# 容器—DevOps的必由之路

--标准化带来devops

# About me

张春源

希云cSphere

**恪守契约精神,务实开放合作。**

专一、专注为企业客户提供容器私有云平台。已为金融,国企,高校,汽车,航空等企业提供过基于容器云的完整解决方案

# DevOps的好处



## Development Operations

改善软件开发人员、测试人员和运维人员之间的协作关系，并通过自动化流程（系统）更加快捷、频繁、易重复且可靠的构建软件、测试、以及发布部署。

**敏捷开发**

软件版本发布后可快速发现问题，并及时修改，修改完成后通过pipeline流水线即可快速得完成部署并进行测试。

**高效交付**

确定发版后，自动触发构建步骤，将代码以及运行环境整体打包并交付到测试或运维。

**打通IT工具链**

从开发到测试再到运维，3个大部门之间通过DevOps打通后可形成闭环的生态。

**重复且可靠**

从代码打包，以及交付，整个pipeline正确地执行了上千遍，自动且不是手动操作。

**对人员弱依赖**

交付流程代码化，大部分工作由系统完成，不依赖特定人员。

**低风险**

上千上万遍执行无误，极大提升了发布的正确性，且多版本可双重降低用发布的风险。

**有力支撑快速发展的业务市场**

多一次正确的发布、缩短处理故障的时间，快速应对市场的变化等，均是实践DevOps的价值体现。

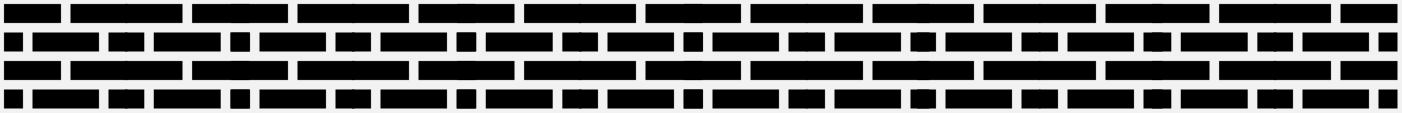
**大家都是主人**

从系统开发、测试、以及上线，整个过程大家都参与其中，熟知项目的进度及项目情况。

**低成本扩展**

企业文化及自动化系统的建立，大规模扩展成本重点在采购硬件设备上，而不是大量招聘工程师。

# DevOps的前世



自动化脚本

配置引擎

- cfenginne
- puppet
- chef
- ansible

# 无法在企业中落地的原因

脚本的缺陷

- 人员强依赖
- 不具备收敛
- 没有标准
- 不具备回退

配置引擎的缺陷

- DSL
- 门槛高
- 解耦不够
- 掌握的人少

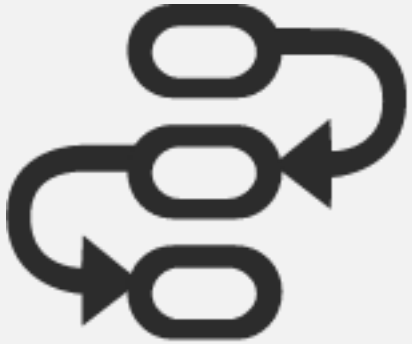
悬空

多维度，多因素

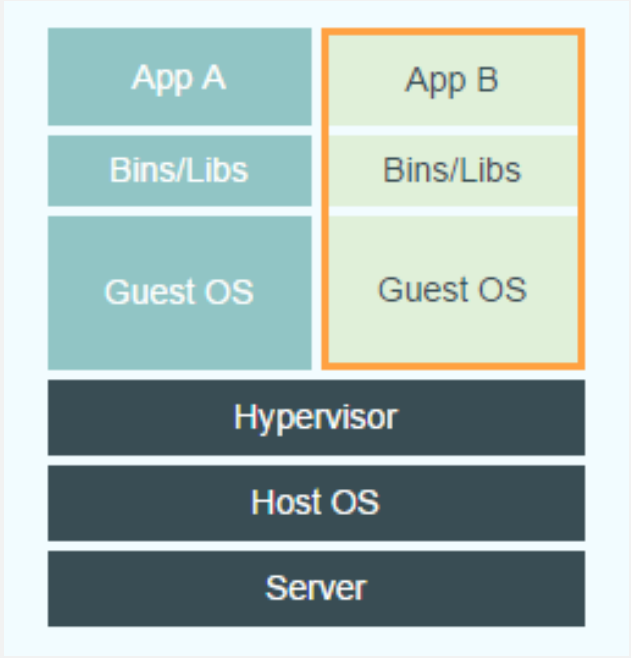
# 某客户实践DevOps失败的总结



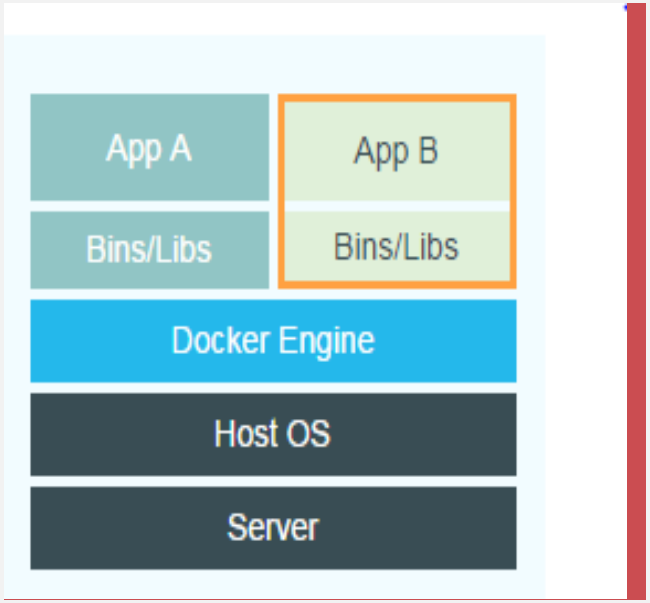
| 很难形成从开发到测试 | |
|---|---|
| 招人难 | 工资高 |
| 开发运营分裂 | 开发掌控工具难 |
| 再到生产的统一的一致的流水线作业 | |



