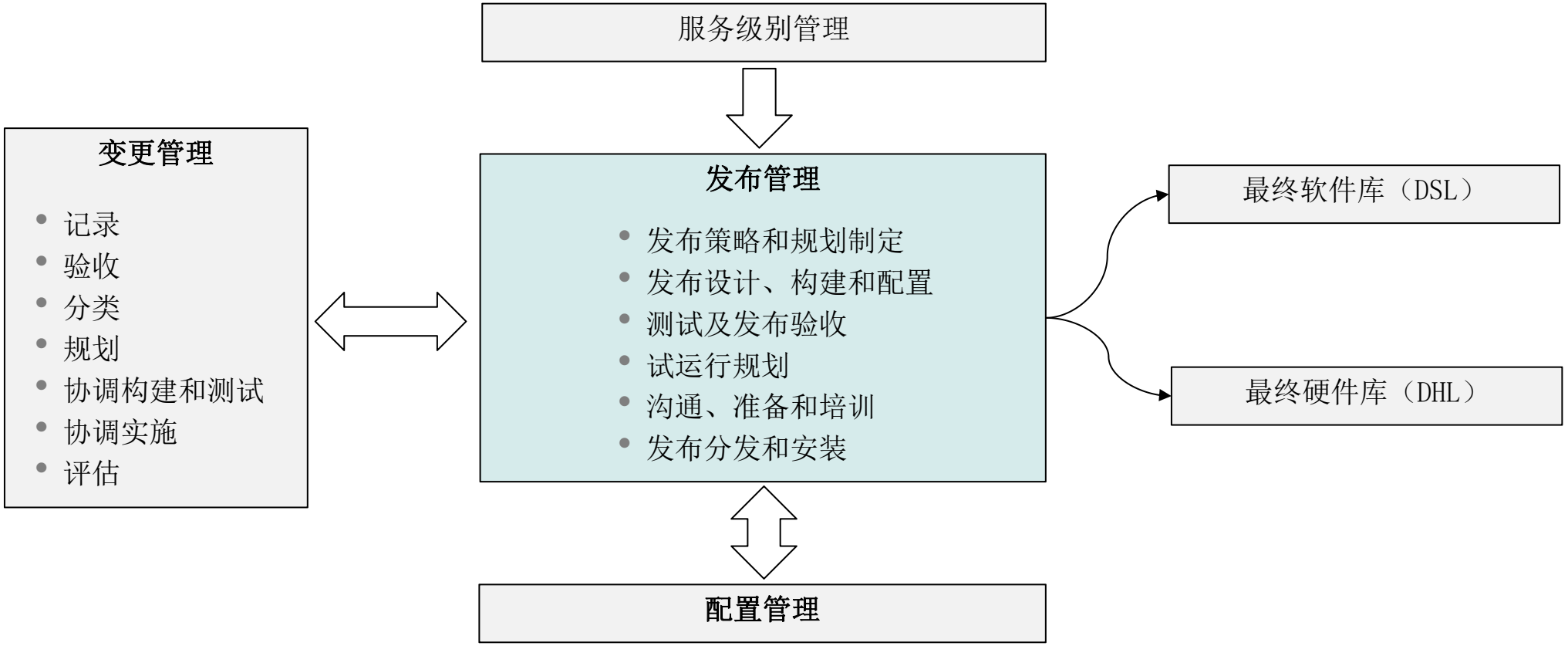
发布管理流程

ISO/IEC 2000 IT服务管理国际标准体系中定义的发布管理流程



发布管理流程与其他流程间的关系

成功的发布管理依赖于其他ITIL流程提供的信息和密切配合，下图展示了发布管理流程与其他流程的接口关系：



发布相关基本概念

- 重大发布（Major Release）

一般指有重大功能增强的版本发布，也通常会包含多个问题的修复。通常使用主发布版本号进行标识，如V.1>>V.2>>V.3

- 小型发布

一般指对问题或缺陷的小的改进或修复的版本发布.通常使用特征版本号进行标识，如V.1.1>>V.1.2>>V.1.3

- 紧急修复（Emergency Fixes）

一般指对某个问题进行的临时性修复，通常使用缺陷修复版本号进行标识，如V.1.1.1>>V.1.1.2>>V.1.1.3

- 开发环境

开发人员基于旧版本开发新版本进行功能调测的环境，可以频繁变更升级，经常部署在开发员本地。

- 测试环境

用于版本测试的环境，通常可以按照不同的测试场景区分不同的环境，如SIT环境、UAT环境等。

- 生产环境

用户使用的实际系统运行环境。

- 发布单元

发布单元描述的是出于对实施的变更进行控制和确保变更效果而同时发布的IT基础设施的组合。

发布类型

- **德尔塔发布（Delta Release）**

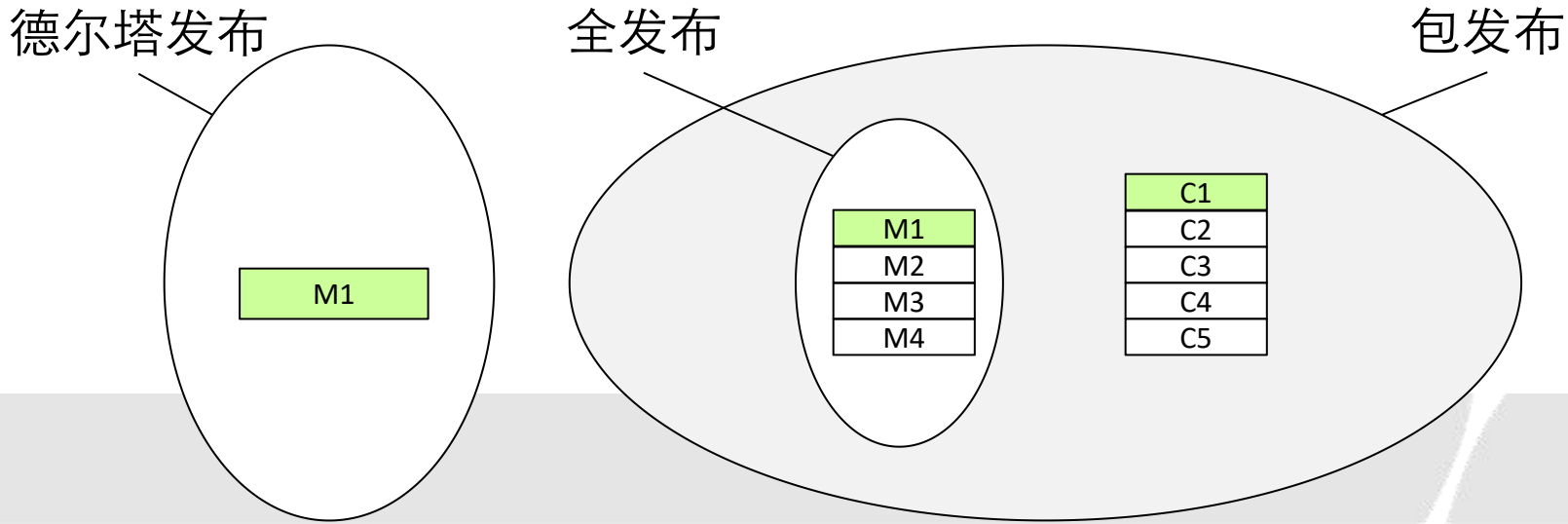
一种局部增量发布，只包括发生变更的硬件和软件组件。通常在紧急修复或临时修复中使用。其优点是只需花 很少的时间就能完成测试环境构建；缺点是无法对变更组件以外的环境和依赖组件测试通常不够彻底。

- **全发布（Full Release）**

指同时对发布单元内所有的组件进行构建、测试和分发，包括哪些无需变更的组件。这种发布方式下，软件和硬件将得到更加彻底的测试，对那些不是完全清楚哪些组件会发生变更的场景特别有用。但是会比德尔塔发布需要更多的准备工作和资源。

- **包发布（Package Release）**

指一组相关的应用系统和基础设置的全发布和（或）德尔塔发布组成。一般在一个更长的时间维度内进行，它通过修复小的软件错误以及将多项新的功能有效组合到一起为用户提供更长时间的稳定期。



主要的软件发布阶段

- **Pre-alpha发布：** 测试前阶段，包含需求分析、方案设计、编码开发、单元测试等多个活动。期间会构建多个不稳定软件包进行功能调测
- **Alpha发布：**alpha版本是首个软件测试版本，可能不稳定、也可能不包含所有最终版本的特性。开发人员在这个阶段进行白盒测试后移交给测试团队进行黑盒测试。通常该版本不会面向外部用户测试。
- **Beta发布：**beta版本是首个完整特性的软件测试版本，但依然存在较多的功能、易用性、性能等缺陷。通常有内部beta和开放beta两种方式。内部beta局限于内部测试、演示，会邀请部分用户代表参与但不会大范围开放。开放beta则面向尽可能多的外部用户开放，用于最终发布前向用户展示宣传产品并通过用户公测发现隐藏问题。
- **RC/Gamma发布：** RC版本是预发布版本，所有的计划功能都已完成设计、编码并经过多轮的beta测试同时没有遗留的严重缺陷。该阶段开发人员仍然可以进行代码修改来修复缺陷等操作。用户代表通常会在此版本进行验收测试及公测试用。如果没有重大紧急的缺陷出现，RC版本将作为最终版本发布。
- **最终发布：**正式发布版本，面向所有最终用户，开发人员无法再对该版本进行代码修改，只能通过补丁版本修复缺陷等问题。

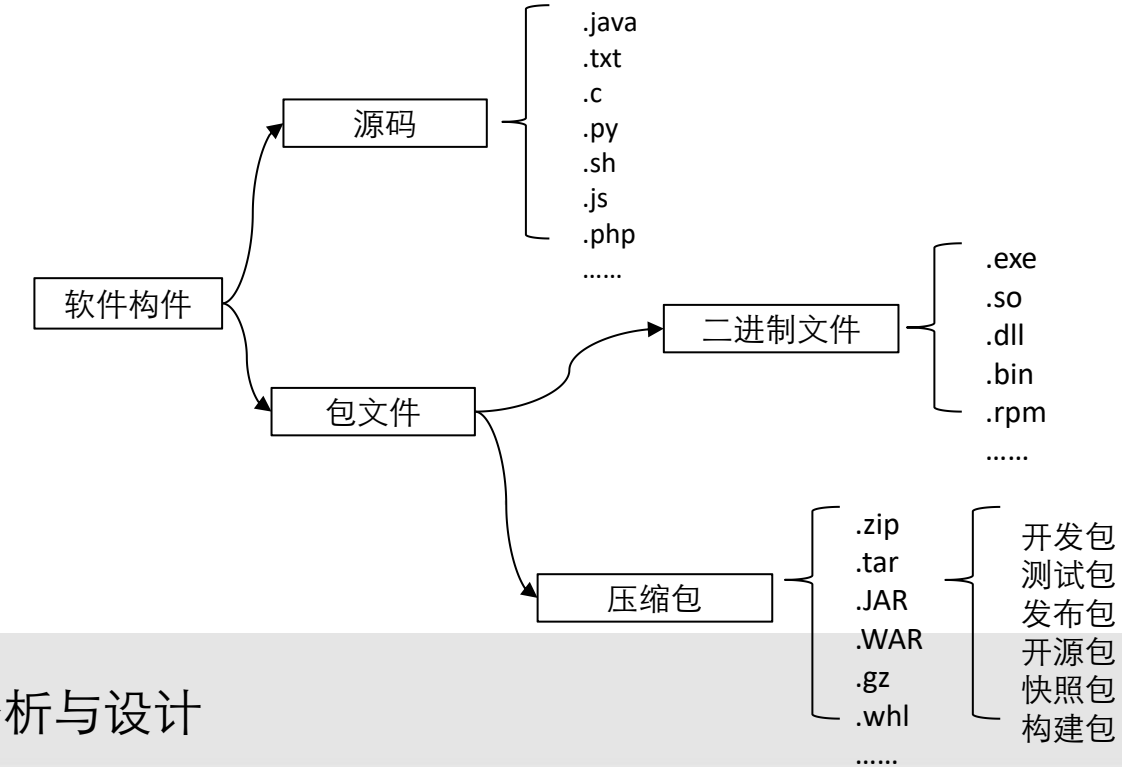
软件构件

软件构件概述

- 软件构件分类
- 源码及包文件的不同使用特征
- 源码及包文件配置管理

软件构件概述 – 构件分类

- 文件是一个有序的数据集合，它拥有一个特定的开始和结尾数据项。文件技术上的定义跟计算机环境的文件系统相关，由文件系统控制其产生和访问。
- 软件交付过程中通常产生的构件可以大致分为源码和包文件两种。包文件通常是源码文件的集合或者编译后的产物，因此主要有二进制包和压缩包两种形式。包文件的管理和复用在发布管理有着关键的作用。由于两种类型的构件产生和使用的方式不同，它们有着不同的属性描述和管理方法：



