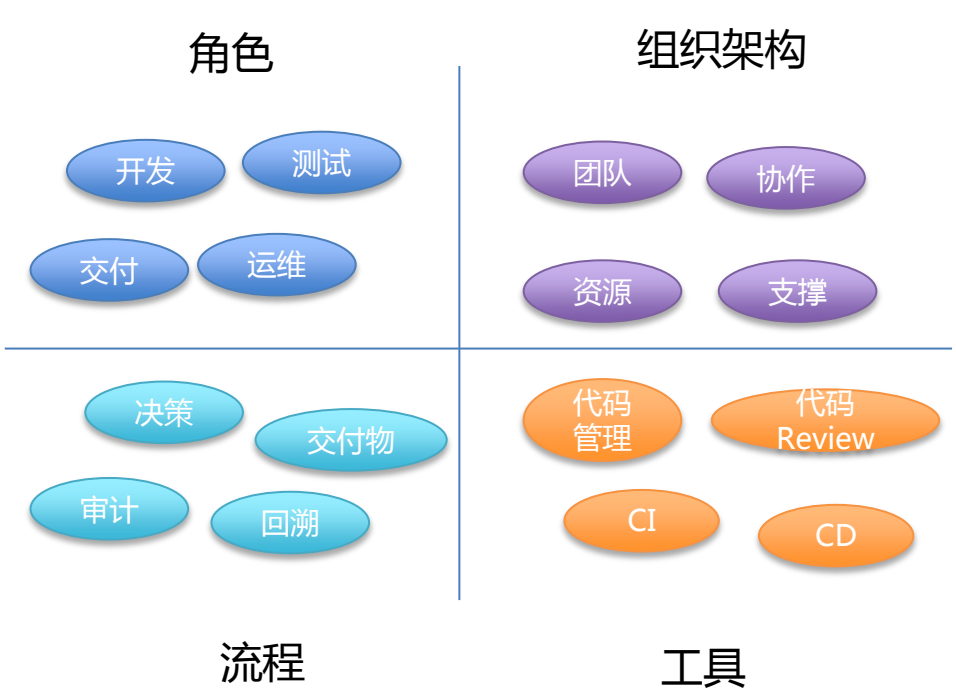
基于Docker+CI/CD的DevOps实践经验分享

博云CTO 李亚琼
Email：
liyaqiong@beyondcent.com

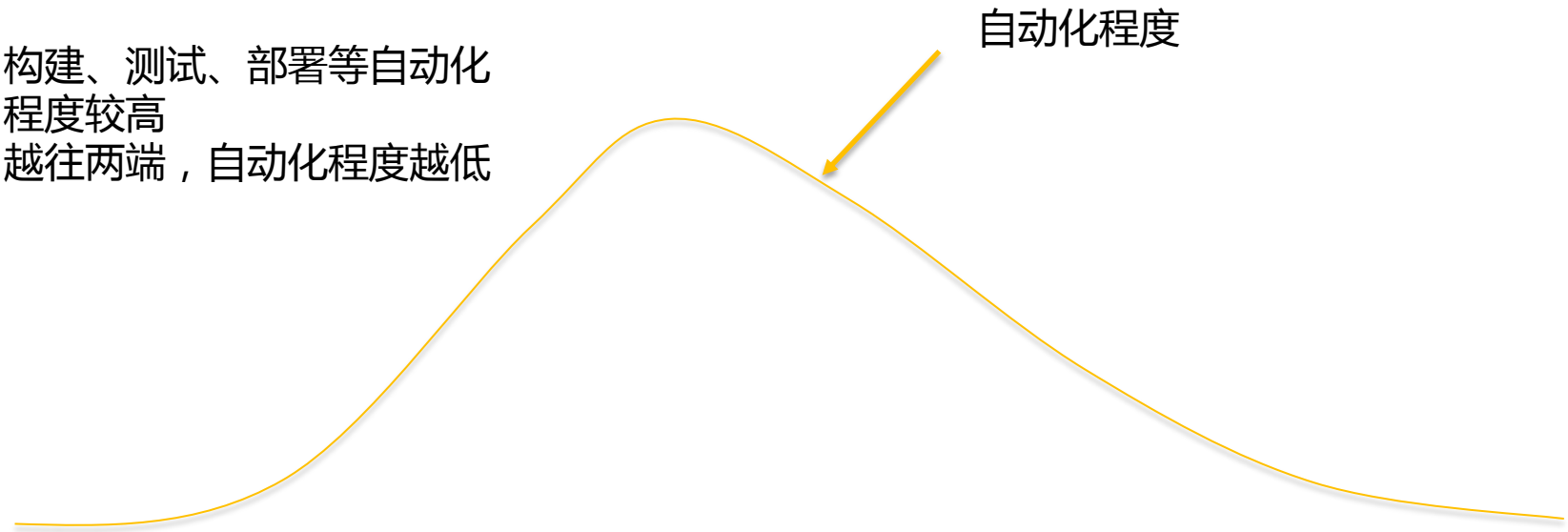
DevOps落地困境



- 涉及部门多
- 流程改造复杂
- 责任边界需要重新划分
- 考核等配套机制没有跟上
- 技术成熟度低

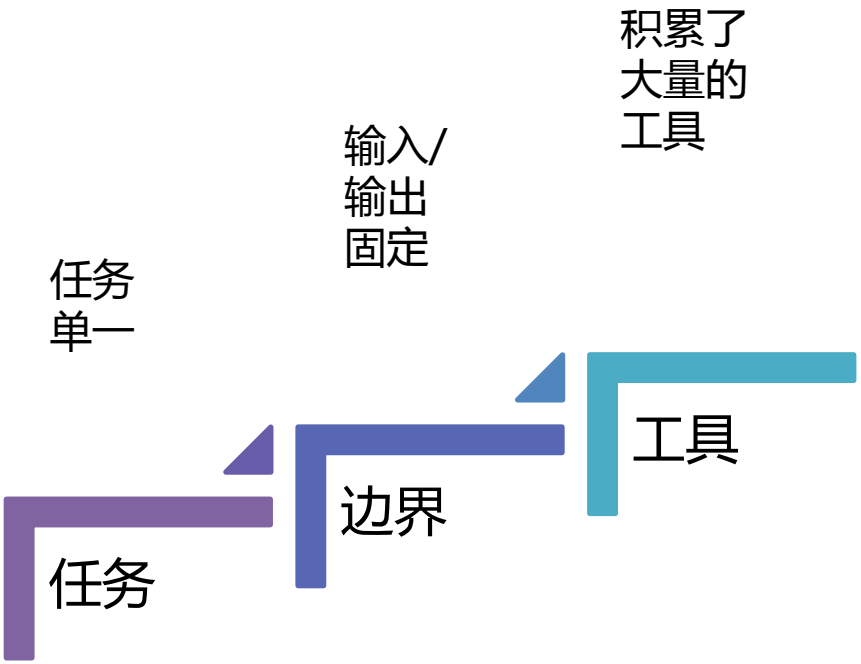
流程分析

- 构建、测试、部署等自动化程度较高
- 越往两端，自动化程度越低



流程动作

为什么CI/CD能够自动化？



构建工具



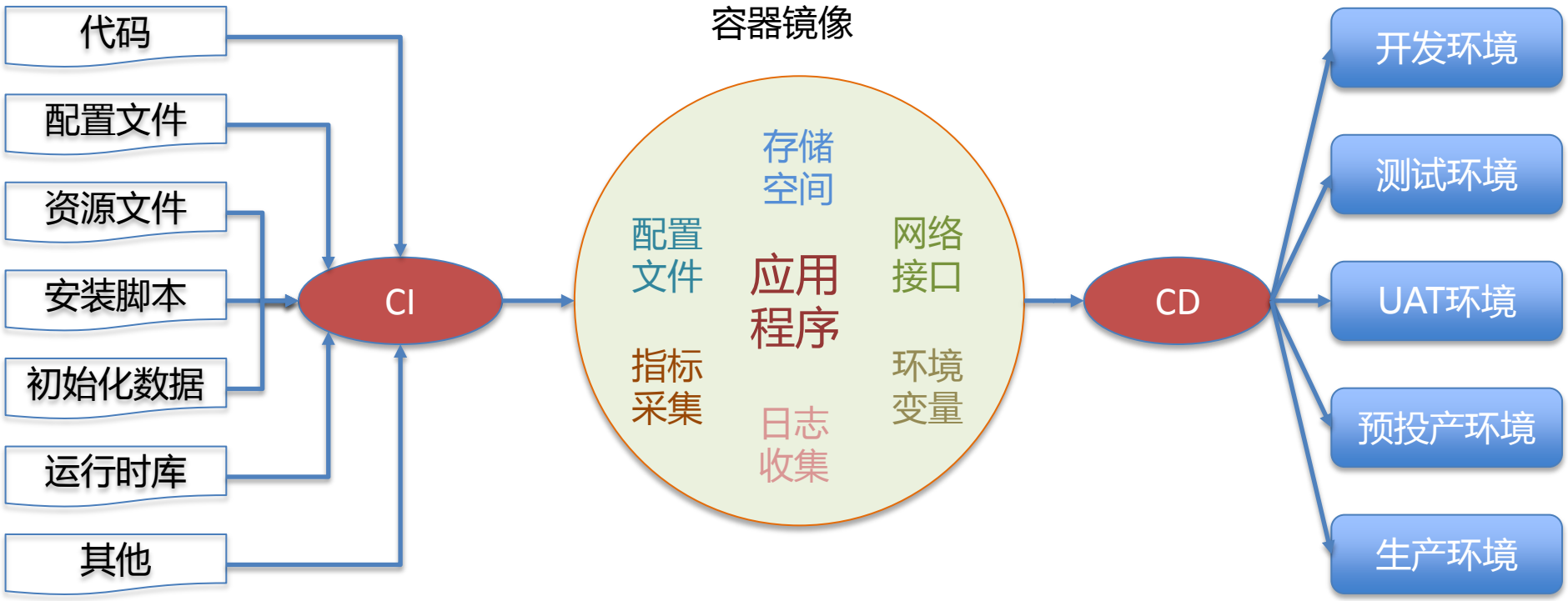
包管理器

自动化测试

配置自动化

Docker环境下的CI/CD愿景

一次构建，多次部署



Docker+CI/CD的现实困境

环境问题

- 容器环境建设