# 容器是什么？肯定不是虚拟机！




虚拟机

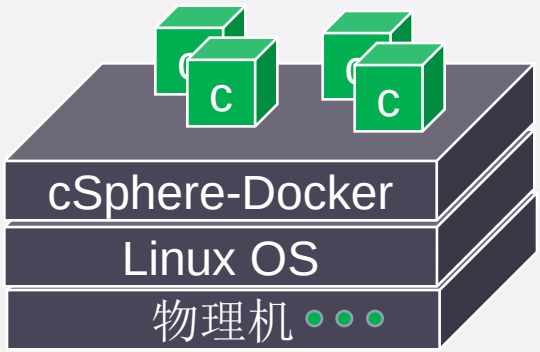



容器

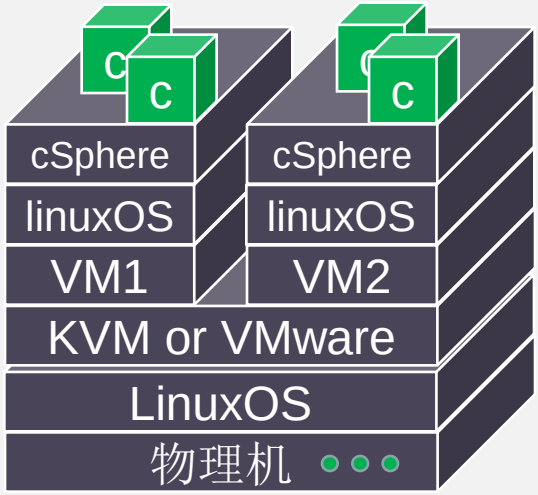
# 容器vs虚拟机vs物理机服务器



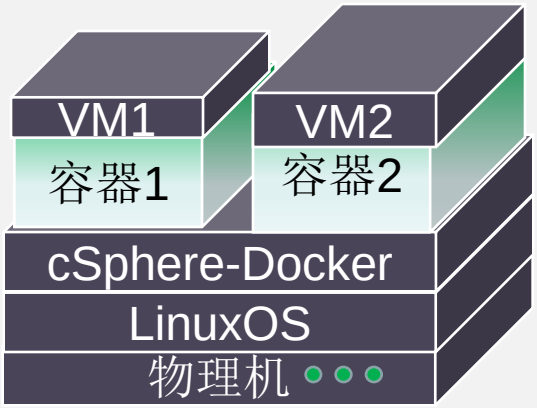
Container容器




物理机&容器




虚拟机&容器

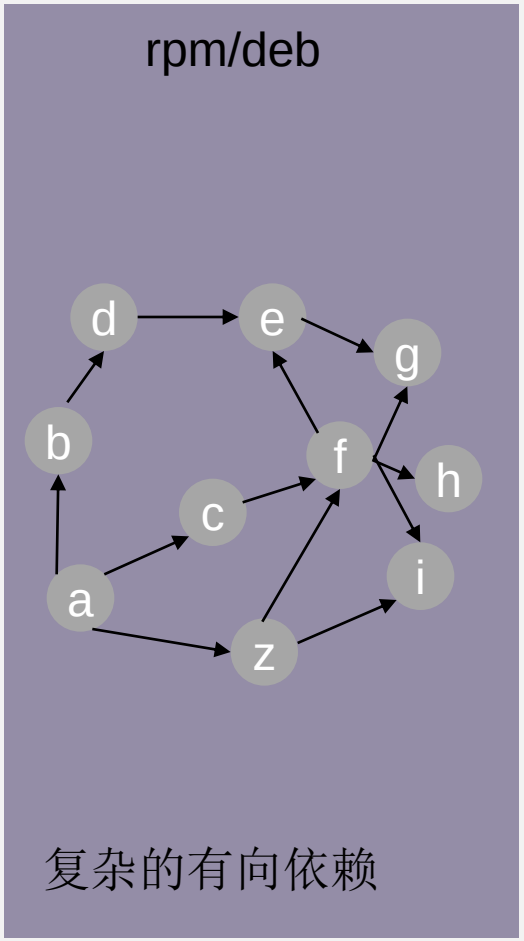