源码文件	包文件
文件名	文件名
版本	版本
大小	大小
创建时间	创建时间
	依赖
	许可
	风险
	……

源码及包文件的不同使用特征

- 包文件相对于源码文件，通常较大（M~G级别都有），一般修改也较少且通常采用覆盖方式。
- 包文件一般的生命周期环节也较多，因此需要管理更多的元属性来标识不同阶段的状态，通常包文件不放在源码库中一同管理。

源码文件	包文件
经常频繁修改	修改较少
一般较小	通常较大
增量修改	覆盖修改
修改增量存储	修改全量存储
频繁对比，分支，标签	基本没有
属性值较少	属性值多
异地分发简单	异地分发较困难

源码及包文件配置管理简介

- 源码配置管理
  - 主要是指对源码修改进行版本控制，实现多人对源码操作的协同作业。主要操作涉及签出/签入、分支、冲突、合并等，通常使用SVN、Git、clearcase等版本控制工具进行管理。
- 包文件配置管理
  - 主要是指对软件研发过程中使用的私有包和开源包的管理，用于构建持续集成和持续发布自动化能力。主要涉及包文件的下载、搜索、上传、依赖跟踪、license控制、属性管理等操作，由于包文件与源码的使用区别，通常使用单独的仓库工具进行存储（业界有Nexus，Artifactory等开源工具）并配合包依赖管理工具使用（如Maven、npm、ivy等）。

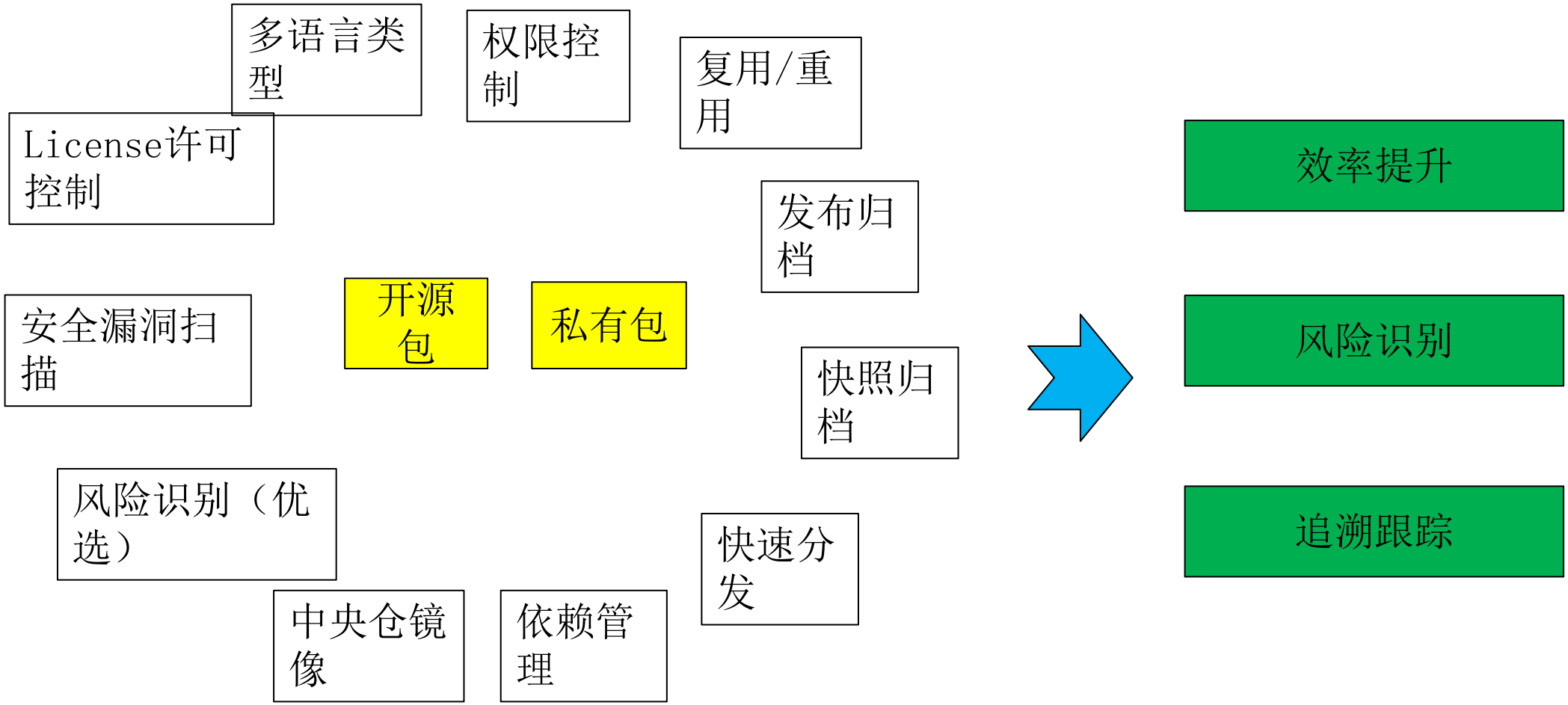
问题讨论

- 包文件是否应该保存在源码库中?

包文件管理概述

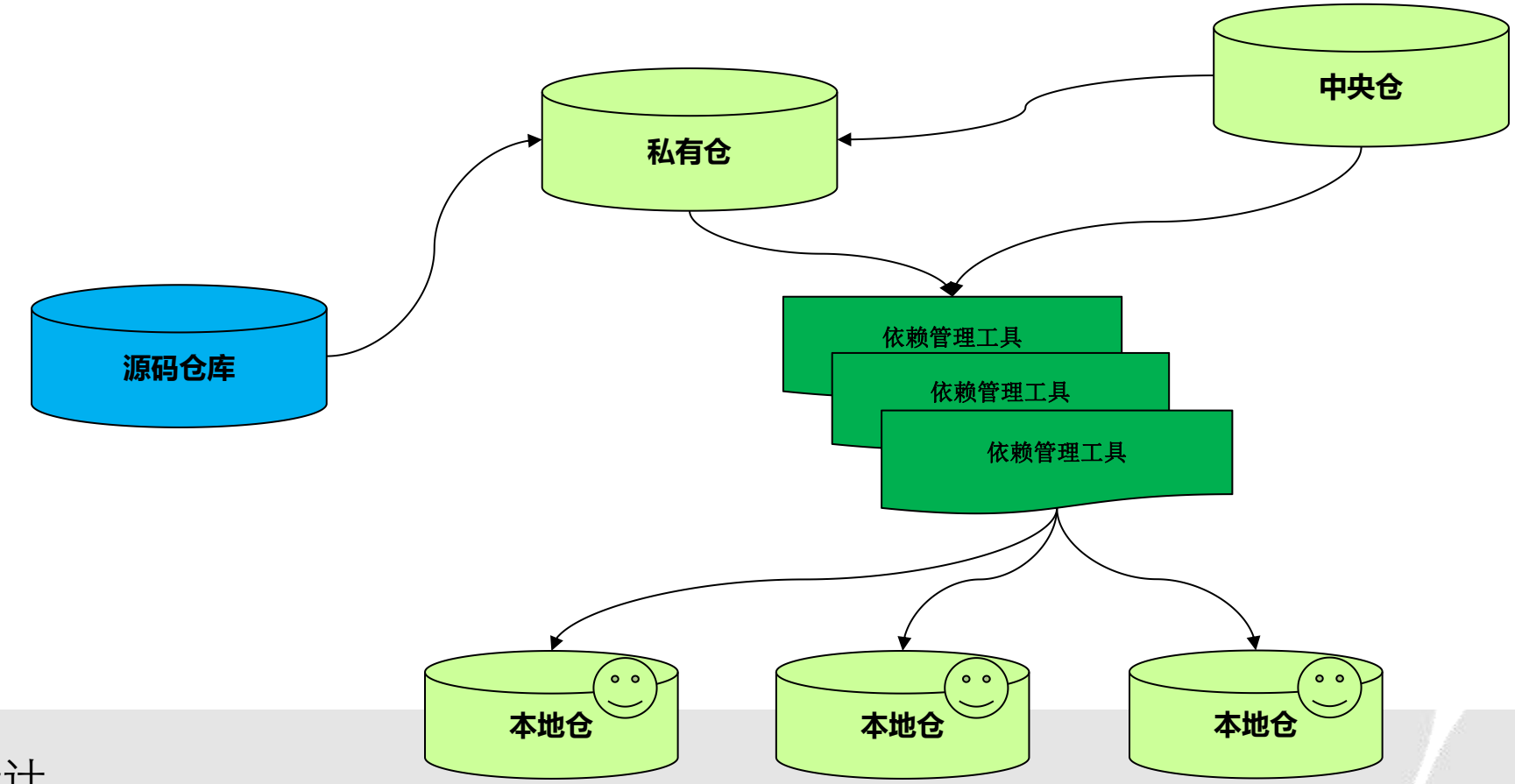
- 为什么要进行包文件管理
- 包文件一般管理方法
- 开发人员如何使用包文件
- 常用开源包中央仓库

为什么要进行包文件管理



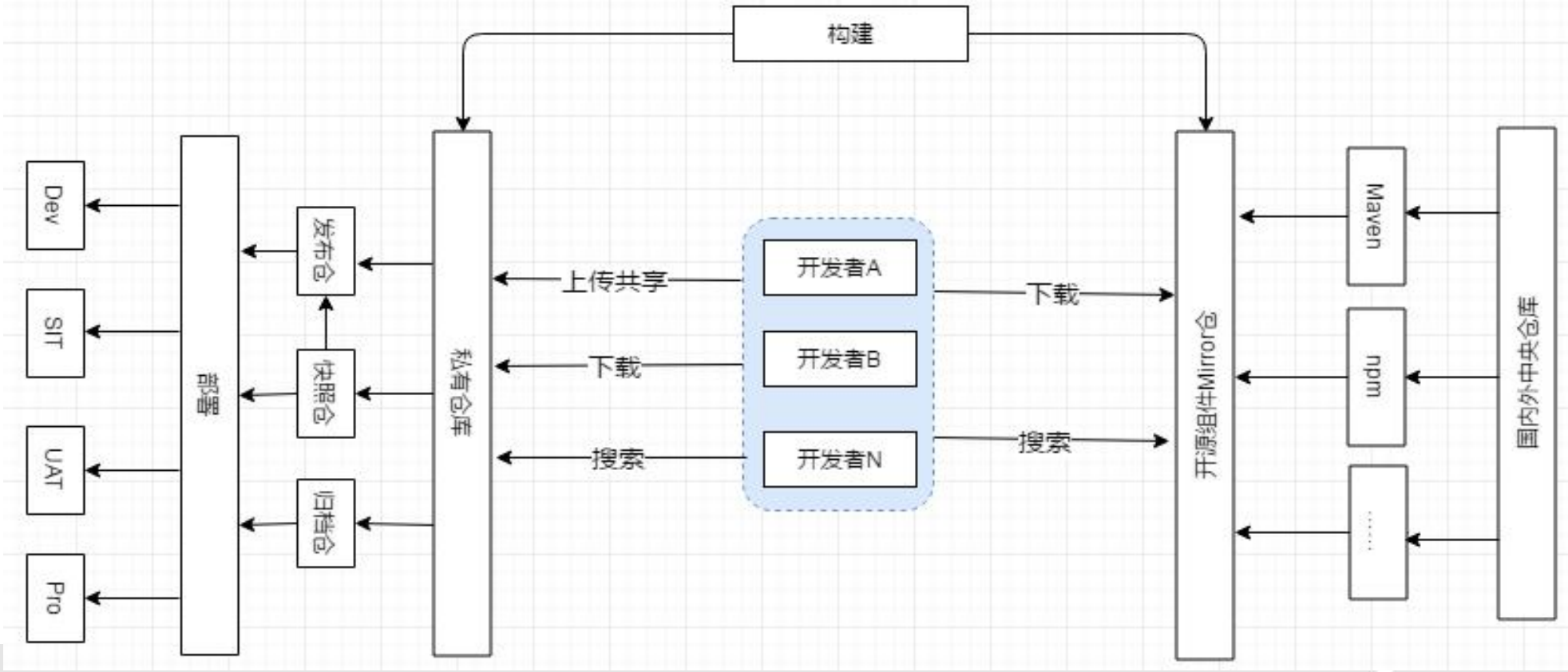
包文件一般管理方法

- 包文件通常不放在源码库中管理，而是使用专门的包文件仓库（repository）进行存储并配合包文件依赖管理工具（Maven、npm、Ivy等）进行使用。包文件仓库可以大致分为本地仓库、私服仓库、中央仓库三种。本地仓库是指开发者个人PC中包文件的存储；私服仓库通常是企业为了提升包文件使用性能搭建的局域网内共用的包文件仓库，通常使用开源的Nexus、artifactory等工具搭建；中央仓库是指开源包文件的共享社区。