- 平台支撑能力对接
- CI/CD平台对接
- 多环境对接

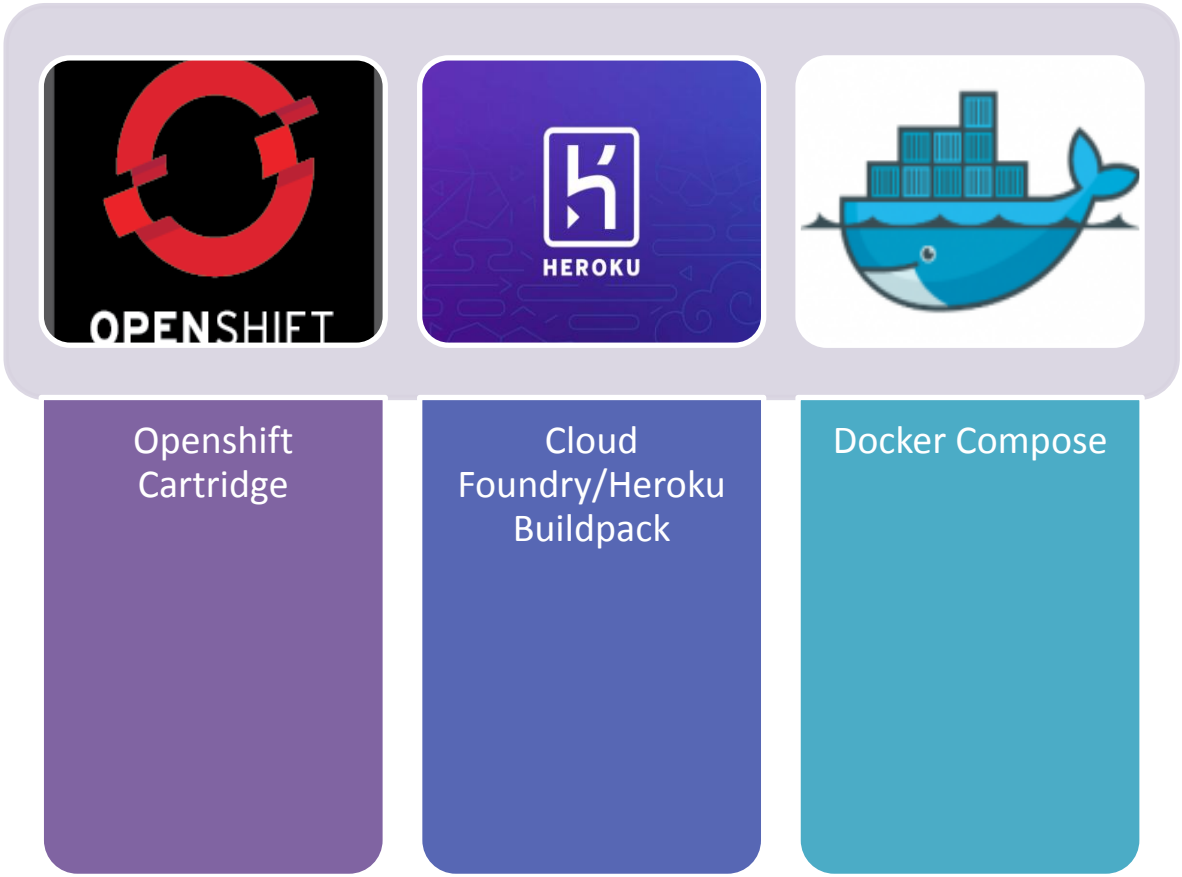
应用挑战

- 多语言适配
- 应用组件依赖
- 配置动态性
- 框架约定挑战

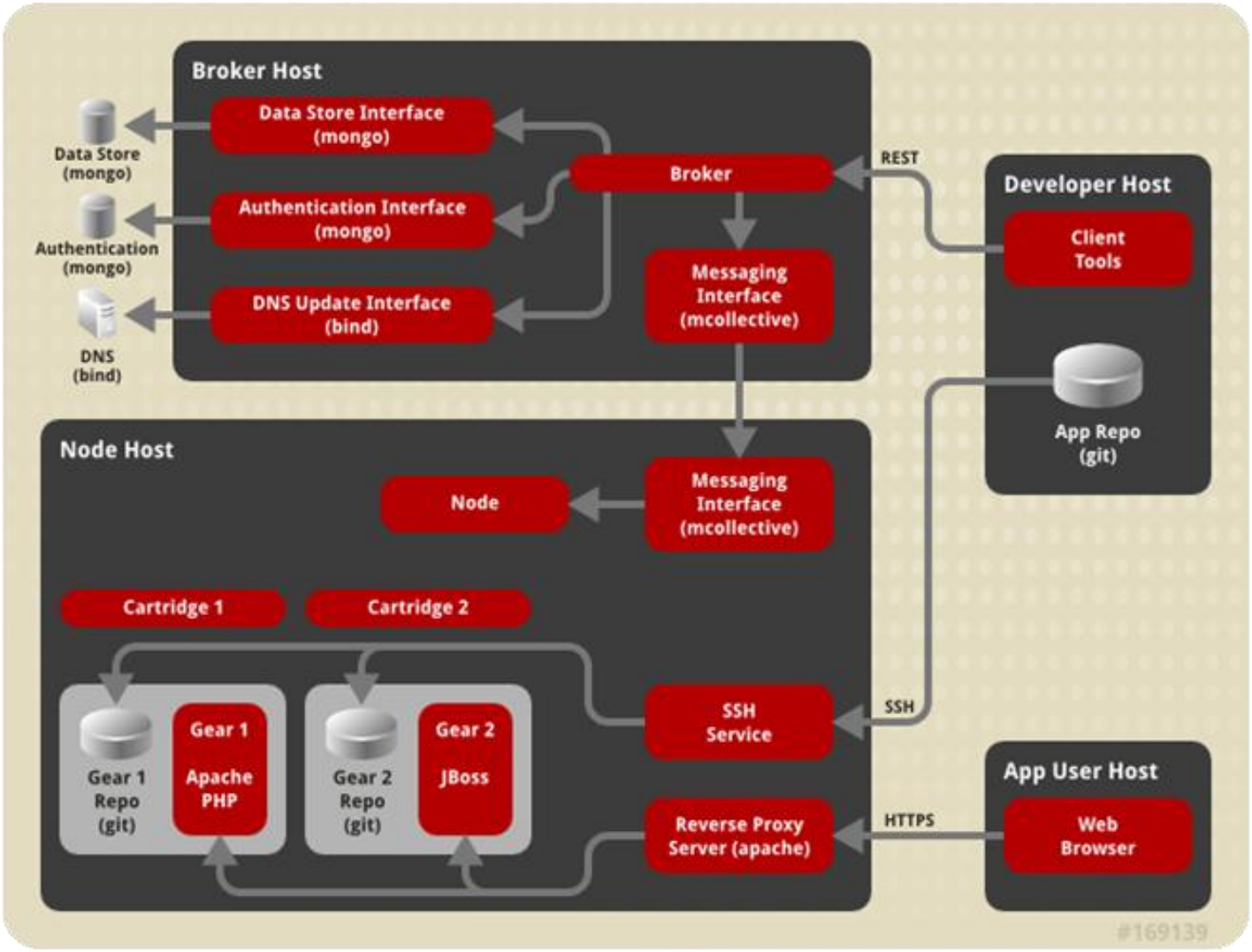
合规性约束

- 多角色功能划分
- 流程合规性

PaaS软件/平台中的CI/CD技术



Openshift v2



三个重要概念：

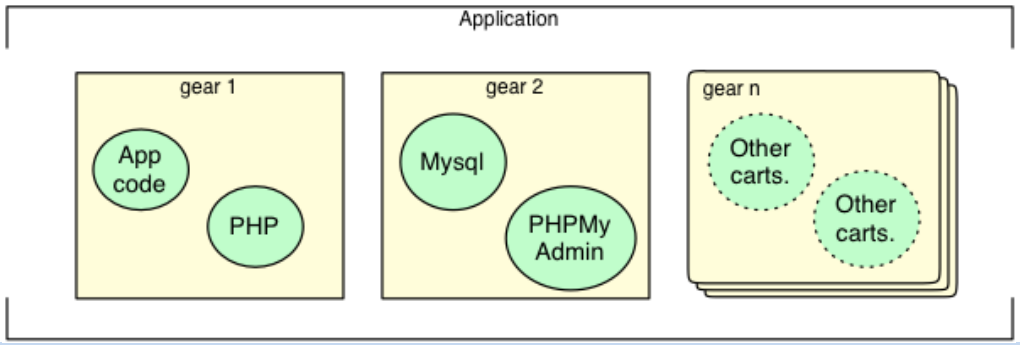
Broker——管理大脑，控制整个环境和应用的构建、配置等

Gear——应用运行的容器，约束着CPU、内存、存储等软硬件资源；并负责运行应用。

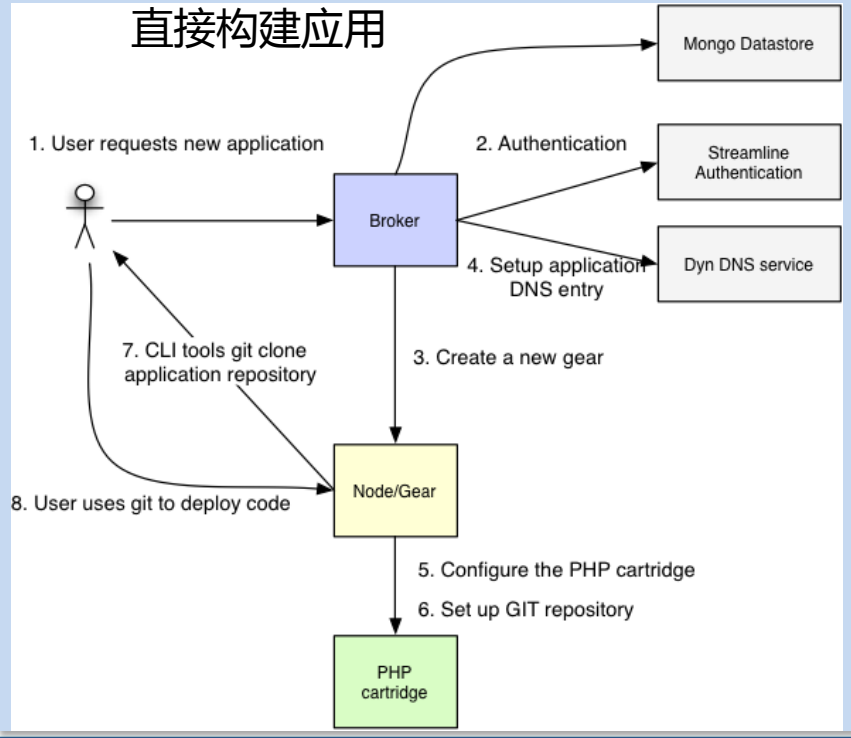
Cartridge——技术推栈，如语言、框架、服务，或者常用被打包的常用功能。

Openshift Cartridge