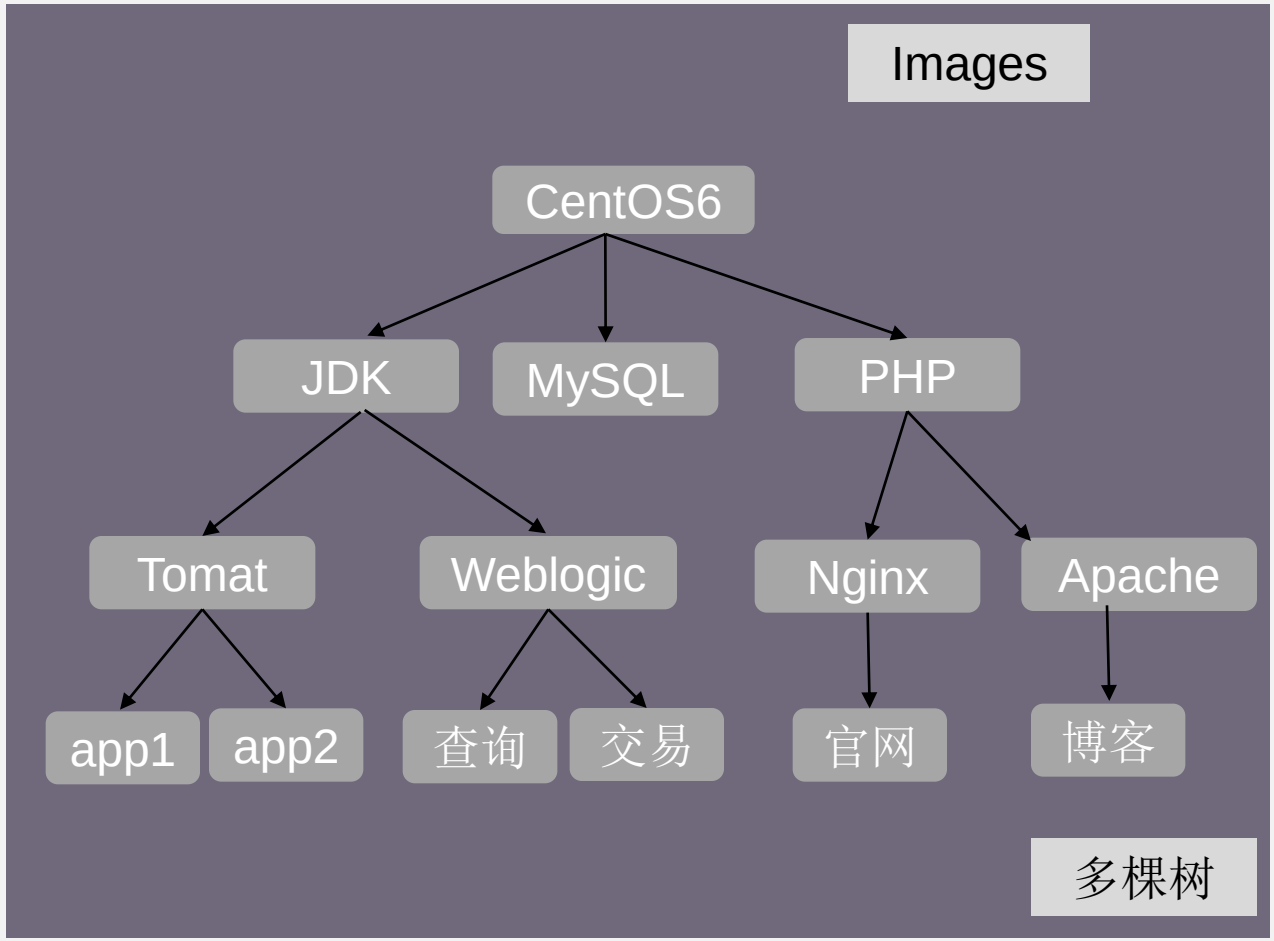

物理机&容器&虚拟机

# 容器是增强版进程

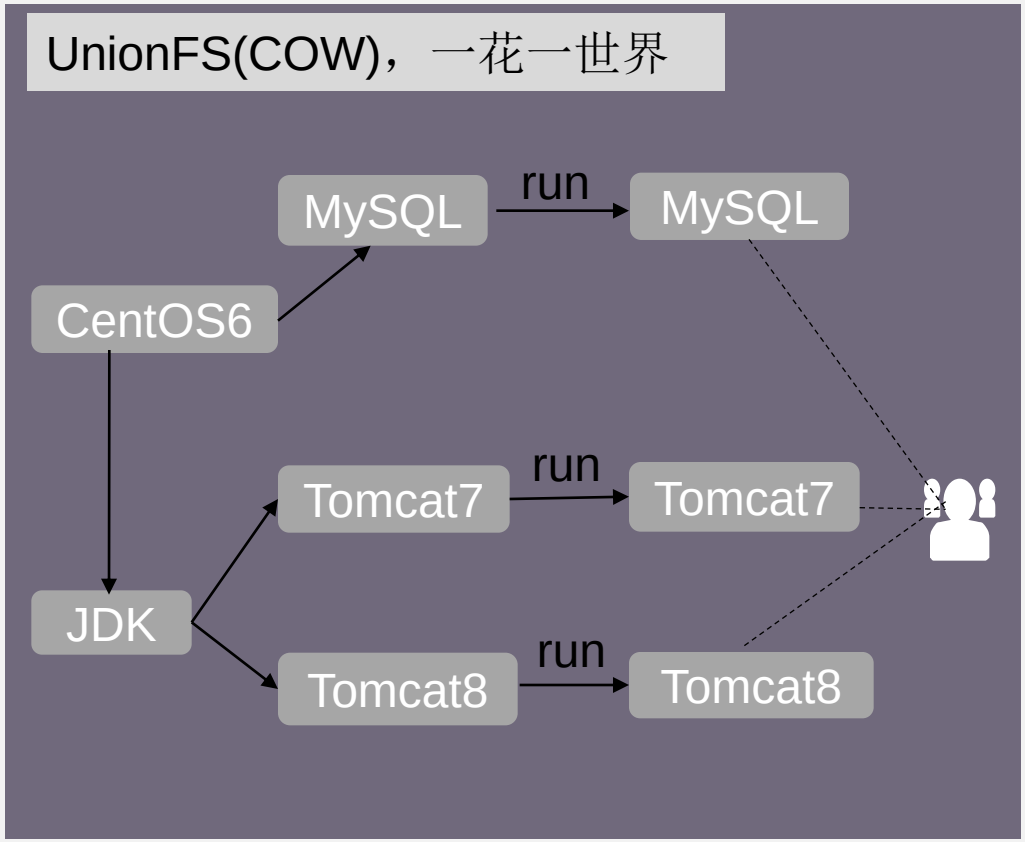
| 对比项 | 传统 Linux | Container Linux | |
|---|---|---|---|
| 软件包 | rpm/deb | images | |
| | 有向图定义依赖 | 单继承、UnionFS | |
| 进程 | 普通进程 | 增强版进程 | |
| | 所有进程在一个平面、管理界面单一 | 隔离 | 管理API (启停、统计进程信息) |