开发人员如何使用包文件

- 开发人员对包文件的使用集中在下载、搜索、发布上传几个操作上。开发和构建时，开发人员通过包依赖管理工具定义好需要使用的私有及开源包文件，在构建或运行时自动从私服仓库或开原中央仓库中下载依赖包文件来提升开发效率。



常用开源包中央仓库

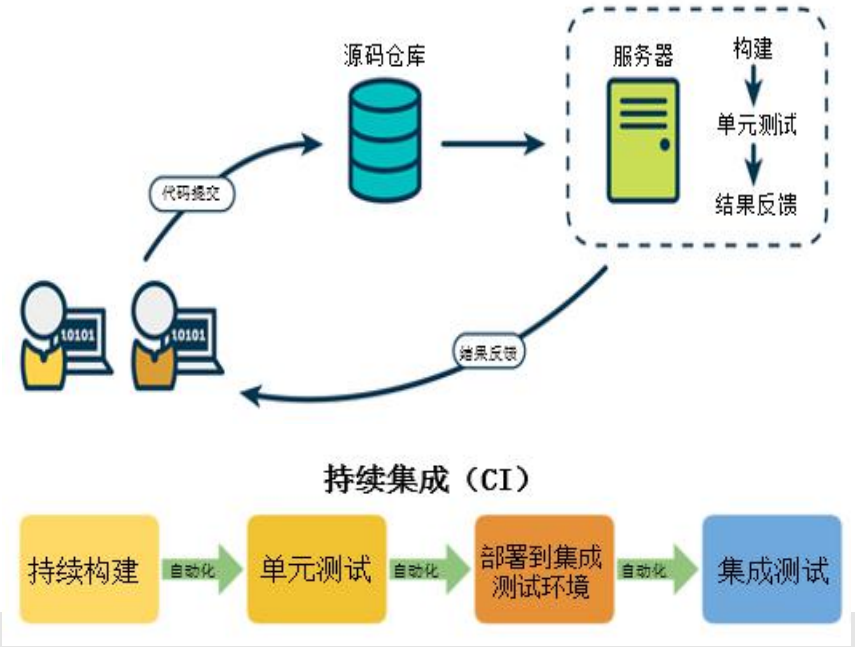
- Maven中央仓（java）
- http://search.maven.org/
- http://central.maven.org/
- http://repo2.maven.org/
- http://repo1.maven.org/
- http://uk.maven.org/
- https://repo.maven.apache.org/
- http://mvnrepository.com/
- https://bintray.com/bintray/jcenter
- NPM中央仓（nodejs）
- https://www.npmjs.com/package/npm-registry
- Nuget中央仓（.net）
- https://www.nuget.org/packages
- Bower中央仓(web)
- https://bower.io/search/
- pyPI中央仓(python)
- https://pypi.python.org/pypi
- RubyGems中央仓(ruby)
- https://rubygems.org/gems
- Opkg中央仓(OpenWrt)
- http://downloads.openwrt.org
- Debian中央仓(Debian软件包）
- https://www.debian.org/distrib/packages
- Cocoapods中央仓(swift&Objective-C)
- https://cocoapods.org/

包文件管理在DevOps中的应用

- 持续集成（CI）
- 持续交付（CD）
- 包文件仓库在DevOps工具链中的作用
- 包文件仓库管理的关键特性

持续集成（CI）

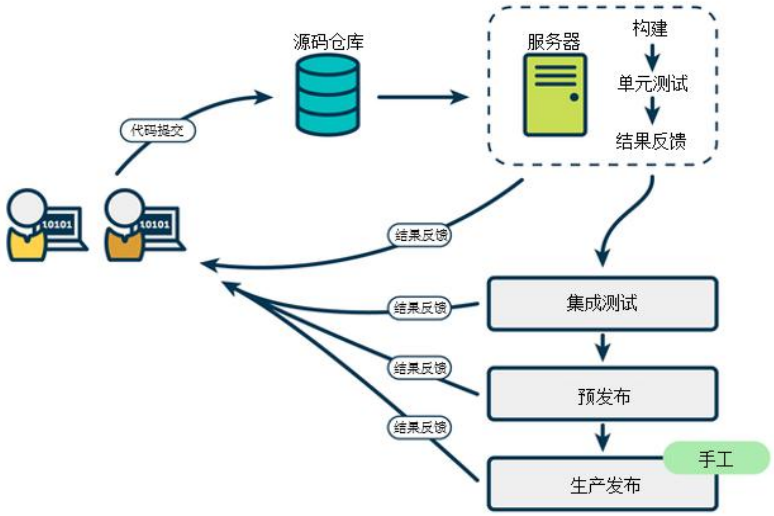
- Martin Fowler：持续集成是一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建（包括编译，发布，自动化测试)来验证，从而尽快地发现集成错误。许多团队发现这个过程可以大大减少集成的问题，让团队能够更快的开发内聚的软件。Grady Booch在1994年首次提出该概念，1997年被Kent Beck and Ron Jeffries引入作为极限编程方法的一部分。

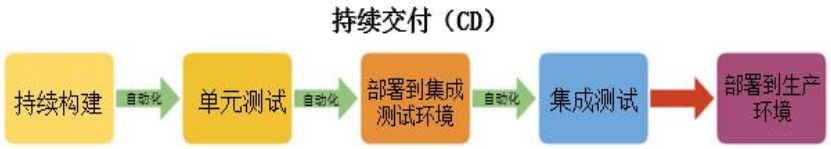


- 持续集成最佳实践：
- 维护一个统一的代码库
- 实现自动构建
- 实现自动单元测试
- 每个团队成员每天至少提交一次代码到主干
- 每次主干提交都出发一次自动构建
- 及时处理构建阻塞
- 提升构建性能保障快速构建
- 模拟生产环境进行自动测试
- 保障每个人可快速获取最新可工作的应用程序
- 保障信息共享

持续交付（CD）

- 持续交付：是一种软件工程手法，让软件产品的产出过程在一个短周期内完成，以保证软件可以稳定、持续的保持在随时可以释出的状况。它的目标在于让软件的建置、测试与释出变得更快以及更频繁。这种方式可以减少软件开发的成本与时间，减少风险。持续交付在持续集成的基础上，将集成后的代码部署到更贴近真实运行环境的「类生产环境」（production-like environments）中。





持续交付最佳实践：

- 可视化：团队中每一个成员对交付过程中的构建、测试、部署、发布等环节信息都能及时接收和处理以保障交付高效协同。
- 反馈：团队成员能第一时间收到问题反馈以便进行尽可能快的修复。
- 持续部署：打造持续交付流水线，保障每一个版本的应用程序可以快速部署到任意环境中。

持续交付带来的好处：

- 快速发布。能够应对业务需求，并更快地实现软件价值。
- 编码->测试->上线->交付的频繁迭代周期缩短，同时获得迅速反馈。
- 高质量的软件发布标准。整个交付过程标准化、可重复、可靠。
- 整个交付过程进度可视化，方便团队人员了解项目成熟度。
- 更先进的团队协作方式。从需求分析、产品的用户体验到交互 设计、开发、测试、运维等角色密切协作，相比于传统的瀑布式软件团队，更少浪费。

包文件仓库在DevOps工具链中的作用

IDE对接

快速获取依赖包

识别开源包风险

中央仓库对接

开源包镜像，提升下载效率

开源包策略管理，过滤有害包文件

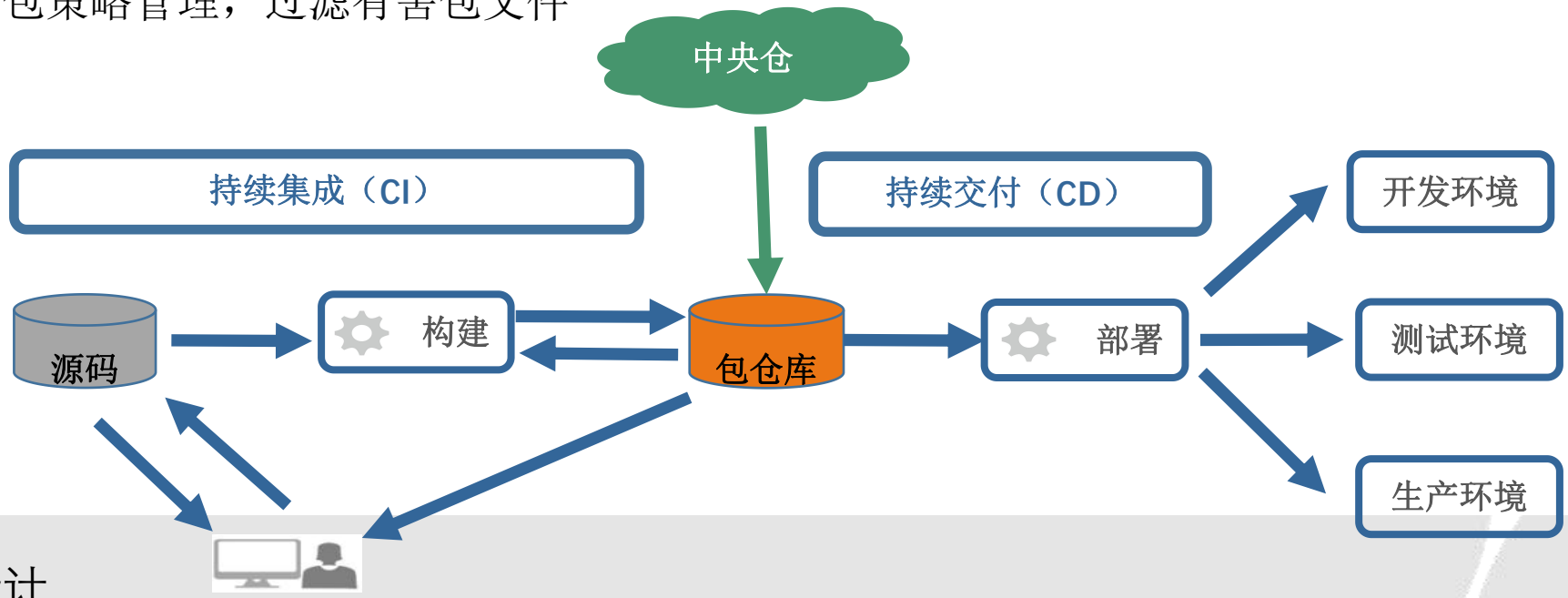
构建工具对接

复用私有包，提升构建效率

开源依赖包获取，提升构建效率

部署工具对接

利用有序调度不同环境部署，实现持续交付



包文件仓库管理的关键特性

- **高可用：**包文件管理仓库处于DevOps自动化工具链的中心，其稳定性决定了整个交付过程的稳定，一次高可用对于每一个包文件管理工具都至关重要，应用层常见的有server-slave和alive-alive两种方式，存储层一般稳定性较高，但通常需要备份保障容灾问题。
- **多仓库类型支持：** 一方面，软件项目目前技术区域多样化，会使用及产生多种类型的第三方开源包和私有包。另一方面，不同类型的包文件标识方式、使用方式、对接的包依赖管理工具等都不同。因此包文件管理工具需要能支撑不同类型的包文件类型，通常有Maven（java）、npm（nodejs）、Nuget（.net）、Bower等类型。
- **开源第三方库代理和镜像存储：**大多数开源第三方中央仓库在国外，公网下载较慢。另外，许多企业使用局域网络或专有网络，部署自己的开源镜像私服可以有效提升包文件使用效率，首次被使用后即可被镜像存储在私服中，方便后来者使用。
- **生命周期管理：**包文件（主要是指私有包文件）在不同的研发阶段展现不同的属性和用法，如在构建、部署、测试等环节需要能管理各自的环境参数、任务时间、报告、QA审视等属性。

包文件仓库管理的关键特性

- **健康防护和检查**

  开源包等广泛使用也为软件带来了风险，比如漏洞、不稳定、license许可、病毒等，因此需要管理工具具备方案和策略管理能力。另外对于私有包文件，一致性检查，组件追溯等对构建稳定的软件功能也有很大的意义。

- **异地分发**

  对于大型的软件项目，常常涉及异地开发协作，因此需要分布式节点才能保障使用性能。另外容灾等场景也常常需要进行异地备份。

- **主流IDE、构建工具、包依赖管理工具集成**

  包文件管理工具最大的价值在于打通CI和CD环节，因此跟主流的自动化工具集成是基础能力。常见的集成工具有：eclipse、intellijIDEA、Maven、gradle、ivy、jenkins、Bamboo、puppet等