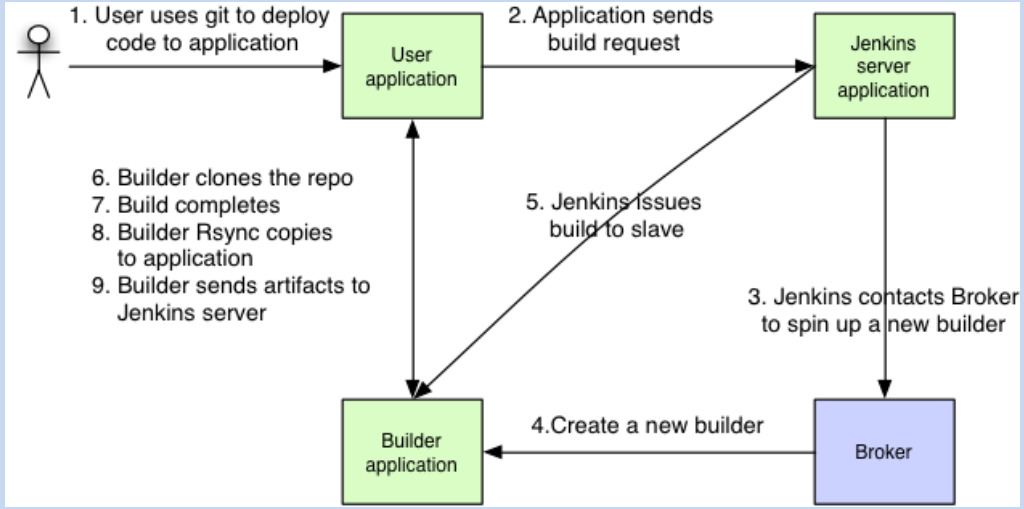
应用



直接构建应用

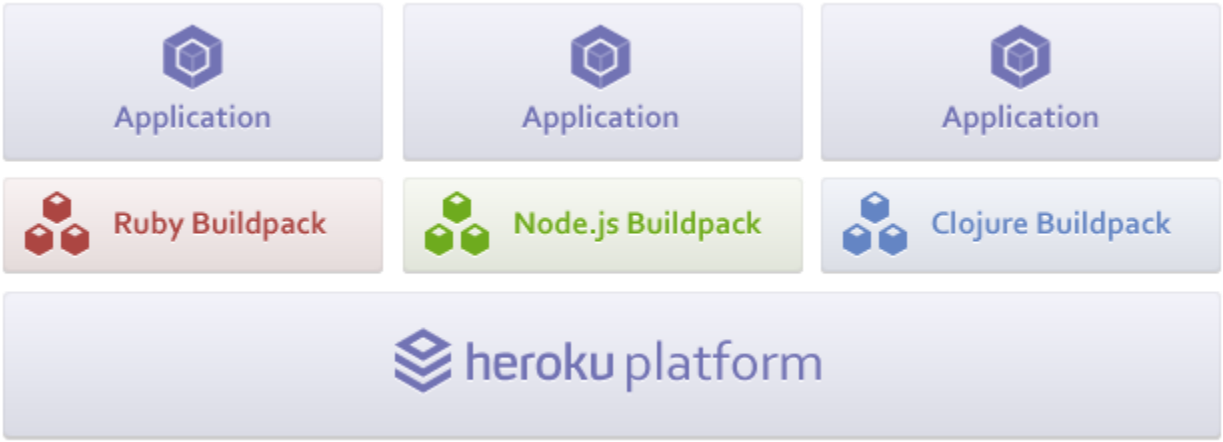


基于Jenkins构建应用



Cloud Foundry/Heroku buildpack

Buildpack——are responsible for transforming deployed code into a slug, which can then be executed on a dyno.



composed of a set of scripts, and depending on the programming language, the scripts will retrieve dependencies, output generated assets or compiled code, and more. This output is assembled into a slug by the slug compiler.