# rpm/deb vs images

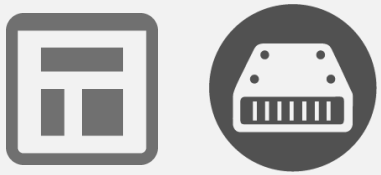


# UNIONFS

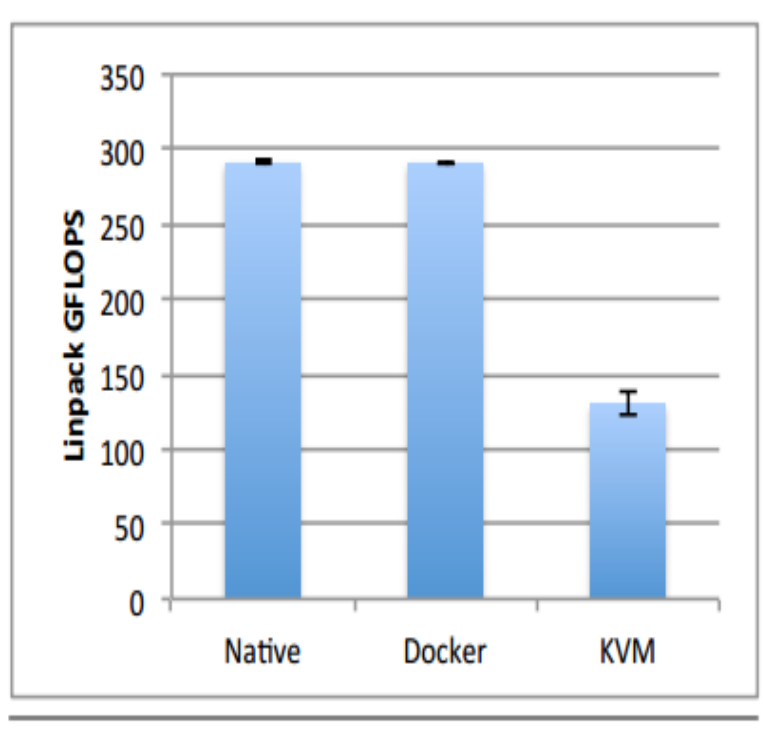


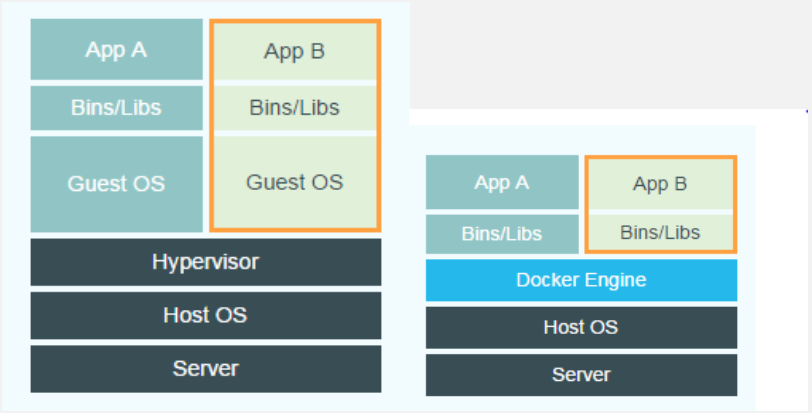

- 逻辑上看到的服务都是独立的
- 彼此间修改不会发生冲突