Maven工具依赖管理

Maven工具依赖管理介绍

- Maven工具简介
- Maven坐标简介
- Maven的依赖配置
- Maven的依赖传递管理

Maven工具简介

- Maven是一款跨平台的Apache开源项目管理工具，主要服务于基于Java平台的项目构建、依赖管理和项目信息管理。它奉行约定优于配置（Convention Over Configuration）原则。

使用Maven开始你的工作

创建一个Java项目

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
```

创建一个Web项目

```
mvn archetype:generate
-DgroupId=org.yourcompany.project
-DartifactId=application
-DarchetypeArtifactId=maven-archetype-webapp
```

创建一个已有项目的原型

```
mvn archetype:create-from-project
```

主要的生命周期阶段

clean — delete target directory
validate — validate, if the project is correct
compile — compile source code, classes stored in target/classes
test — run tests
package — take the compiled code and package it in its distributable format, e.g. JAR, WAR
verify — run any checks to verify the package is valid and meets quality criteria
install — install the package into the local repository
deploy — copies the final package to the remote repository

常用的Maven命令行选项

-DskipTests=true compiles the tests, but skips running them

-Dmaven.test.skip=true skips compiling the tests and does not run them

-T - number of threads:
-T 4 is a decent default
-T 2C - 2 threads per CPU

-rf, --resume-from resume build from the specified project

-pl, --projects makes Maven build only specified modules and not the whole project

-am, --also-make makes Maven figure out what modules out target depends on and build them too

-o, --offline work offline

-X, --debug enable debug output

-P, --activate-profiles comma-delimited list of profiles to activate

-U, --update-snapshots forces a check for updated dependencies on remote repositories

-ff, --fail-fast stop at first failure

Maven坐标简介

Maven的世界中拥有数量庞大的软件构件，在引入坐标概念前没有任何一种统一的方式来唯一标识一个构件，人们要想使用这些构件只能在分散的网站上进行搜索，无法自动化，费时费力。

Maven中定义了一组规则：世界上任何一个构件都可以使用maven坐标来唯一标识。

Maven坐标的元素包括：groupId、artifactId、version必须，packaging可选，classifier不能直接定义

- groupId
  定义当前Maven项目隶属的实际组织项目（一个组织项目可能包含多个Maven项目）.该元素表示方式与java包名类似，与反向域名一一对应，如org.sonatype.nexus,但通常不建议定于Maven项目对应的组织或公司，因为一个组织或公司下可能会有多个项目。
- artifactId
  定义实际组织项目中的一个Maven项目（模块），通常推荐使用“组织项目名-maven项目名”方式表示以方便后续构件寻找，例如nexus-indexer。默认情况下，Maven中生成的构件名称会以artifactId作为开头，如nexus-indexer-1.2.jar。
- version
  定义当前Maven项目的所处的版本，如1.0.0；2.0.0。
- packaging
  定义当前Maven项目的打包方式，通常打包方式与所生成构件的文件扩展名相对应，如jar、war。
- classifier
  用来帮助定义构建输出一些附属构件，如xxx-javadoc.jar,xxx-source.jar。

Maven的依赖配置

- POM：Project Object Model，Maven工具中一个项目所有的配置都放置在 POM 文件中：定义项目的类型、名字，管理依赖关系，定制插件的行为等等。
- 在 POM 中，groupId,artifactId, packaging, version 叫作 maven 坐标，它能唯一的确定一个构件。有了 maven 坐标，我们就可以用它来指定我们的项目所依赖的其他项目，插件，或者父项目。
- 在 POM 中，依赖关系是在 dependencies部分中定义的。

```html
<dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

Maven的依赖传递管理

- 间接依赖：被使用的依赖包，通常自身还依赖了其他的开源包文件，因此构成了复杂的依赖关系。当你引用此种依赖包时，就产生了对其他依赖包的间接依赖。
- 那么对于间接依赖Maven是如何处理的呢？Maven中提供了依赖传递的特性，通过POM文件实现，当你使用某一依赖包时，它的POM文件会被一同下载，继而将他自身的依赖文件也一并下载下来。
- POM文件中通过scope字段来标识依赖关系的生效范围：

Scope一般有以下几种属性值：

<test>
指测试范围有效，编译和打包时都不使用该依赖。

<compile>
（为默认值）编译范围有效，编译和运行（打包）时都会将依赖存进去

<provided>
测试、编译范围都有效，最后生成war包时不会加入。

<runtime>
编译时不依赖，运行（打包）时依赖。

另外，当发生依赖包冲突等情况时，也可以使用exclusion标签排除依赖来完成依赖调解。

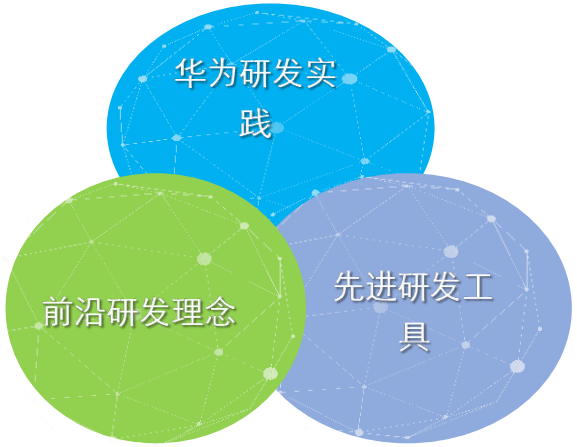
华为云（Devcloud）发布管理服务

华为云（Devcloud）发布管理服务介绍

- 华为云（DevCloud）简介
- Devcloud如何进行发布管理
- Devcloud发布管理服务介绍

软件开发云（DevCloud）简介

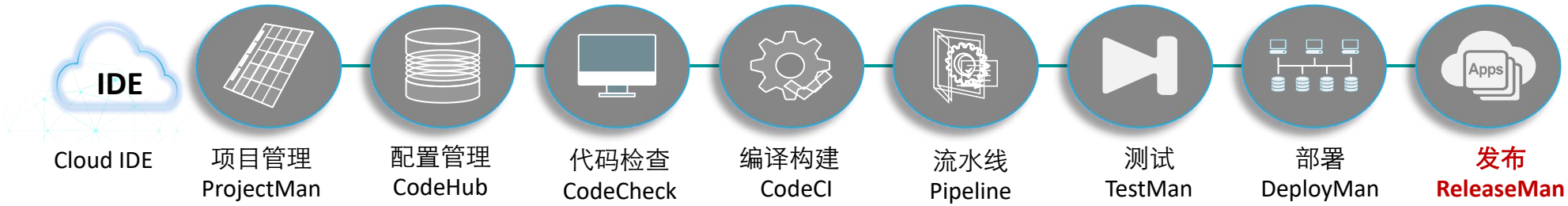
- **软件开发云**（Devcloud）是集华为研发实践、前沿研发理念、先进研发工具为一体的研发云平台；面向开发者提供研发工具服务，让软件开发简单高效。



Web开发（Web Application）

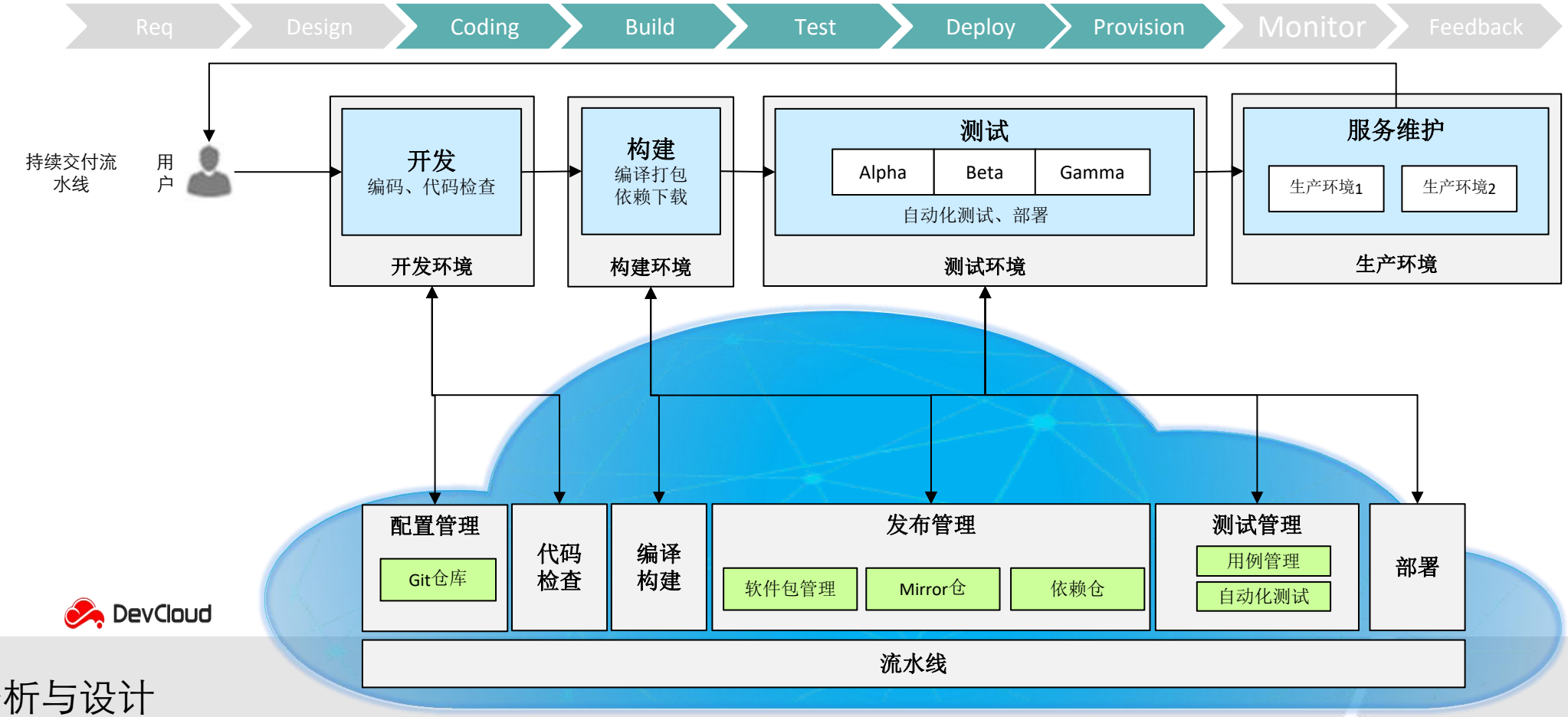
微服务开发（MicroService）

移动应用开发（Mobile APP）



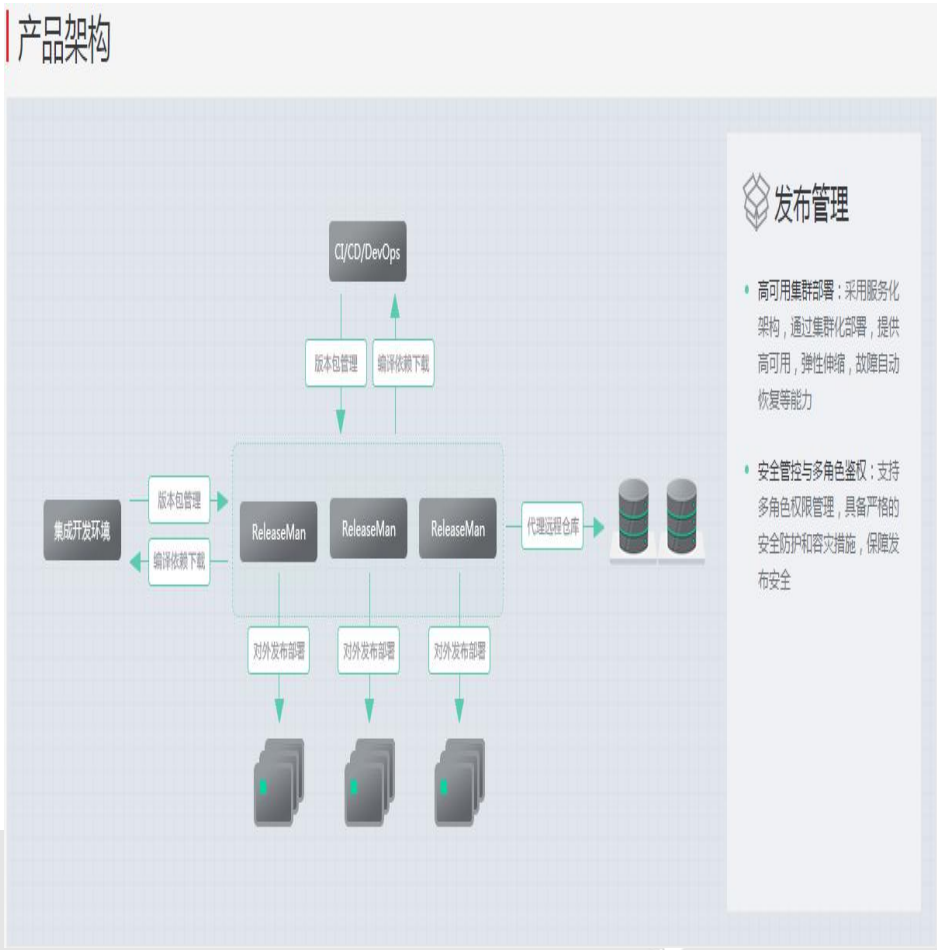
Devcloud如何进行发布管理

- 上个章节中介绍了ITIL和ISO/IEC 2000中对发布管理的定义、流程等内容，但是上述内容只是一个指导性的标准体系，较多的强调流程和管控。在正式的实践中往往需要根据具体的交付模式、组织来调整并通过工具来配合提升发布管理的效率。