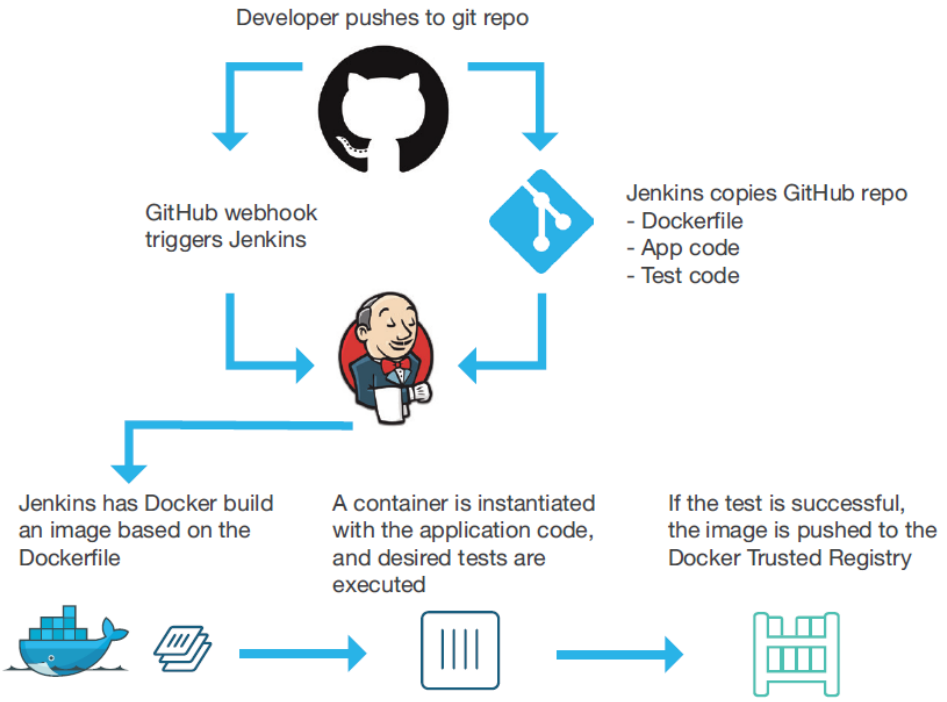
重要概念：

Buildpack：控制脚本，用于构建环境、编译代码、输出结果等。

Slug：可运行的软件包，由程序、环境、配置等构成。

Dyno：Slug的运行平台。

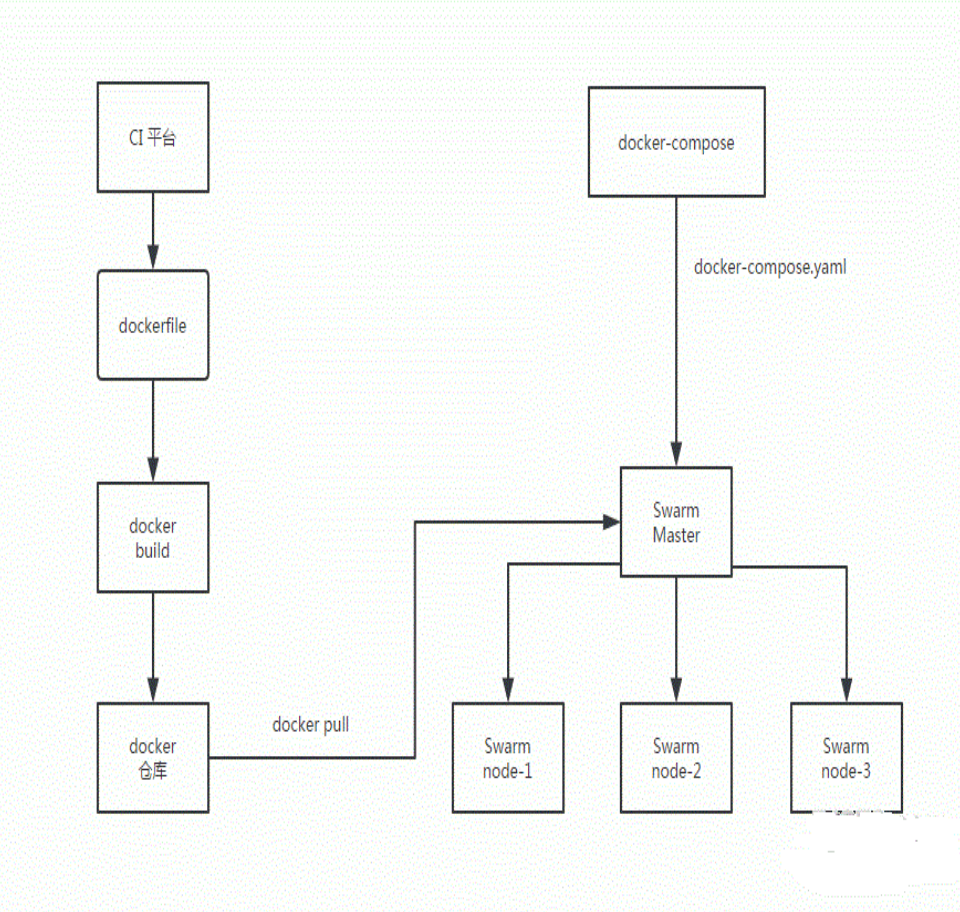
Docker CI



CI流程

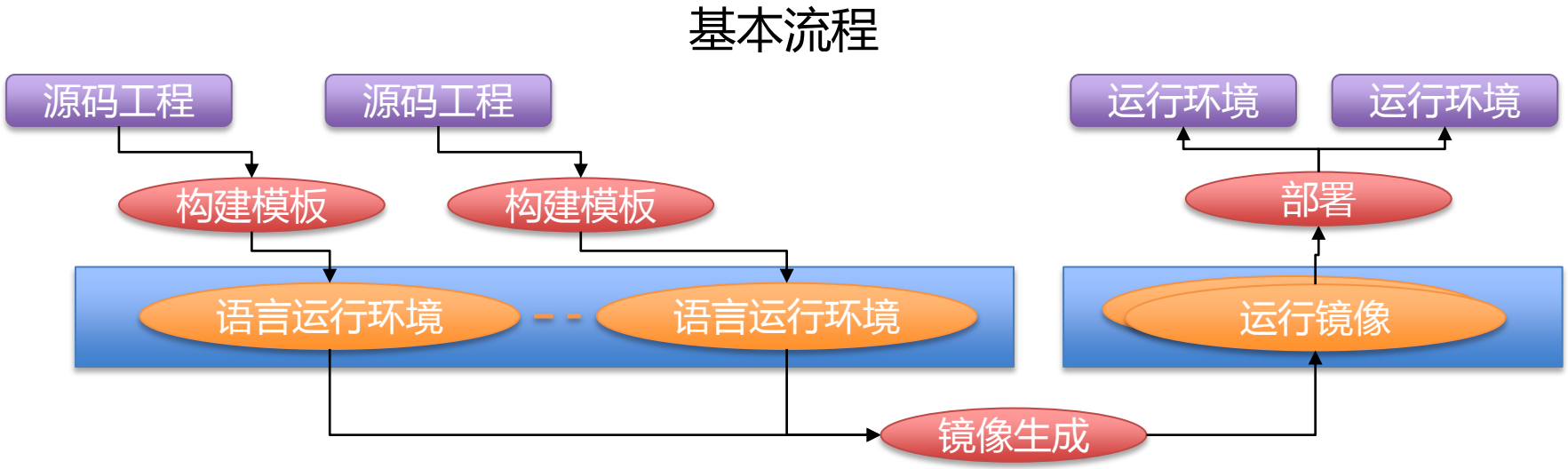
1. 开发者向Github提交代码
2. Github通过webhook通知Jenkins有更新
3. Jenkins从Github下拉最新代码、Dockfile及其他文档
4. Jenkins在Slave节点构建Docker镜像
5. Jenkins在Slave节点实例化Docker镜像，并且执行测试代码
6. 如果测试通过，Jenkins推送Docker镜像到仓库

Docker CD（Compose）



build(构建yml中某个服务的镜像)
ps(查看已经启动的服务状态）
kill(停止某个服务）
logs(可以查看某个服务的log）
port(打印绑定的public port）
pull(pull服务镜像)
up(启动yml定义的所有服务）
stop(停止yml中定义的所有服务）
start(启动被停止的yml中的所有服务）
kill(强行停止yml中定义的所有服务）
rm（删除yml中定义的所有服务）
restart(重启yml中定义的所有服务）
scale(扩展某个服务的个数，可以向上或向下）
version（查看compose的版本）