# 容器性能

## IBM x3650 M4服务器



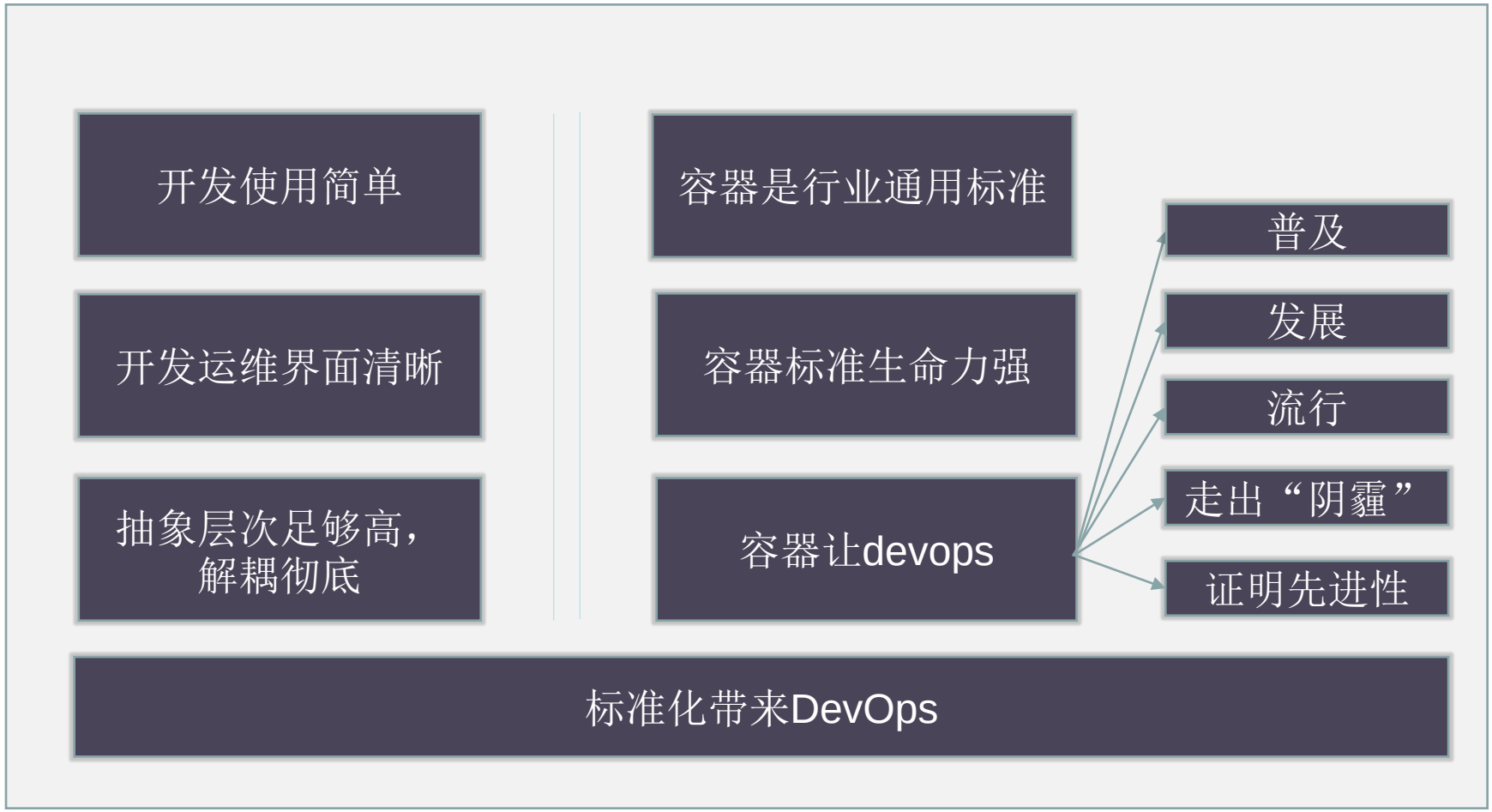




虚拟机　　容器

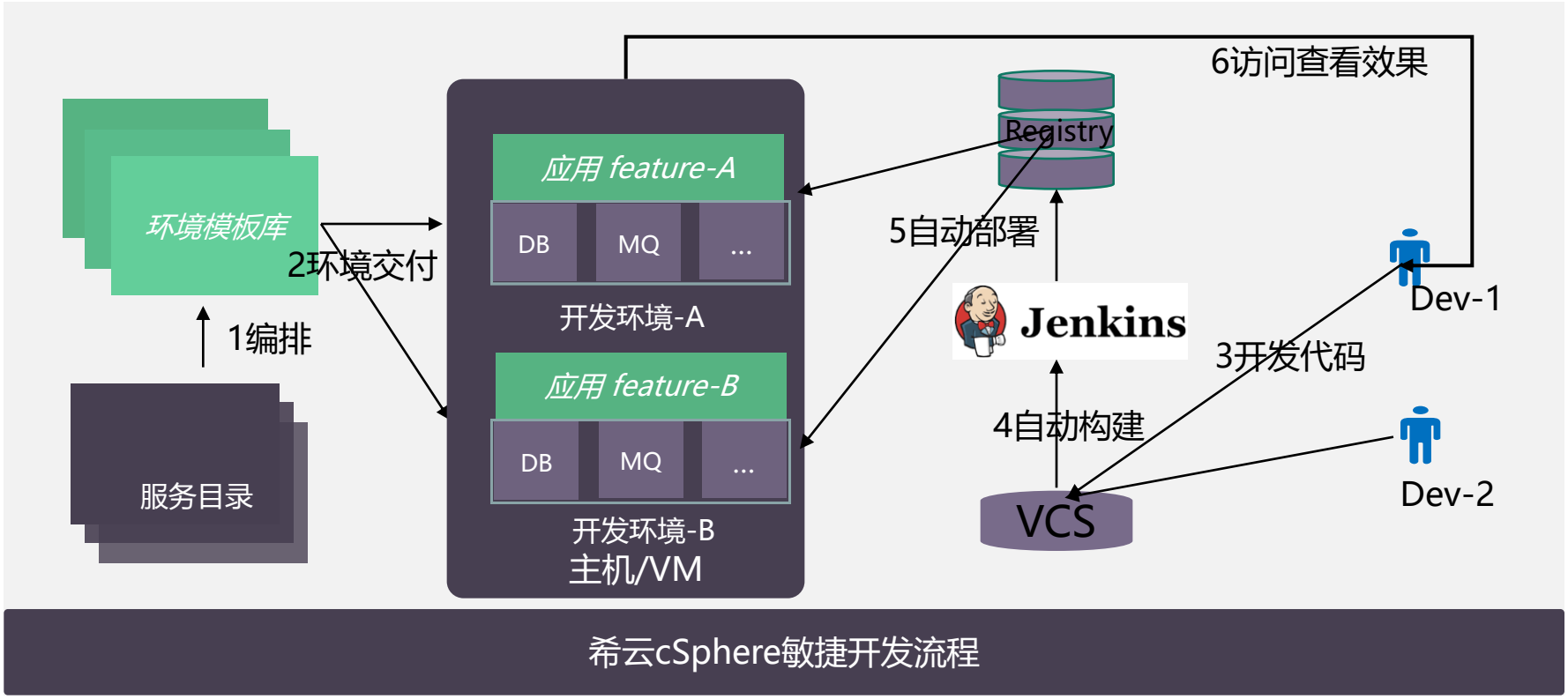
# 容器技术恰恰克服了这些阻力



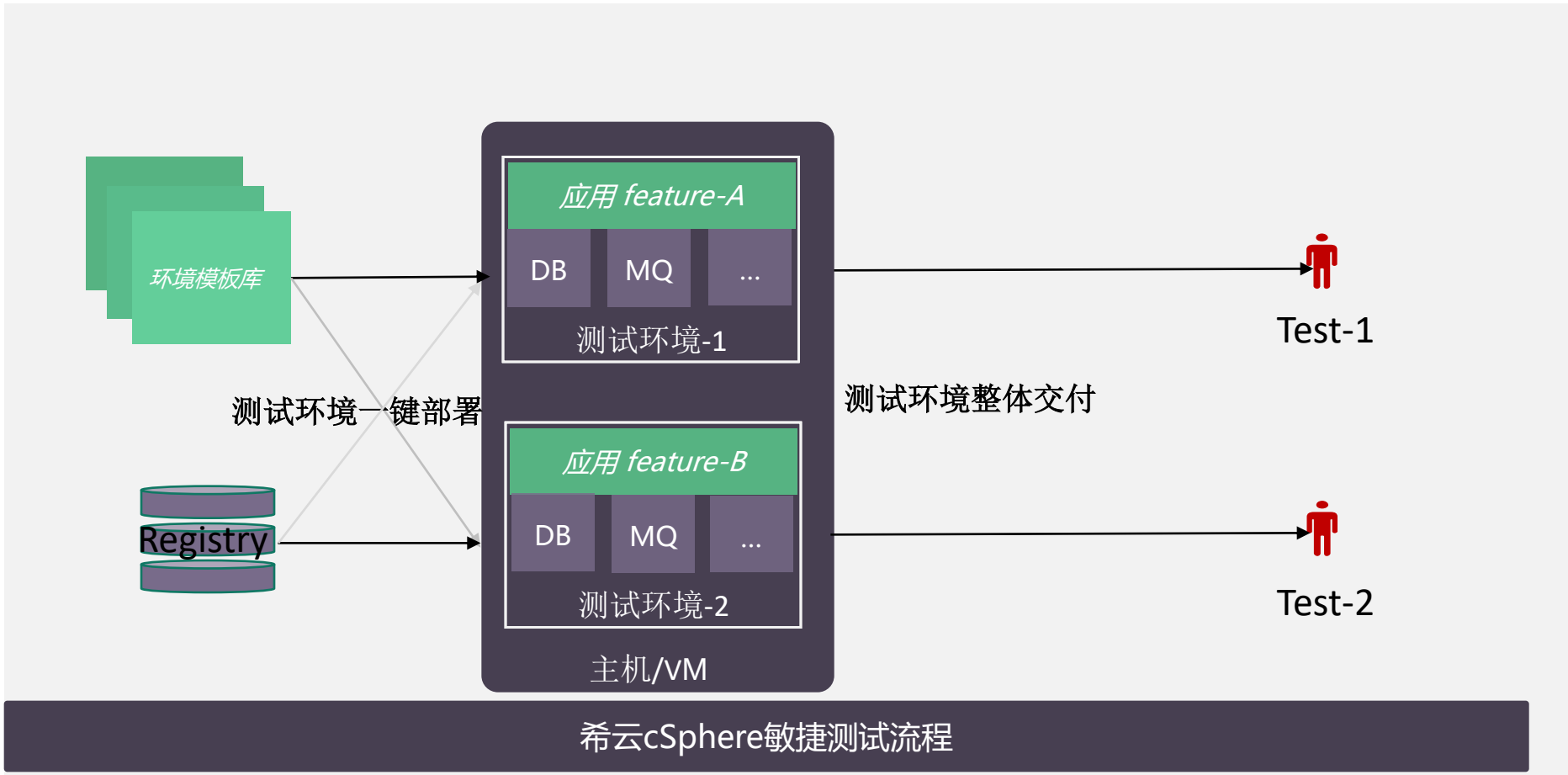
# 为何容器克服了这些问题



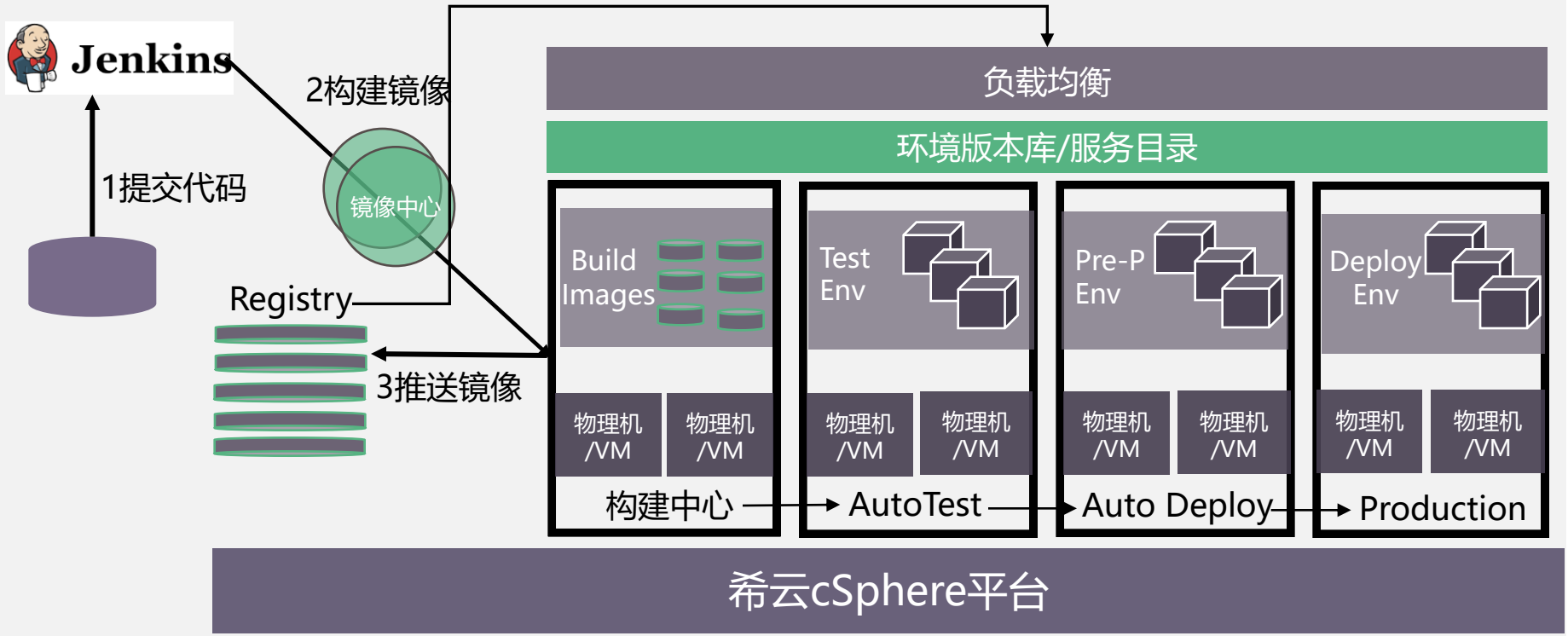
# 容器-敏捷开发


6访问查看效果
Registry
应用 feature-A
DB
MQ
...
开发环境-A
应用 feature-B
DB
MQ
...
开发环境-B
主机/VM
环境模板库
2环境交付
1编排
服务目录
5自动部署
Jenkins
3开发代码
4自动构建
VCS
Dev-1
Dev-2
希云cSphere敏捷开发流程