Devcloud发布管理服务介绍

- **发布管理**（ReleaseMan）是面向软件开发者提供软件发布管理的云服务，提供软件仓库、软件发布、发布包下载、发布包元数据管理等功能，通过安全可靠的软件仓库，实现软件包版本管理，提升发布质量和效率，实现产品的持续发布。
  http://www.hwclouds.com/product/releaseman.html



持续交付流水线

软件交付面临的挑战

- 互联网业务特点
  - 客户范围广、个体差异大
  - 个性化需求多、变化快

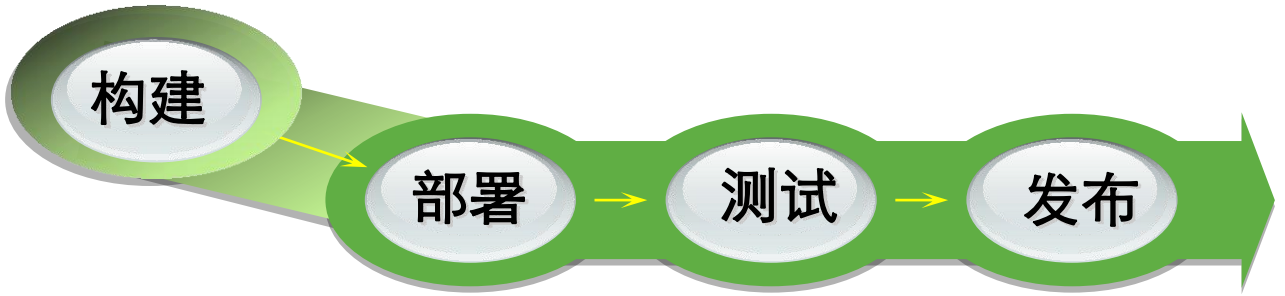
- 软件交付困难
  - 各环节能力没有自动化拉通，交付周期长
  - 问题不能及早发现，不断向后端积累，交付质量差；
  - 解决问题不能自动化验证，效率低
  - 研发过程缺乏一视角，沟通效率差，导致大量浪费和返工

路在何方?

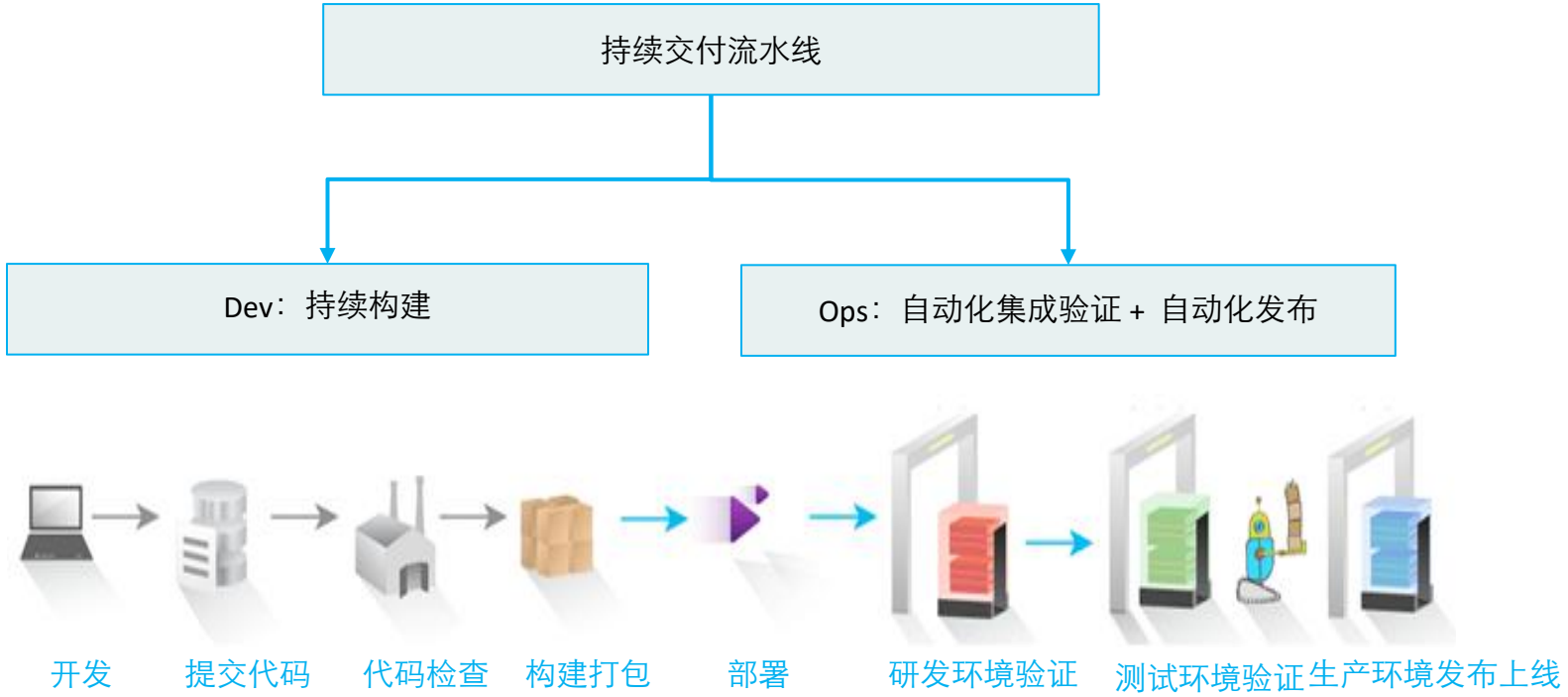
客户期望快速和频繁交付，软件企业和个人面临效率和质量双重挑战

持续交付

- **理念来源**于敏捷：“尽早持续交付有价值的软件并让客户满意”、“提前并频繁地做让你感到痛苦的事情（以降低风险）”。
- **目标**：产品团队具备通过重复、可靠的研发过程(自动化)，采取小批量频繁的部署或发布，尽可能早的获取质量反馈，使版本快速达到随时可交付状态。
- **主要措施**：“小批量/小粒度频繁的持续部署或发布”、“为软件的开发到发布创建一个可重复且可靠的自动化过程”、“每次修改都能经过一次构建、测试、部署、发布完整高效的自动化验证过程，实现高速频繁验证，快速问题闭环”。



持续交付流水线



打造持续交付流水线拉通Dev&Ops，保证研发过程高效、可靠、可重复，支撑企业小、频、快地交付价值特性

分层分级流水线

复杂产品或者服务，可以建立个人级、项目组级、版本级、解决方案级四级流水线，实现产品或服务的分层分级交付。每次新增/修改都经过分层闭环，触发构建、各层级&各环境部署、测试，通过高速频繁验证，实现快速定位问题与快速问题闭环