还有什么问题？



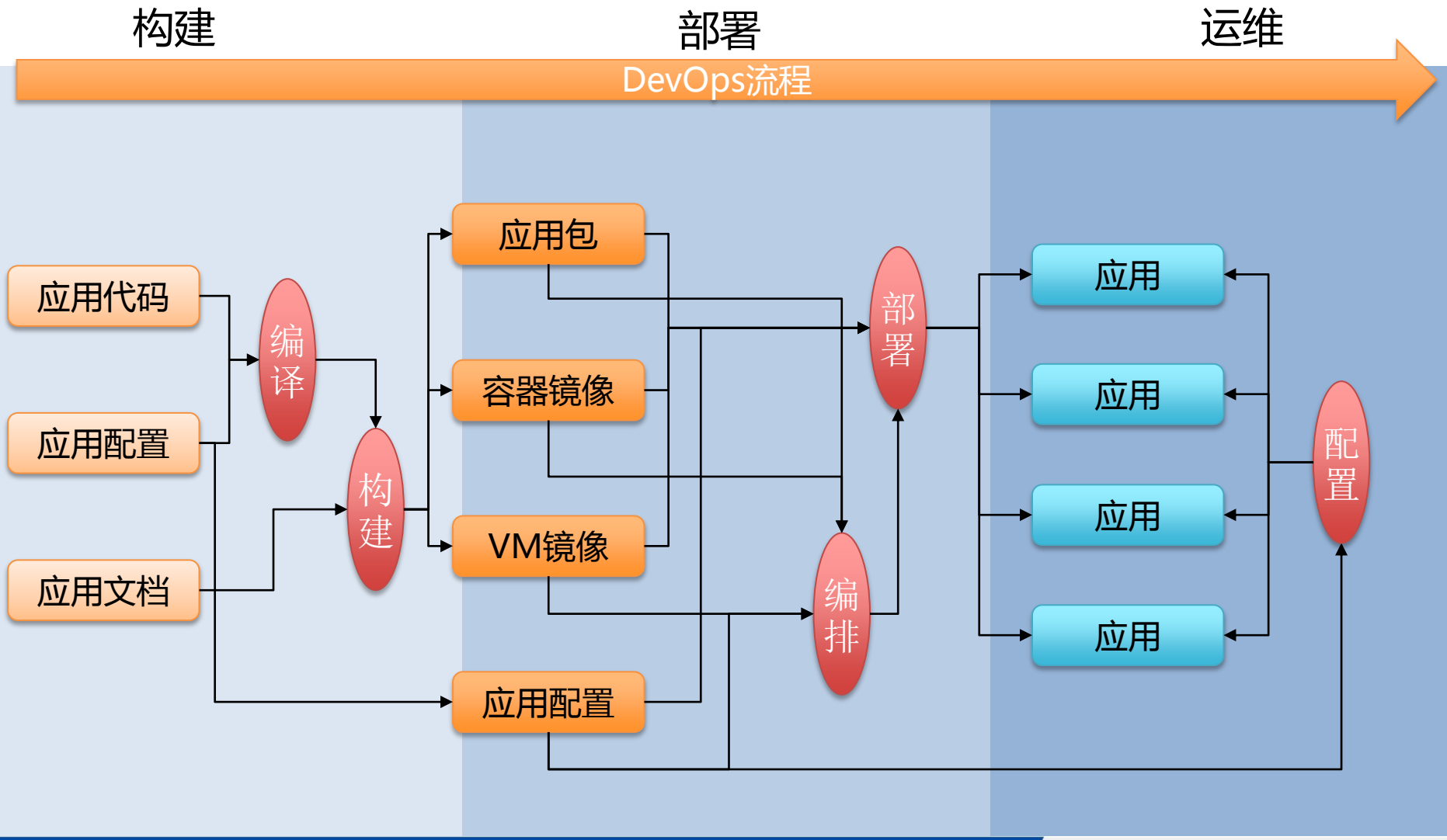
优点：

- 流程简单，易于上手
- 新应用友好
- 工具丰富，方便定制
- 方便二次分发

不足：

- 多语言应用难以支持
- 多组件（子系统）应用需要拆分手工配置
- 部署时配置能力差
- 跨环境应用需手工处理

开发运维流程


构建
部署
运维
DevOps流程
应用代码
应用配置
应用文档
编译
构建
应用包
容器镜像
VM镜像
应用配置
部署
编排
应用
应用
应用
应用
配置

关键动作

- **构建**：构建动作用来生成应用程序包、配置文件、安装/部署脚本等，维护程序包的版本。
- **编排**：根据应用系统的拓扑、应用依赖平台、软件包、配置文件（脚本）等，根据依赖关系进行架构编排、动作设置。
- **部署**：根据编排控制文件，进行实例化部署，完成环境设置、平台配置、软件安装、配置更新等动作。
- **配置**：定义不同阶段要进行的配置行为，解析配置模板，自动进行依赖解析，完成应用系统配置。