# 持续测试


环境模板库
Registry
测试环境一键部署
应用 feature-A
DB
MQ
...
测试环境-1
应用 feature-B
DB
MQ
...
测试环境-2
主机/VM
测试环境整体交付
Test-1
Test-2
希云cSphere敏捷测试流程

# 基于容器建设DevOps



# DevOps其他技术路线为什么必然失效

# DevOps其他技术路线为什么必然失效

- 生命力弱
- 内部标准
- 难推广


- 生命力极强
- 行业通用标准
- 易推广

# 容器为什么是DevOps的必由之路



- 更小、更频繁的变更

技术需求

- 持续集成
- 自动构建
- 自动化发布
- 快速回滚


# 无容器，应用变更存在的问题



- 构建环境不确定
- DSL难度高
- 发布结果不一致
- 回滚周期长

# 有容器，应用变更的优点



- 构建环境确定
- 可视化变更，门槛低
- 发布结果一致
- 周期短，秒级回滚

# 容器为什么是DevOps的必由之路



- 让开发人员更多的控制生产环境

技术需求

- 有限度授权
- 可视化操作
- 操作审计
- 可视化查询

控

# 无容器存在的问题



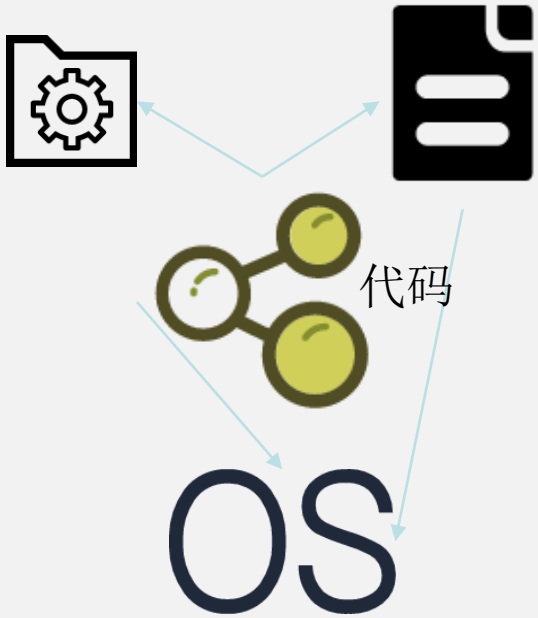
- 控制粒度难控制
- 命令行操作
- 依赖外部系统
- 管理系统分散

# 使用容器的优点

- 细粒度授权
- 可视化操作
- 精密审计
- 高度集成，易维护

# 容器为什么是DevOps的必由之路

- 以应用程序为中心来理解基础设施




**依赖程序难以管理**

- 手动修改，无记录
- 配置管理与代码分离
- 依赖的修改复杂，速度慢

**基础设施管理复杂度高**

- 不同角色的服务器
- 经常的配置变更
- 同时管理不同版本的操作系统

# 将容器作为应用的基础设施

# 容器为什么是DevOps的必由之路



- 定义简单明了的流程

| | |
|---|---|
| 流程复杂 | • 不同类型的应用程序，项目组的项目，都会有不同的部署，升级，回滚流程<br>• 复杂度高<br>• 开发人员对代码，架构的调整，会导致运维人员做很多相应配置变更工作<br>• 周期长，易出错 |
| 开发与运维冲突 | • 开发人员由功能性需求驱动，往往需要经常做变更以满足产品需求<br>• 运维人员希望尽量避免变更，以增加稳定性，避免出错 |

# 容器为什么是DevOps的必由之路



- 定义简单明了的流程

流程简单

- 所有类似的程序，pull镜像，start容器，统一应用发布，升级，回滚流程
- 复杂度低
- 开发人员对代码，架构的调整，只需修改镜像，运维人员不需要跟着做改动
- 周期端，不出错