1、按照 分层夯实、分层快速闭环的原则，建立并完善 四层交付体系

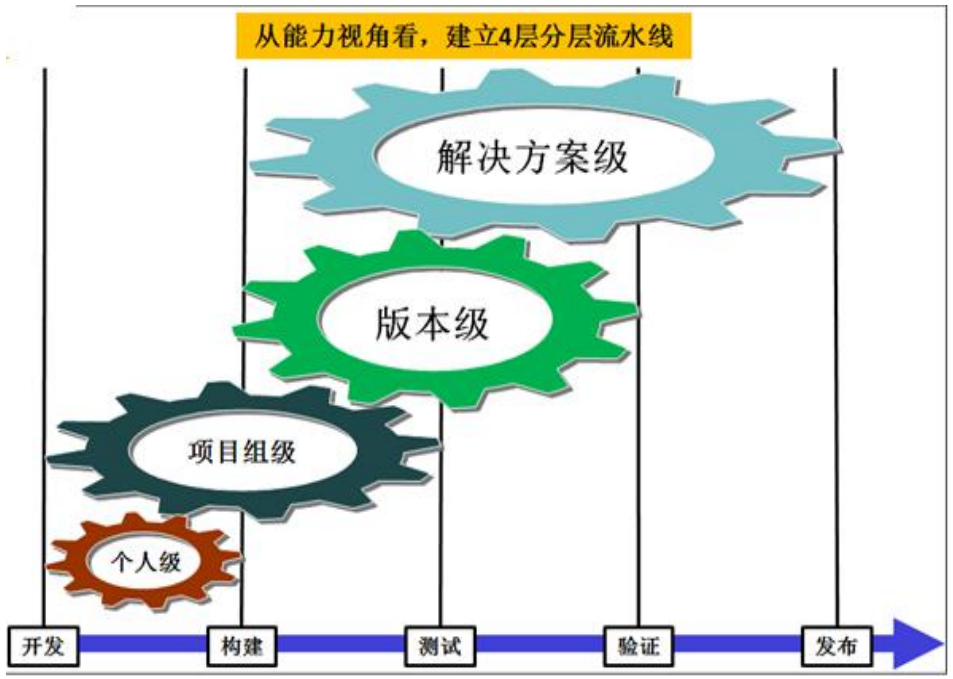
2、围绕四级分层交付，建立并完善开发与集成能力，包括：

1）技术上，设计解耦、开发与集成的依赖与顺序

2）管理上，项目管理（关注交付顺序、耦合/依赖关系）与版本开发分支管理

3）组织上，开发、测试、I&V围绕分层交付构建协同关系

3、围绕四级分层交付，建立并完善分层验证能力

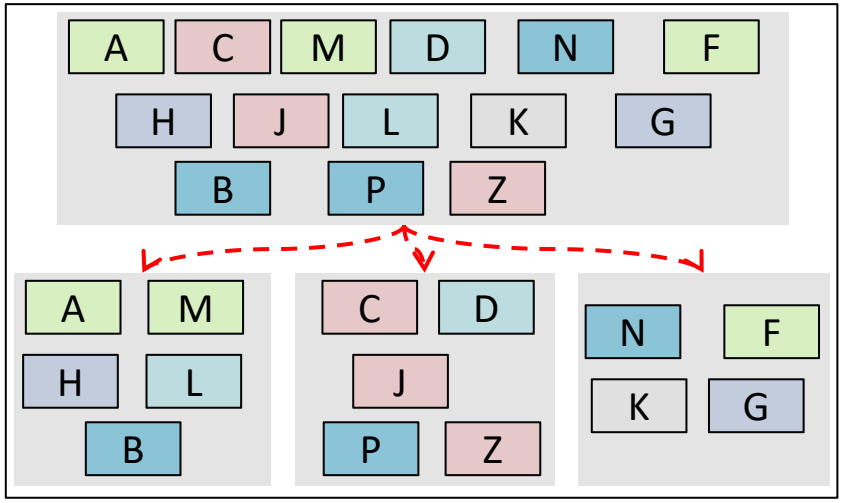


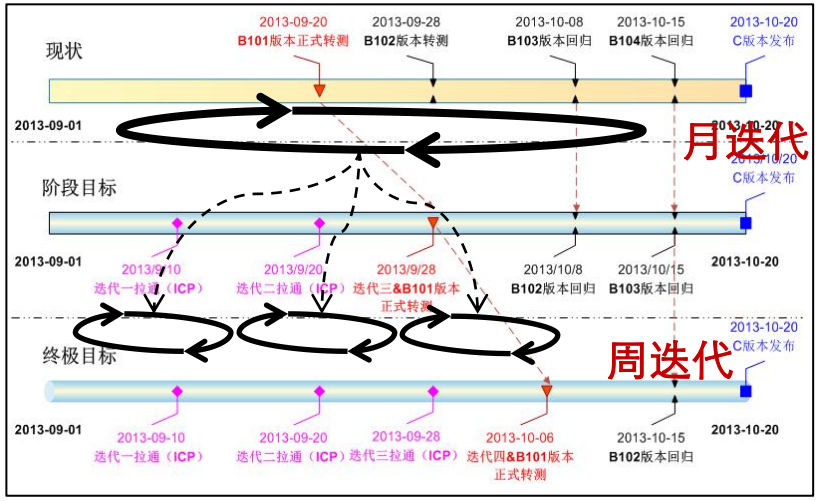
小迭代交付：

月迭代→“周/双周”迭代，提升交付频率和节奏，拉动四层循环高效运转

小迭代交付：

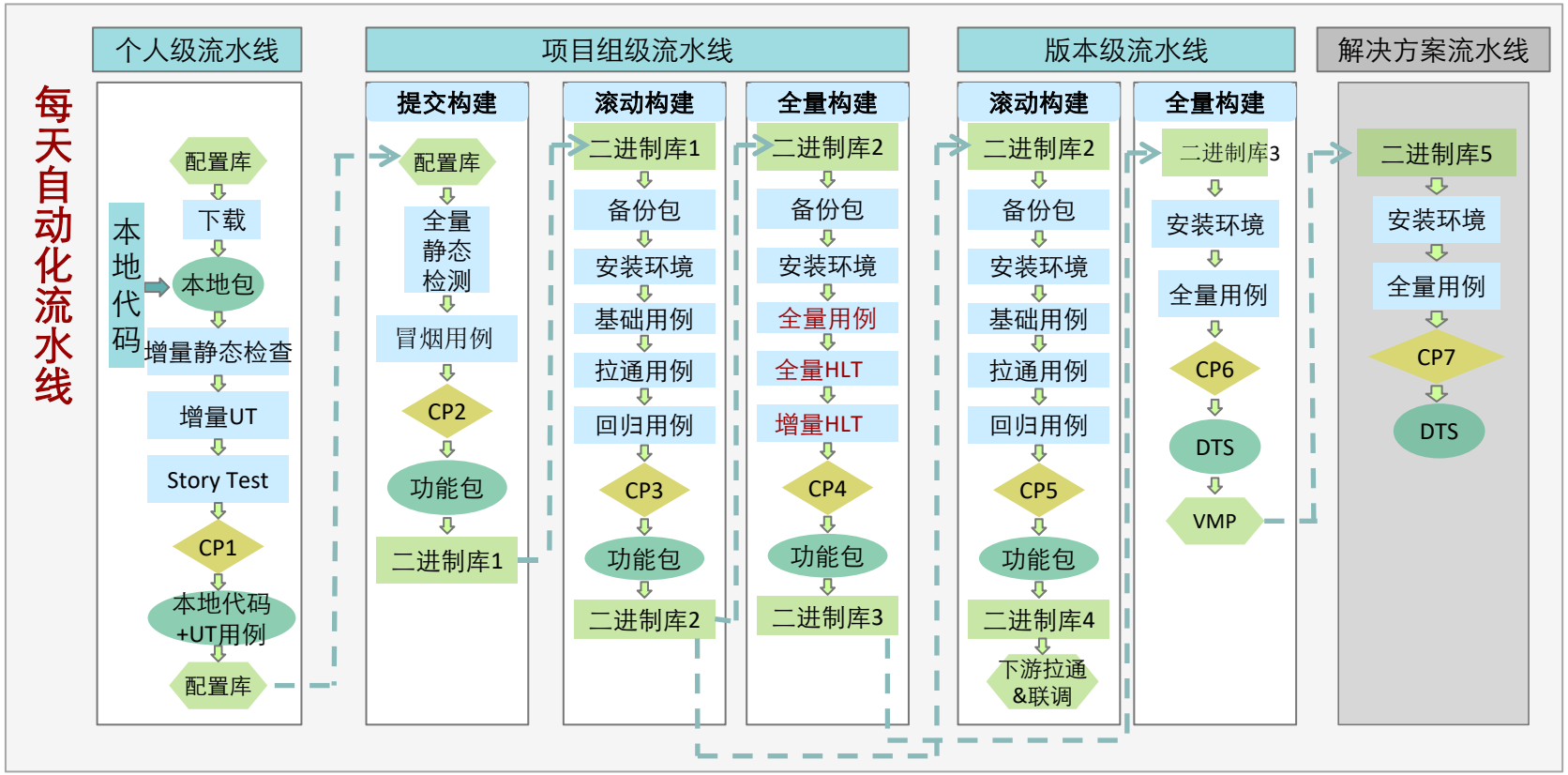
按小批量特性包/增量包（功能点、story）持续快速交付，实现小批量增量需求分析、设计、开发、精准快速验证，更容易暴露设计上的耦合和团队开发、测试间的协同问题





流水线自动化执行

自动化持续交付流水线：根据四层交付模型，建立代码到版本交付的自动化流水线，实现价值流高速流转



流水线可视化(1/2)

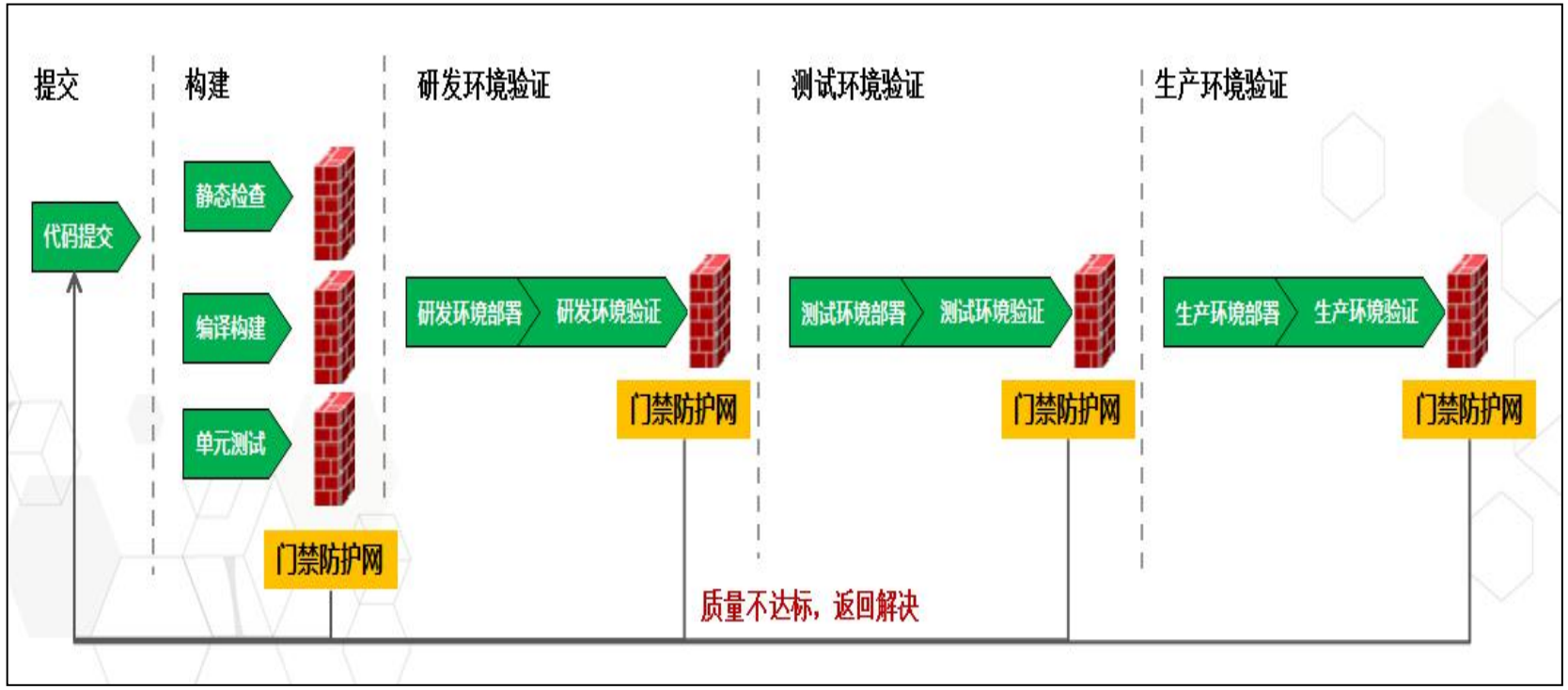
流水线运转可视化：自动化度量**质量、效率和CycleTime**，及时反应流水线运行状况，实现可视化自循环驱动

版本/子系统模块	CP3：项目级构建报表												
	得分	代码量(kloc)	CI构建成功率(%)	QDI值	Cyclic Dependencies	Data Clumps	Message Chains	Unnecessary Coupling	DT独立覆盖率(%)	平均圈复杂度	最大CC	代码重复率(%)	静态告警个数
PMS	92	153.94	100	133	0	6	0	0	52.11	4.71	15	9.48	0
SMS	95	63.54	100	98	0	0	0	0	67.48	4.42	15	6.71	0
PARTNER	89	53.11	100	108	0	6	0	0	43.68	3.74	15	6.87	0
CMS	96	104.81	100	129	0	227	0	0	71.43	4.11	15	8.78	0
Provision	82	107.65	100	61	0	0	0	0	15.4	4.4	15	9.27	0
IB	30	238.78	100	60	0	0	0	0	0	NA	NA	NA	0
CGW	52	114.08	75.86	73	0	0	1	0	0	4.16	31	28.06	0
SMPA	43	43.42	50	94	0	0	0	0	0	3.15	12	7.54	0
UserProfile	75	194.27	68.13	60	3	30	0	0	37.25	5.81	22	13.46	3
Charging	59	217.37	92.68	119	0	53	0	0	0	12.15	86	9.82	0
HDM	80	6.87	100	58	0	0	0	0	0	3.52	11	1	0
Payment	79	37.27	37.5	28	0	0	0	0	65.93	3.43	14	8.25	0
DMS	77	43.62	38.46	140	0	4	0	0	59.48	4.4	15	9.27	0

流水线可视化(2/2)

版本/子系统模块	CP5：版本级构建									
	得分	代码量(kloc)	版本每日可用率	CI构建成功率(%)	平均修复时长(小时)	CI平均构建时长	SDV自动化用例密度(个/KLOC)	HLT全量覆盖率(%)	HLT增量覆盖率(%)	CI构建频度(次/天)
XMS_UPDATE	94	281.3		100	0	10.43	45.82	46.75	54.98	0.43
IB	88	238.78		42.86	15.5	6.64	22.34	0	0	1
CGW	94	114.08		75	24	7.44	31.35	0	0	1.14
SMPA	88	43.42		50	66	6.61	29.23	0	0	0.57
UserProfile	97	198.38		100	0	8.52	33.17	70.27	70.27	0.86
Charging	100	236.68		100	0	7.69	40.59	0	0	1
Payment	100	29.54		100	0	1.45	34.77	79.4	79.72	1