建设云环境下的CI/CD遵循的原则

原则1：

尽量不破坏现有的流程和责任边界，基于现有交付物进行CI/CD平台设计

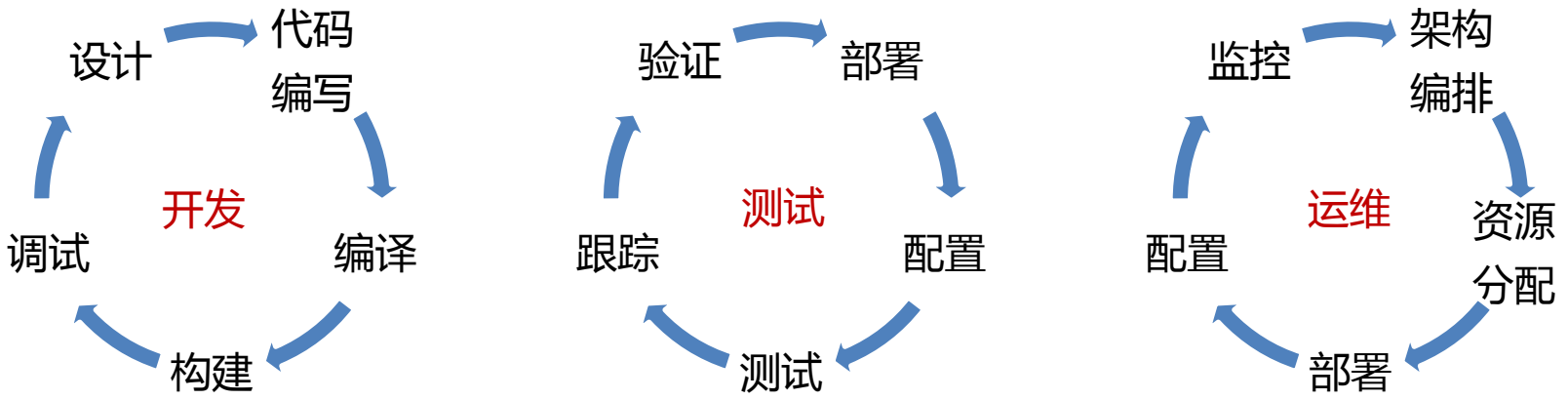
原则2：

尽量兼容现有应用系统架构，对架构不做颠覆式破坏或约束

原则3：

不增加开发、测试、运维等角色的学习成本，而是通过自动化提升其工作效率

角色/流程/关键工具

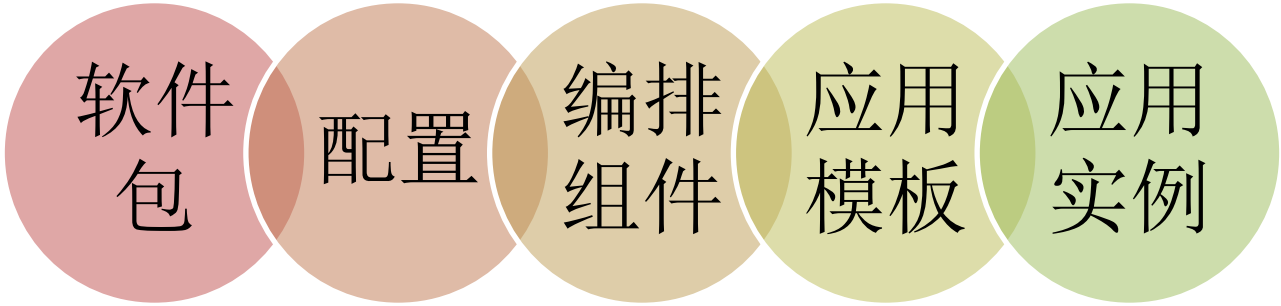


生成软件包和配置文件

基于软件包、配置和目标环境，设计应用架构

基于编排文件，分配资源、部署应用、配置系统

五大界面



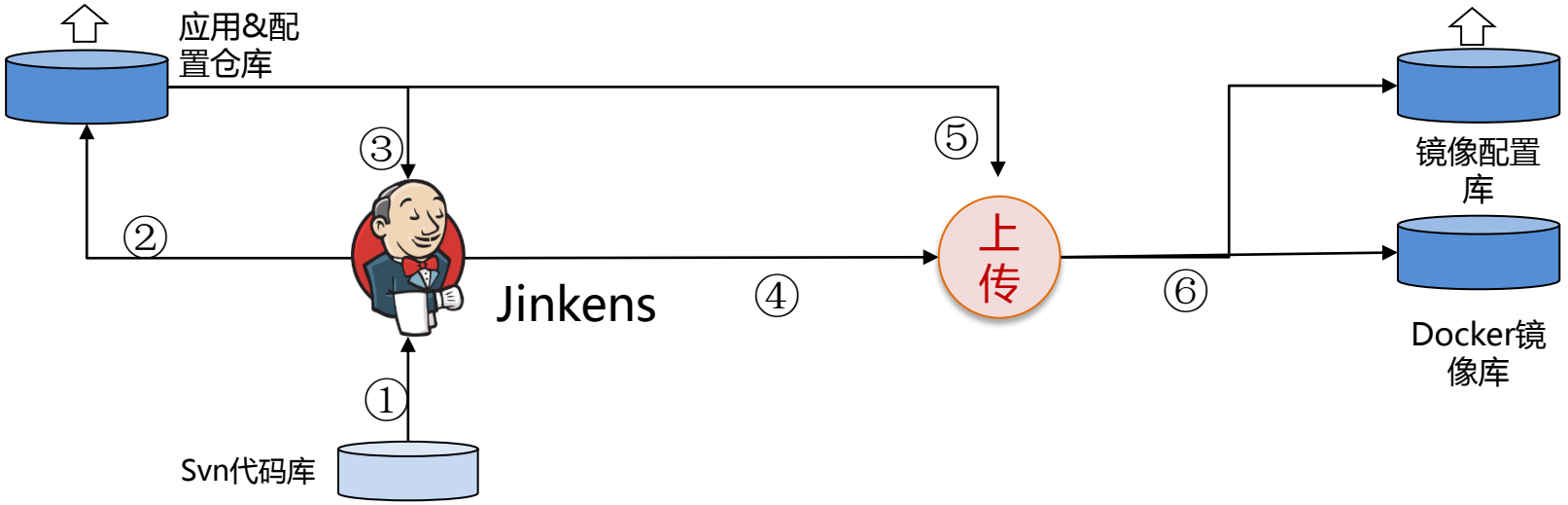
软件包

应用封装格式

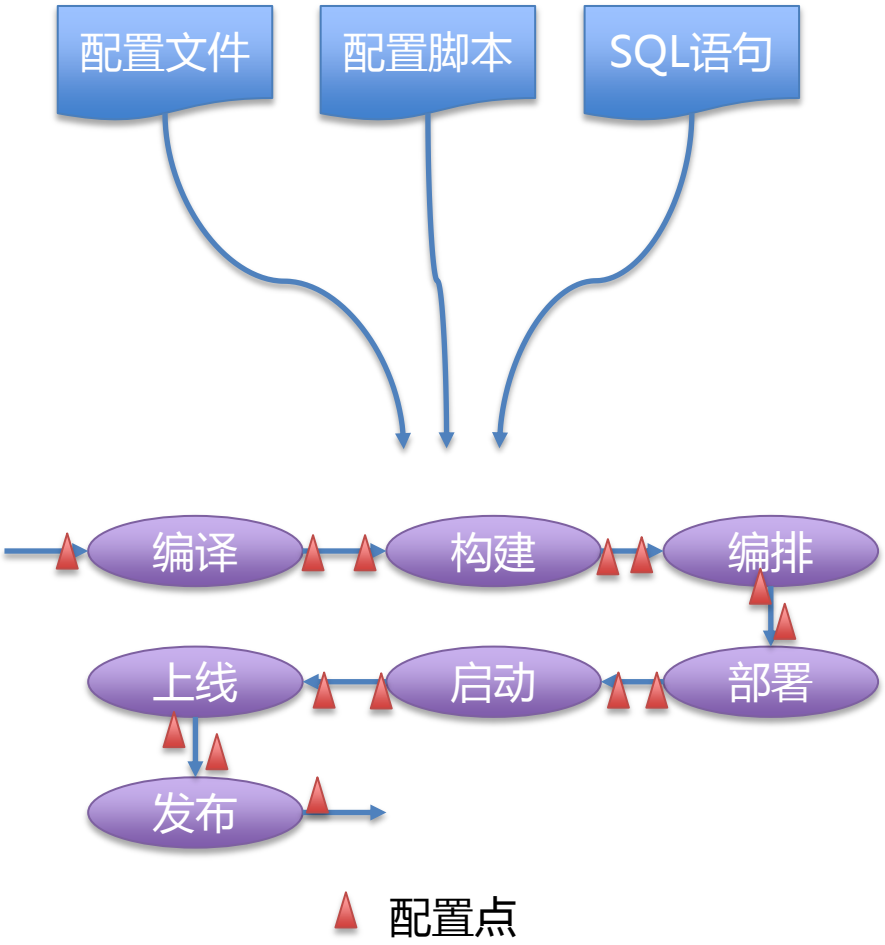
/app1/2.5/bin → 支持jar、tar、war、rpm等多种格式
/app1/2.5/template → 环境变量设置
/app1/2.5/scripts → Jinja2模板文件，支持多种循环、条件、宏、检查等语法
/app1/2.5/environment → 安装后脚本执行
/app1/2.5/deploy → 安装到物理机、容器等多种平台
/app1/2.5/resource
/app1/2.5/readme

多平台配置文件

Docker主机直接部署dockerfile → /imgtag/docker
Kubernetes框架pod定义文件 → /imgtag/kubernetes
Swarm调度配置与调度配置文件 → /imgtag/swarm
Marathon加mesos调度执行配置文件 → /imgtag/marathon



配置管理



配置三要素：

配置类型

不同的类型对应不同的处理流程，比如配置文件类型由配置文件插件处理；支持的类型包括：文件、可执行命令（shell或其他可执行程序）、SQL命令等。

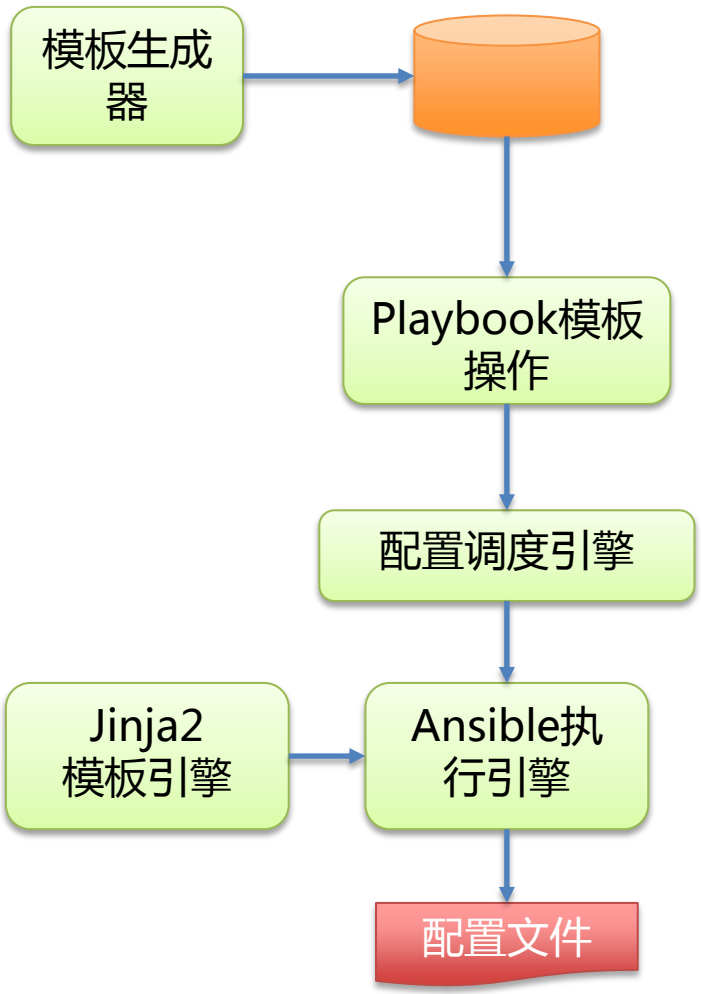
配置输入

不同的配置类型，其输入格式分别对应于配置模板文件、可执行命令（文件）、SQL语句；同时，配置输入还包括配置所需的其他参数，比如配置目标路径等，供插件具体执行配置时作为输入参数。

配置阶段

配置分为检查阶段、环境准备阶段、部署前阶段、部署后阶段、启动前阶段、启动后阶段等，配置阶段定义了配置操作发生的阶段点（Stage），根据不同的配置，对应不同的配置阶段。