开发运维不再冲突

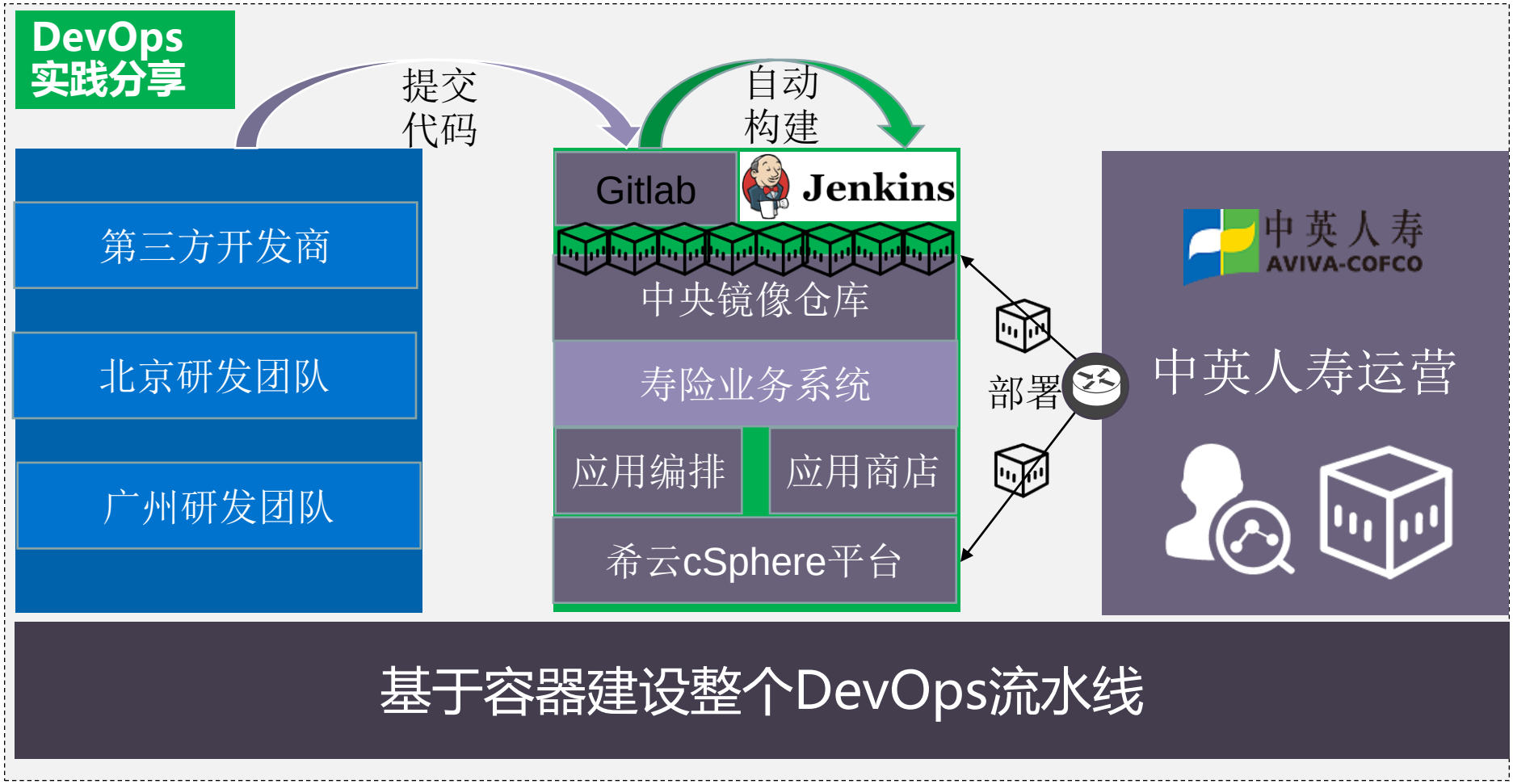
- 开发人员由产品需求变化代理的频繁变更可以独立解决
- 运维人员只需管理好容器运行平台，提供好容器应用发布和升级的流程即可

# 中英人寿保险有限公司DevOps实践--案例分享

**总体目标**

- 如何用容器克服第三方开发商和企业IT管理之间的协作
- 如何通过自动化交付提升协助的效率

# 中英人寿保险有限公司DevOps实践--案例分享



# 中英人寿保险有限公司DevOps实践--案例分享

## 效果收益

**标准化**第三方开发商交付物，中英运营人员标准统一进行部署和管理

**2个月**完成2个应用的开发、测试、上线

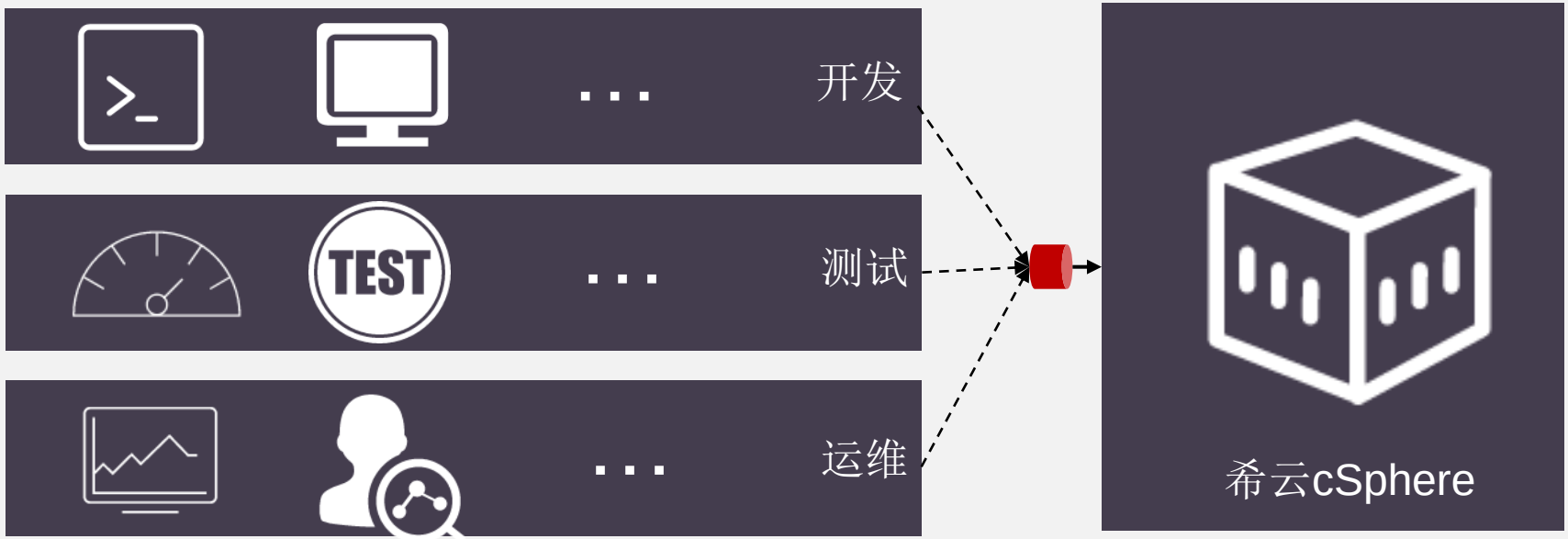
**服务器资源**提升70%、**交付时间**缩短60%以上

整体**工作效率**提升80%

效率

成本

# 未来DevOps和容器合流



恪守契约精神，
务实开放合作。