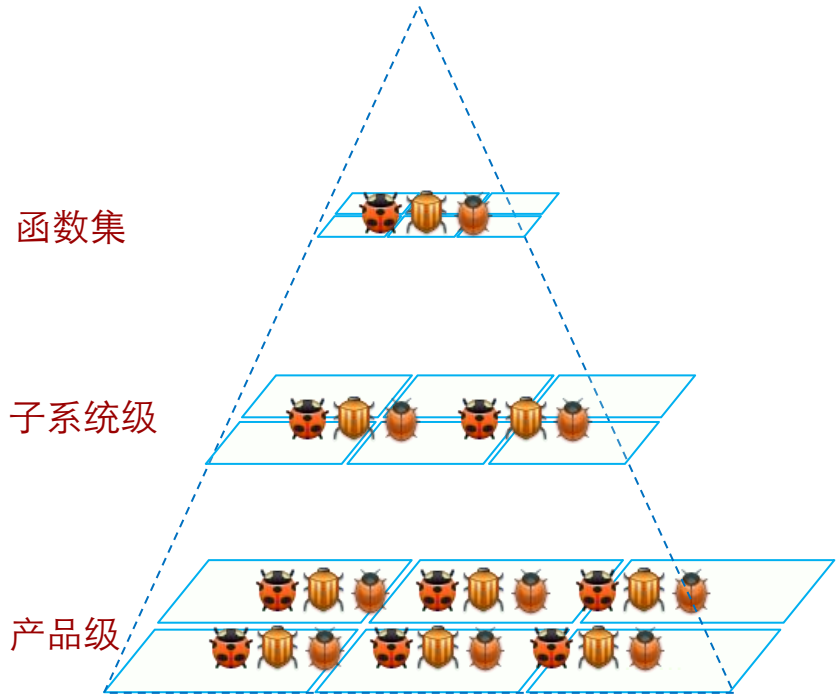
分层质量防护



利用流水线智能门禁功能，分层分级设置质量门禁，当质量不达标时及时终止交付过程。快速发现问题，提升交付质量

缺陷发现前移

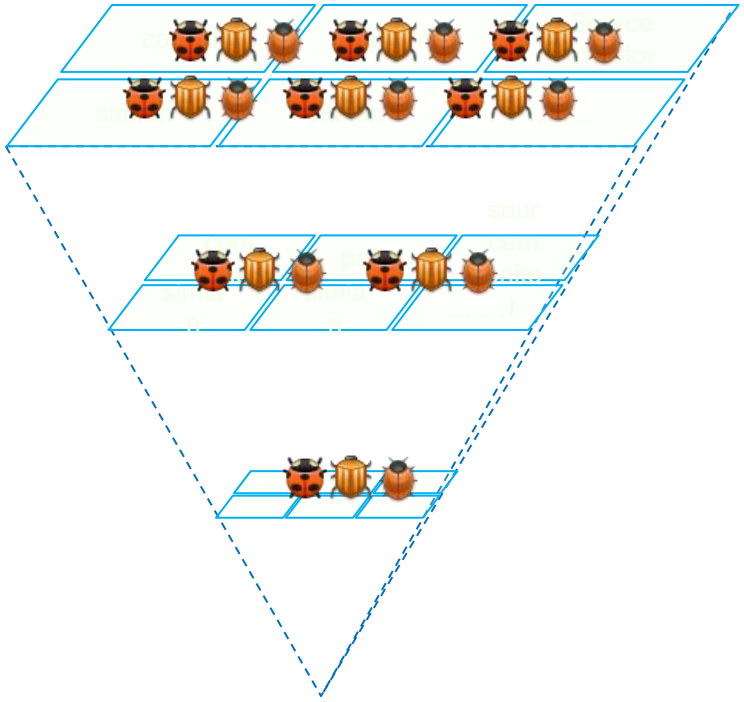
Before



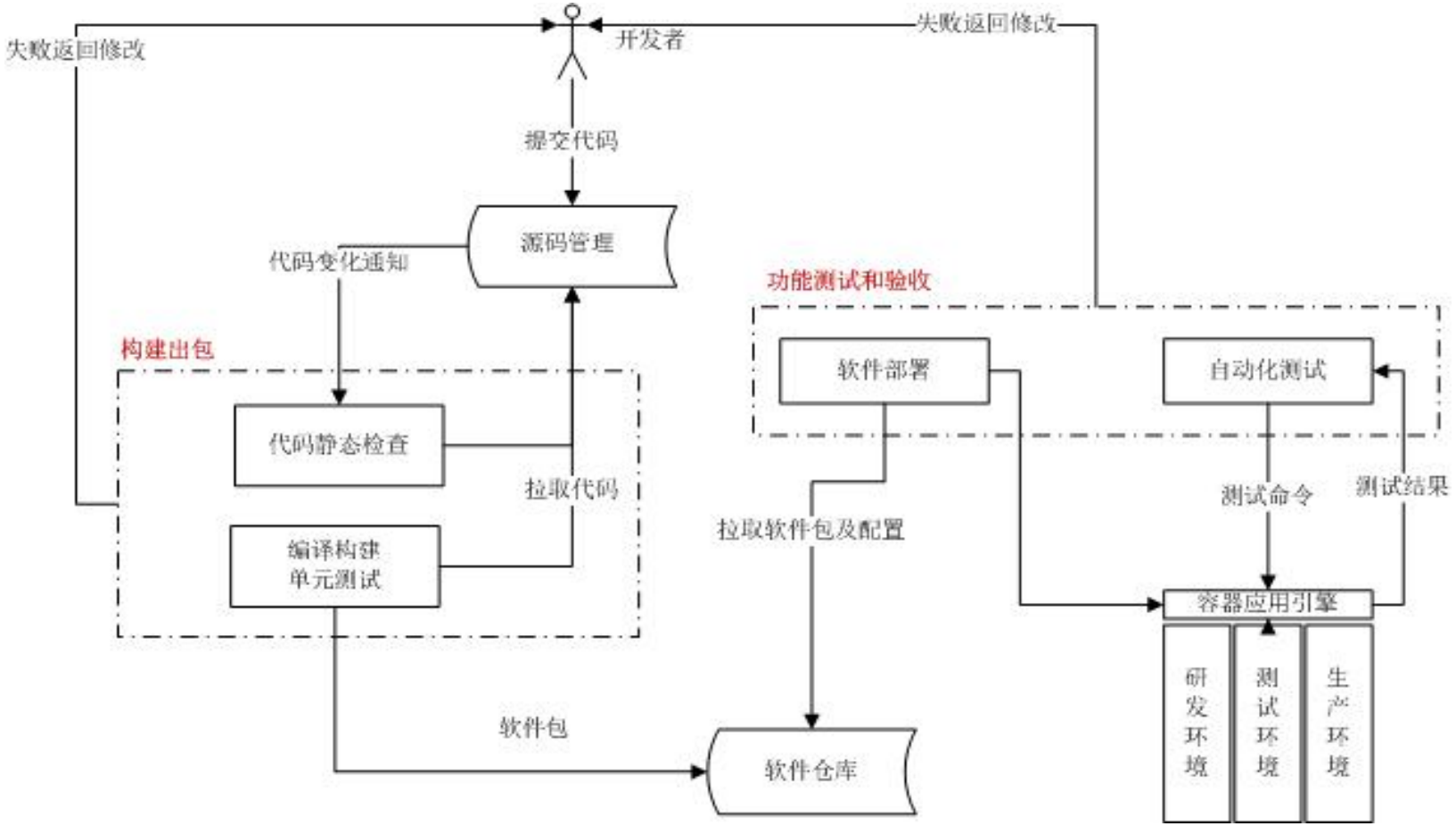
大量问题累积到问题后端发现

Now

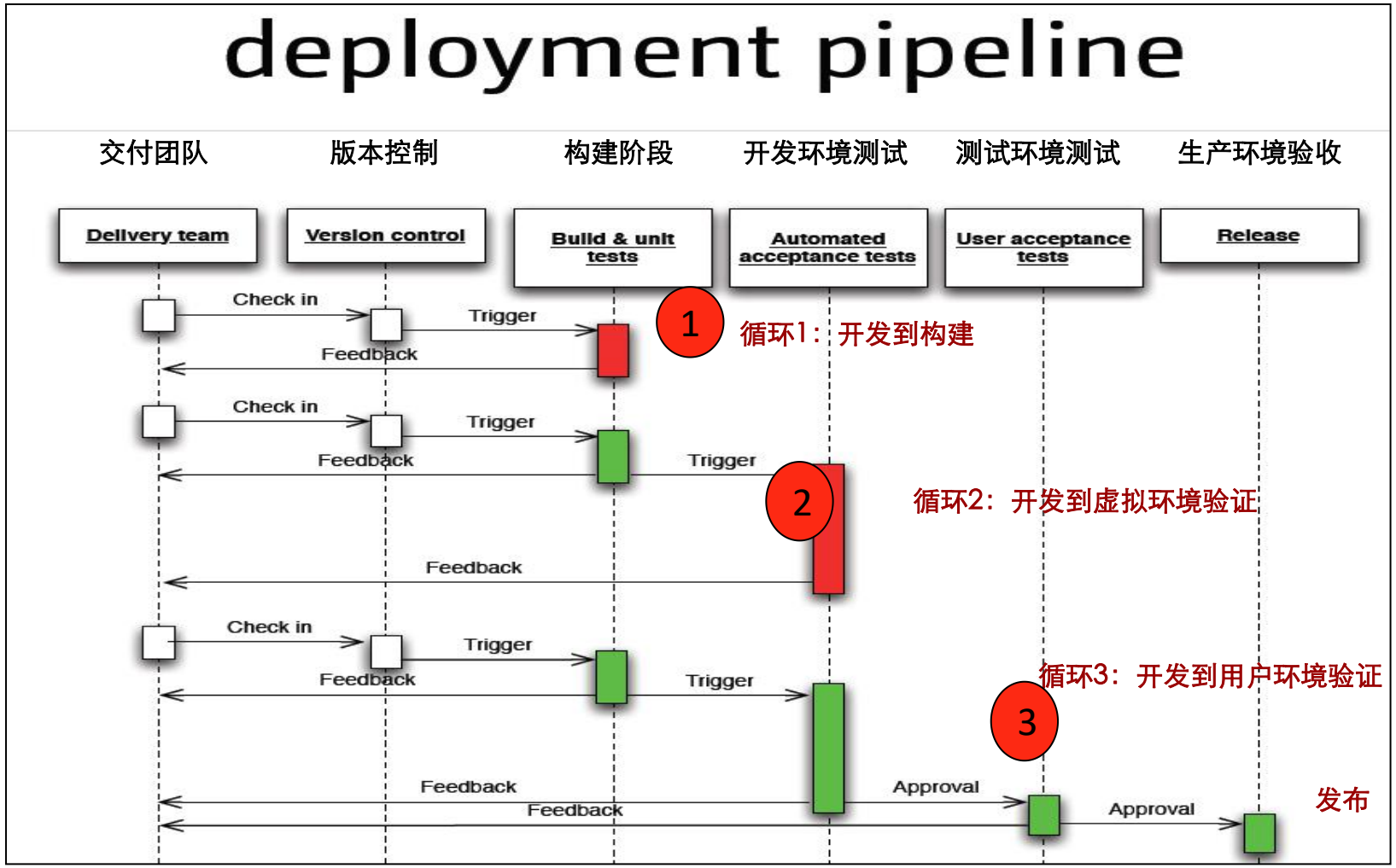
问题前端提前发现



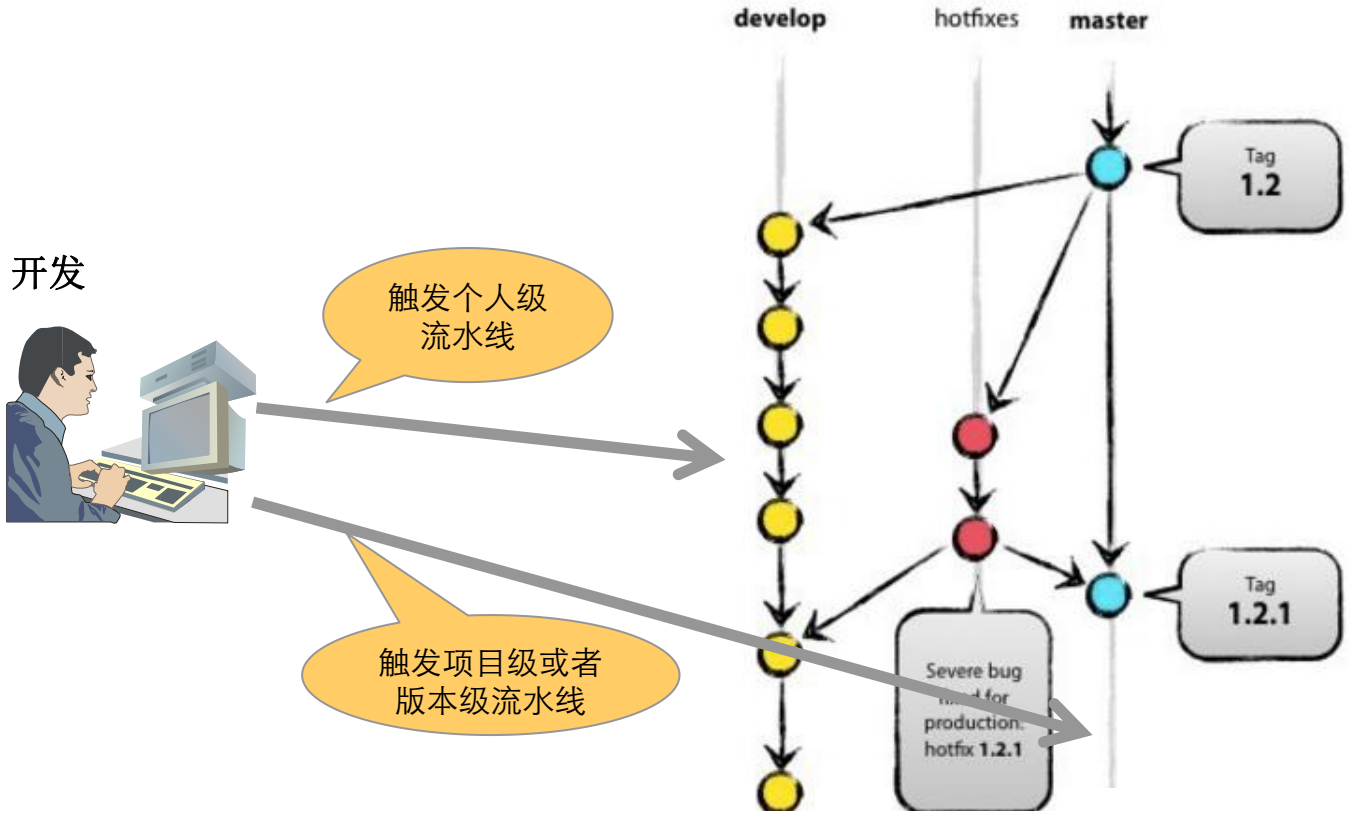
典型持续交付场景


失败返回修改
开发者
失败返回修改
提交代码
源码管理
代码变化通知
构建出包
代码静态检查
拉取代码
编译构建
单元测试
功能测试和验收
软件部署
自动化测试
拉取软件包及配置
测试命令
测试结果
容器应用引擎
研发环境
测试环境
生产环境
软件包
软件仓库