配置模板处理



配置模板

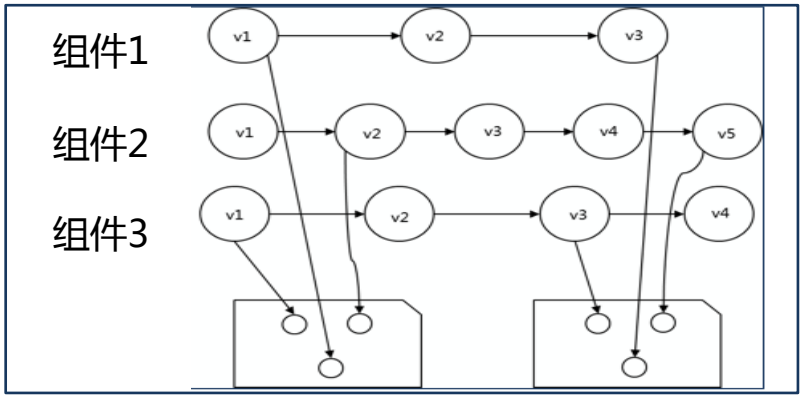
```
user root;
worker_processes {{ ansible_processor_count }};
pid /var/run/nginx.pid;
events {
    worker_connections {{ connections }} ;
    # multi_accept on;
}
http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
include /etc/nginx/mime.types;
    default_type application/octet-stream;
    access_log /var/log/nginx/access.log;
gzip on;
    gzip_disable "msie6";
    gzip_vary on;
gzip_types text/plain text/css application/json application/x-javascript
text/xml application/xml application/xml+rss text/javascript;
    map $scheme $server_https {
        default off;
        https on;
    }
    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

可编排组件

组件实例

- App1
- Docker
- （1.0）.(1.0).(all)
- Admin
- 2016-01-01 18:00:00
- Config:
  - Warehouse=docker:10.0.1.11:8123
  - imgName=nginx
  - Template:
    - Path=/template/xx.config
    - Dst=/usr/sbin/etc

应用版本



组件MetaData：

1，基础信息（名称、版本号、类型、时间）

2，依赖信息（依赖的组件、软件、服务或者环境变量）

3，基础配置信息（获取路径、访问认证信息等）

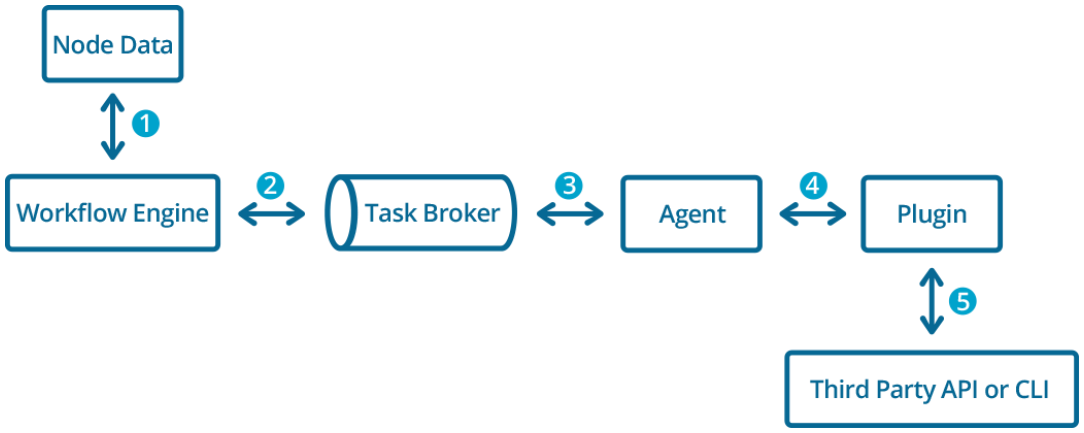
4，配置信息（配置项列表，每个项中包含关于配置类型、依赖等信息）

5，安装部署命令、脚本信息

6，输出的元数据类型信息

编排模板

Cloudify编排引擎



工作流

Blueprint

插件

编排模板（blueprint）

