DevOpsDays

Shanghai

— 2017.8.18-8.19 —

上海龙之梦酒店（长宁区延安西路1116号）

主办单位：高效运维社区 GreatOPS Community　Best Practice 最佳实践

Cloud Native（云原生）应用实践

王磊

华为/中软/软件工程技术专家
ThoughtWorks首席咨询师
Sybase Tech Leader

- 《微服务架构与实践》作者
- 《DevOps Handbook》中文译者之一
- 国内较早倡导和实践微服务的先行者
- 对自动化测试/持续集成/持续交付有丰富的实战经验
- 西安DevOps Meetup 联合发起人

目录

目录

Cloud Native起源

2010.5
Cloud Native@Blog

Paul Fremantle
CTO Of WSO2


Cloud Native

Archived Content

by Paul Fremantle 30 May, 2010

One of our team - Lavi - has a great analogy. Think of a 6-lane freeway/motorway/autobahn as the infrastructure. Before the autobahn existed there were forms of transport optimized first to dirt tracks and then to simple tarmac roads. The horse-drawn cart is optimized to a dirt track. On an autobahn it works - but it doesn't go any faster than on a dirt track. A Ford Model T can go faster, but it can't go safely at autobahn speeds: even if it could accelerate to 100mph it won't steer well enough at that speed or brake quickly enough.

About Author

Paul Fremantle

- **分布式**
- **多租户/按需收费**
- **弹性/可伸缩**

http://wso2.com/library/articles/2010/05/blog-post-cloud-native/

Cloud Native定义

2013.12

Cloud Native Architecture
@Yow Conference

Adrain Cockroft

VP Cloud Strategy of AWS
ex-Netflix Cloud Architect

- 基于公有云
- 微服务/伸缩性/可用性/自愈性
- 持续部署、DevOps
- 敏捷、效率、高度信任的组织

Cloud Native定义

2015.2

Migrating to Cloud Native Application Architectures

Matt Stine

Pivotal， Architect

- 微服务
- DevOps
- 持续交付
- 敏捷基础设施
- 基于API协作
- 十二因子

Cloud Native定义

2015.7

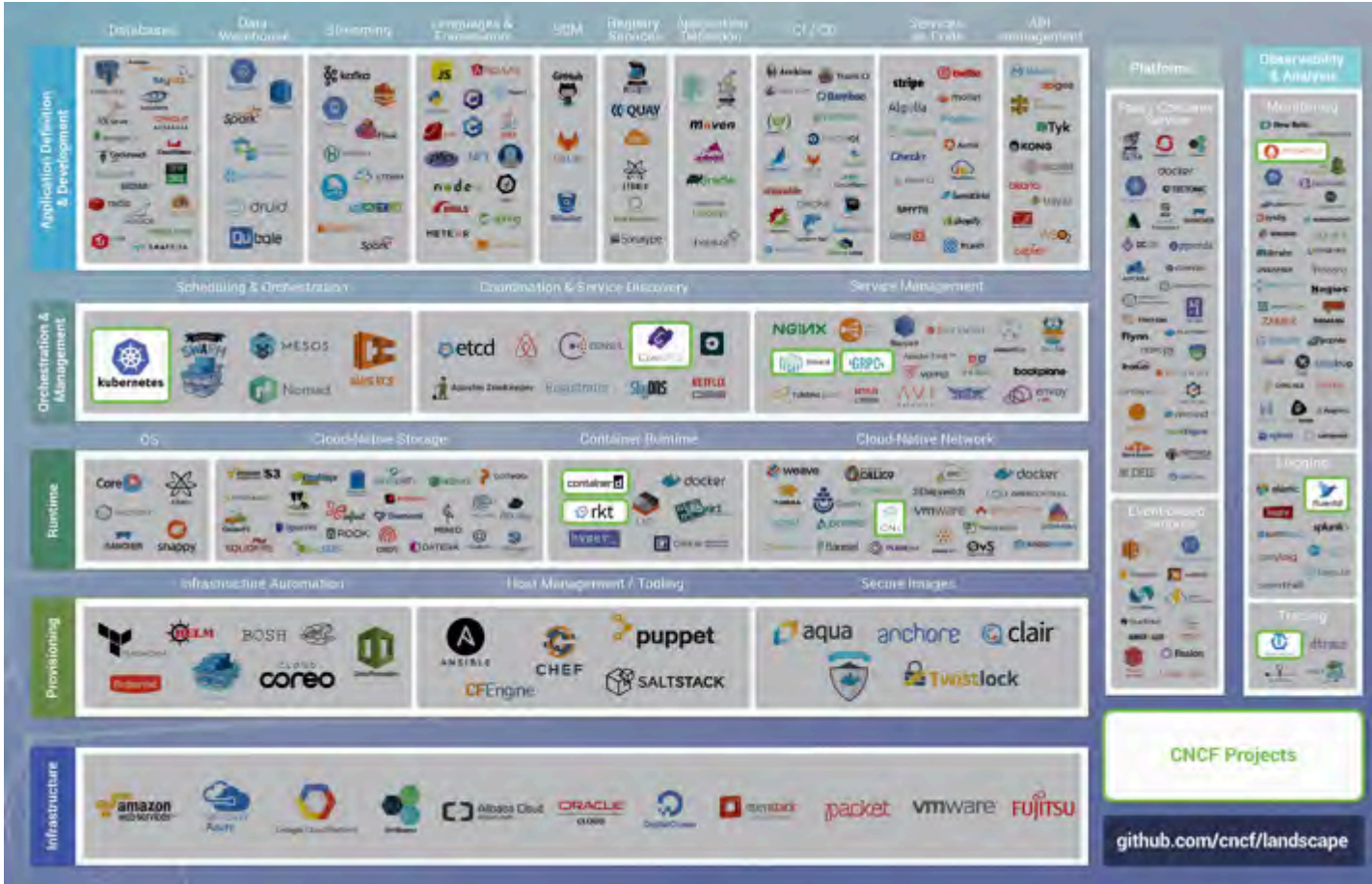
What is Cloud Native?