典型持续交付流水线


deployment pipeline
交付团队
版本控制
构建阶段
开发环境测试
测试环境测试
生产环境验收
Delivery team
Version control
Build & unit tests
Automated acceptance tests
User acceptance tests
Release
Check in
Trigger
Feedback
1
循环1：开发到构建
Check in
Trigger
Feedback
Trigger
2
循环2：开发到虚拟环境验证
Feedback
Check in
Trigger
Feedback
Trigger
循环3：开发到用户环境验证
3
Feedback
Approval
Feedback
Approval
发布

代码提交



开发团队提交最新代码到版本库，手动或者自动触发交付流水线

构建阶段 – 代码检查

- 构建阶段进行的是代码静态检查：在不运行计算机程序的条件下，进行程序分析的方法。大部分的静态程序分析的对象是针对特定版本的源代码，也有些静态程序分析的对象是目标代码。
- 检查的范围包括：潜在的bug、可优化的代码、安全性、性能、可用性、可访问性
- 持续交付流水线构建阶段，并设置代码检查质量门禁

构建阶段 – 构建

- 软件构建是指把软件源码编译成目标文件，并将目标文件和必要的文档制作成软件包并上传的过程，一般包含：从代码仓库拉取源码、从软件仓库拉取依赖包、编译成目标文件、软件打包、上传软件包等步骤。

构建阶段 – 单元测试

- 单元测试（unit testing），属于白盒测试，是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如C语言中单元指一个函数，Java里单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总的来说，单元就是人为规定的最小的被测功能模块。单元测试是在软件开发过程中要进行的最低级别的测试活动，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。
- 构建阶段，可选择进行单元测试，并设置单元测试质量门禁。

构建阶段 – 归档

- 软件包构建构建成功后，如果通过了代码质量检查、单元测试，就可以把包归档到版本仓库中，供后续进行进一步测试和验证。在仓库中，一般包含包名、版本号、其他版本信息等。

开发环境阶段 – 自动化部署

- 从软件仓库拉取构建的软件包，并自动部署到开发环境和自动修改相应配置信息，使开发环境做好准备，进行相应的开发自验证。
- 自动化部署具有如下特征：

  一键式部署：尽可能的自动化所有部署过程，包括基础设施的创建和部署

  多环境支撑：能够适应于开发、测试和生产环境

  支持高并发部署：同时部署到多个环境和多台主机

  无服务中断：能够无缝的进行服务升级、切换

  支持回滚： 可以很容易的回滚到前面的版本以处理意外问题

开发环境阶段 – 测试验证

- 开发环境进行子系统接口测试、功能测试
- 子系统接口测试，属于白盒测试，需要深入了解产品和服务内部，按照接口的定义，进行子系统功能正确性的验证。最好能够进行自动化测试。
- 功能测试，属于黑盒测试，一般根据系统规格说明书，对系统实现的功能正确性进行验证。本阶段功能测试用例由开发写作，站在开发的角度对系统的功能进行初步验证，至少保证系统的基本功能正确。

测试环境阶段 – 部署

从软件仓库拉取经过开发自测试的软件包，并自动部署到测试环境和自动修改相应配置信息，使测试环境做好准备，为系统在测试环境的验证做好准备。

测试环境阶段 – 测试验证

- 测试环境验证属于黑盒测试，包括功能测试、性能测试、压力测试、兼容性测试等。
- 功能测试同开发环境测试，只是用例由测试人员写作，站在不同的角度，加强对异常场景等场景的测试。
- 性能测试，性能测试为系统增加工作负载，收集性能参数，测试响应性和稳定性。
- 压力测试，通过逐步增加系统负载，确定系统失效的临界值，以此来获得系统能提供的最大服务级别。
- 兼容性测试，检查系统和硬件、其他软件、不同浏览器、数据库、中间件等的兼容性。

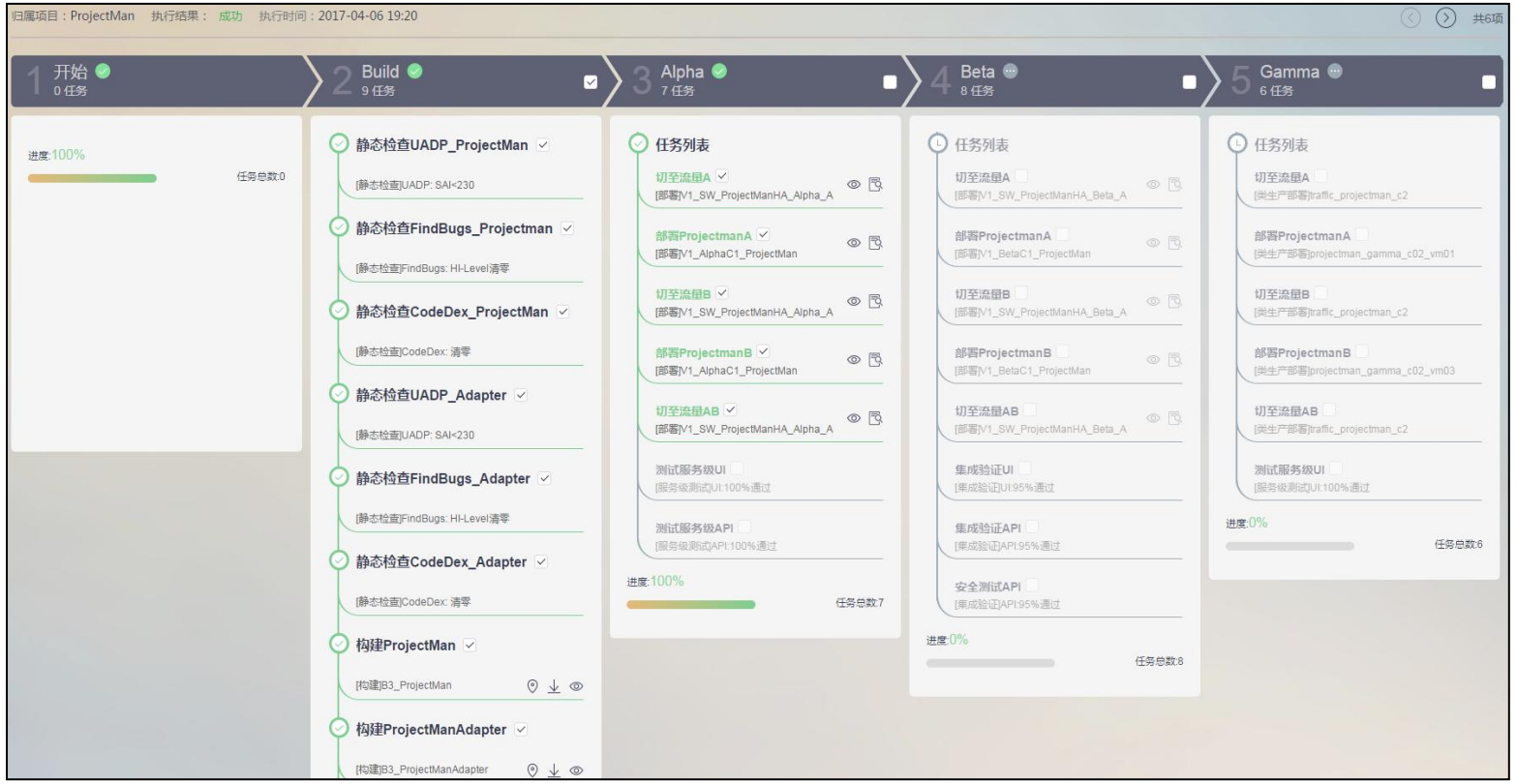
生产环境阶段 – 部署

从软件仓库拉取经过开发自测试的软件包，并自动部署到生产环境和自动修改相应配置信息，为了保证出错时对用户的影响减少到最小，一般会采取蓝绿部署等灰度发布方式。

生产环境阶段 – 测试验收

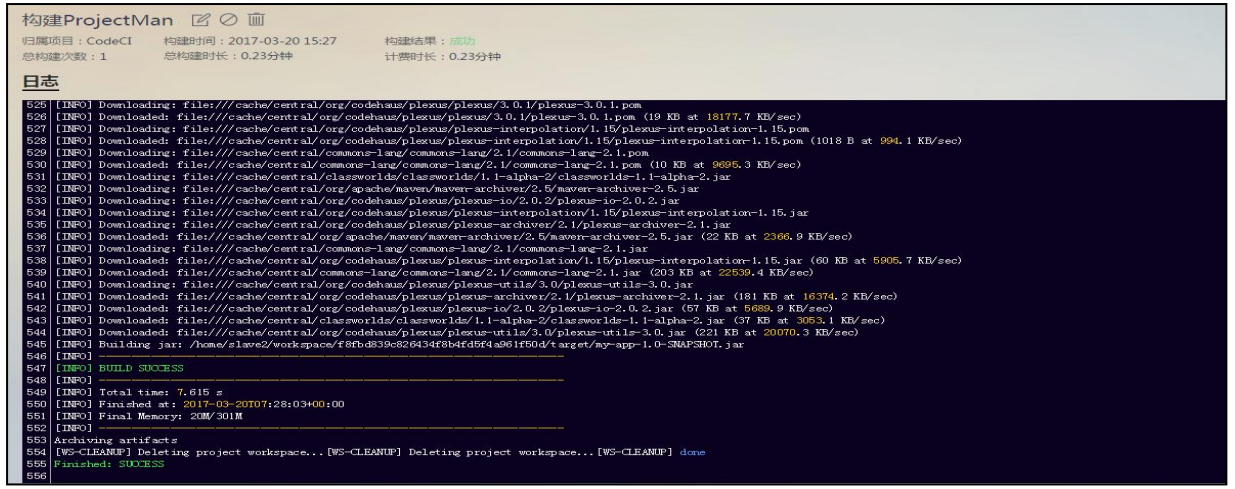
生产环境是站在用户角度进行的验收，属于黑盒测试，一般不进行功能的全量测试，而是根据用户实际使用场景进行特性的验收。测试用例一般由产品经理和业务分析师输出。

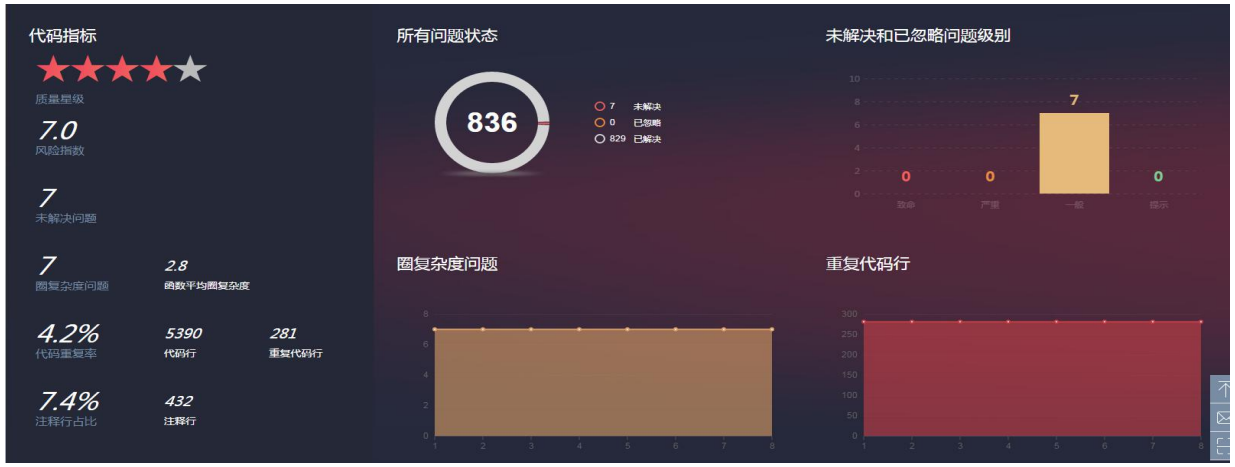
服务流水线



服务流水线包括了构建、研发环境、测试环境和类生产环境等阶段。构建阶段包含代码检查、构建；研发环境、测试环境、类生产环境包含部署及自动化测试等任务。

构建阶段代码检查和构建报告





研发环境部署和测试报告