```
imports:
inputs:
  host_ip:
node_templates:
  nodecellar:
    type: nodecellar.nodes.NodecellarApplicationModule
    relationships:
        - type: node_connected_to_mongo
        target: mongod
mongod:
    type: nodecellar.nodes.MonitoredMongoDatabase
    relationships:
- type: cloudify.relationships.contained_in
        target: host
  nodejs:
    type: nodecellar.nodes.NodeJSServer
    relationships:
      - type: cloudify.relationships.contained_in
        target: host
  host:
    type: nodecellar.nodes.MonitoredServer
    properties:
      ip: { get_input: host_ip }
outputs:
  endpoint:
    description: Web application endpoint
    value:
      ip_address: { get_property: [ host, ip ] }
      port: { get_property: [ nodecellar, port ] }
```

编排技术对比

TOSCA(cloudify)

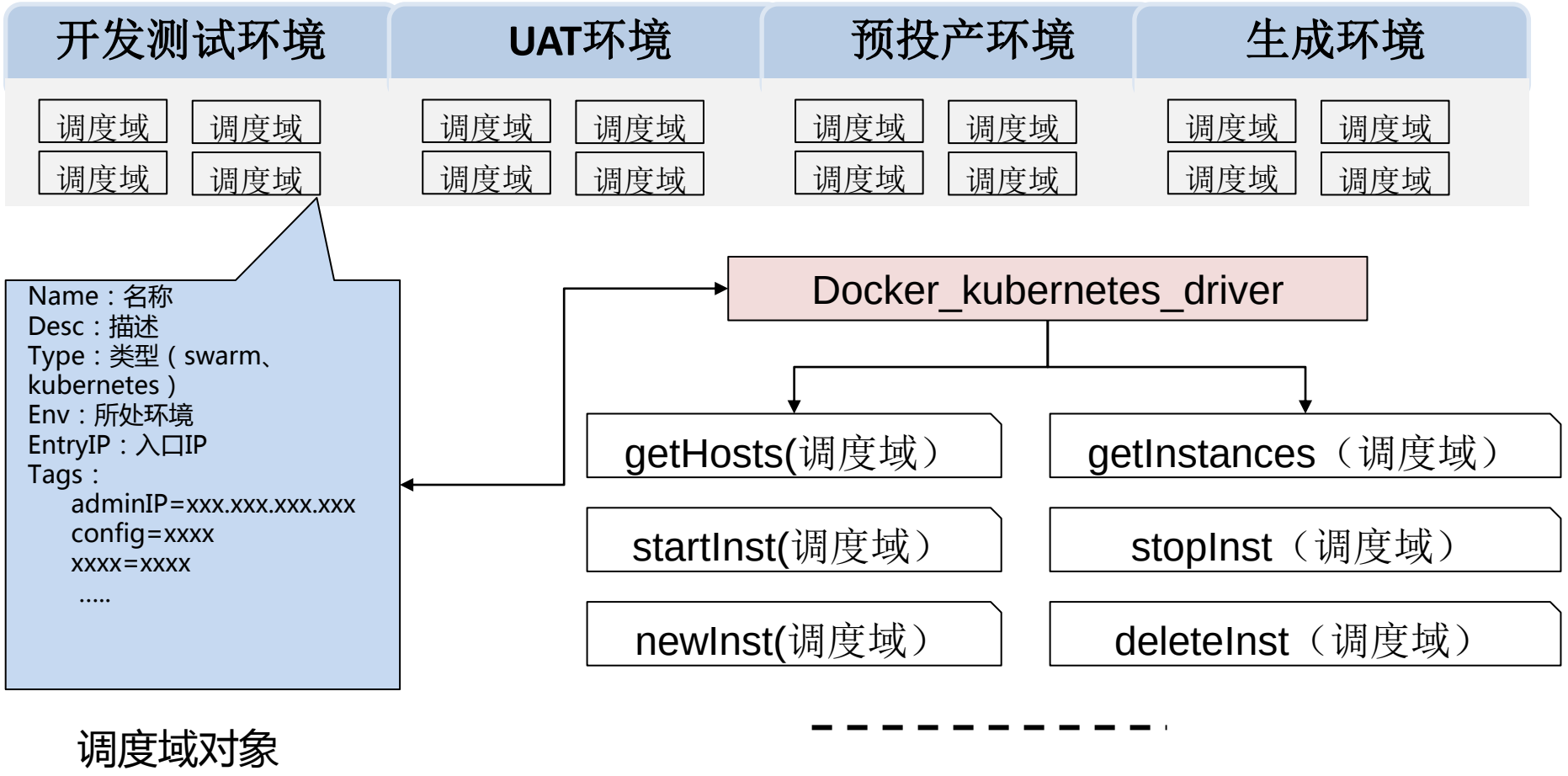
- 支持多种软件包格式
- 支持多种目标平台
- 支持插件，易于扩展
- 工作流自定义，方便集成应用运维
- 开放组织

VS

Docker Compose

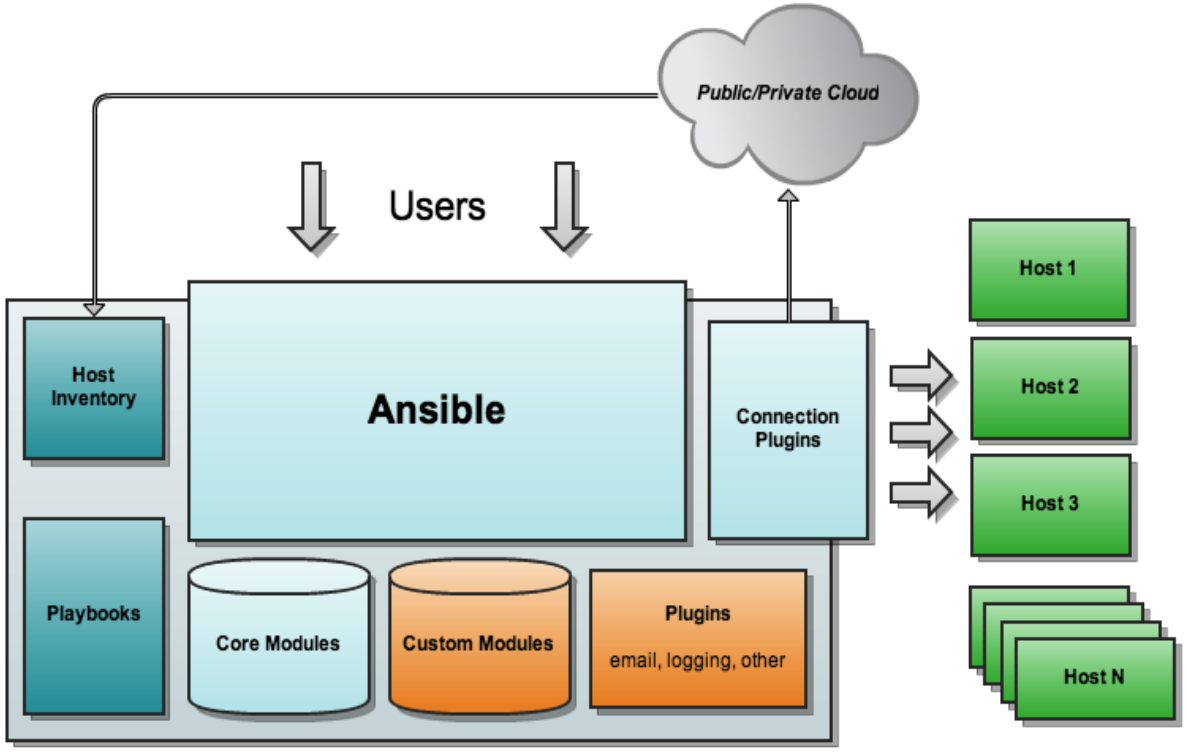
- Docker官方支持
- 丰富的Docker镜像资源可以利用

环境管理


开发测试环境
UAT环境
预投产环境
生成环境
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
调度域
Name：名称
Desc：描述
Type：类型（swarm、kubernetes）
Env：所处环境
EntryIP：入口IP
Tags：
adminIP=xxx.xxx.xxx.xxx
config=xxxx
xxxx=xxxx
.....
Docker_kubernetes_driver
getHosts(调度域）
getInstances（调度域）
startInst(调度域）
stopInst（调度域）
newInst(调度域）
deleteInst（调度域）

调度域对象

部署引擎



Ansible优势：

1，无代理，更加简便；

2，基于Python，可自定义扩展，增强功能；

3，配置语言基于Yaml，简单易学；

4，丰富的集成组件，在线资源库强大；

5，支撑厂商强大，社区发展活跃；

应用实例管理

系统名称	当前运行版本	部署历史
LDP	V1R6	查看部署历史

系统名称：LDP

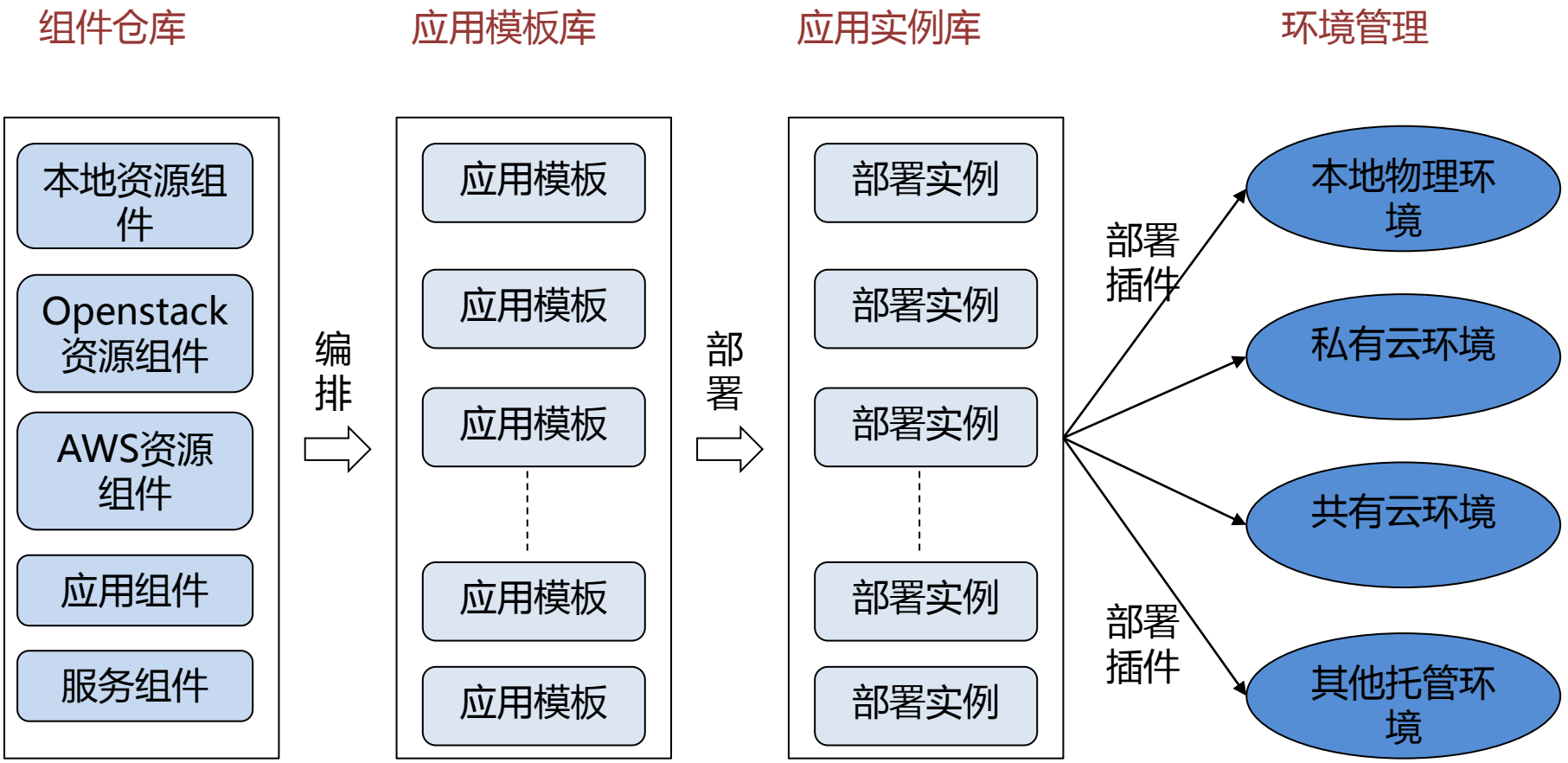
部署版本	部署时间	版本状态	操作
V1R6		运行中	下线
V1R5		已下线	回滚到该版本
V1R4		已下线	回滚到该版本

LDP系统回滚到的V1R5版本将下线当前版本，请确认

确认　取消

回滚操作是一个下线当前版本，并按照对应的历史模板，重新部署历史版本的过程

总结


组件仓库
应用模板库
应用实例库
环境管理
本地资源组件
Openstack资源组件
AWS资源组件
应用组件
服务组件
编排
应用模板
应用模板
应用模板
应用模板
应用模板
部署
部署实例
部署实例
部署实例
部署实例
部署实例
部署插件
部署插件
本地物理环境
私有云环境
共有云环境
其他托管环境

THANKS!