Cloud native computing uses an open source software stack to be:

1. Containerized. Each part (applications, processes, etc) is packaged in its own container. This facilitates reproducibility and resource isolation.
2. Dynamically orchestrated. Containers are actively scheduled and managed to optimize resource utilization.
3. Microservices oriented. Applications are segmented into microservices. This significantly increases the overall agility.

- 容器化(Containerized)
- 动态编排(Dynamically orchestrated)
- 面向微服务(Microservices oriented)

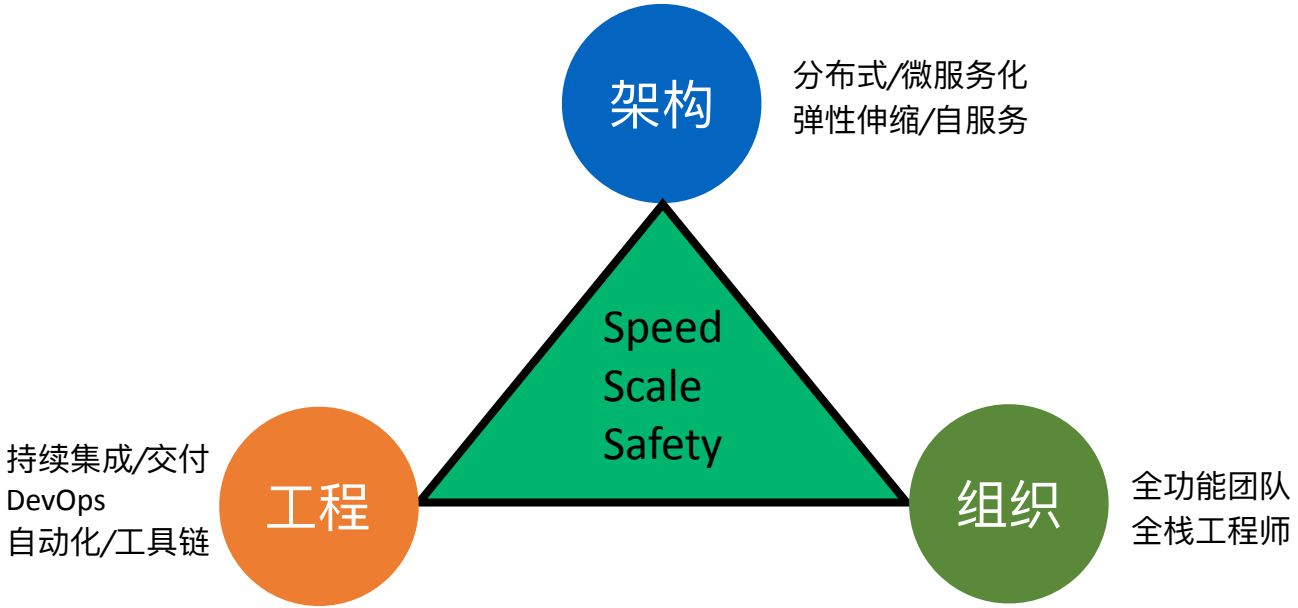
Cloud Native LandScape



目录

Cloud Native的核心特征

基于云的基础设施，运行/管理系统的实践范式



Cloud Native的核心特征 - 架构

应用架构

- 快速响应
- 弹性伸缩
- 可用性
- 可重用

基础设施架构

- 云化基础设施
- 自服务接口
- 多租户/可靠性
- 可用性

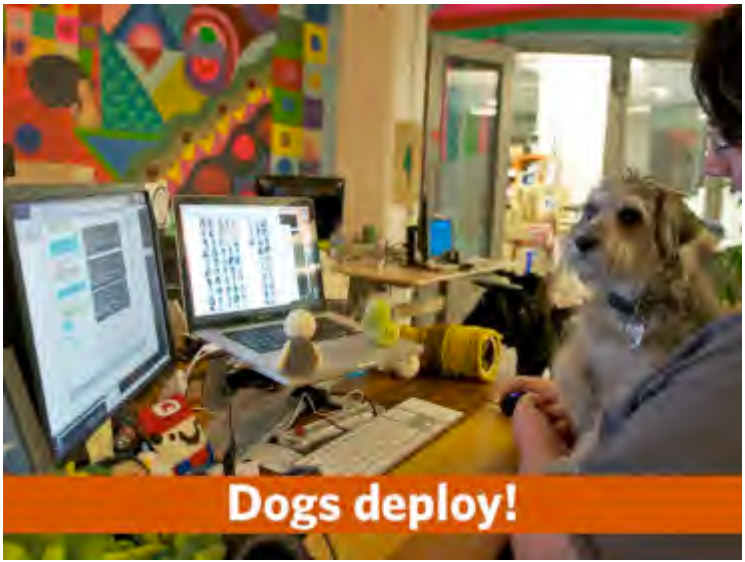
伸缩性、快速响应

稳定性，可靠性

Cloud Native的核心特征 - 工程

持续交付

- 低风险/高质量/自动化
- 将发布作为业务决策



https://www.slideshare.net/beamrider9/continuous-deployment-at-etsy

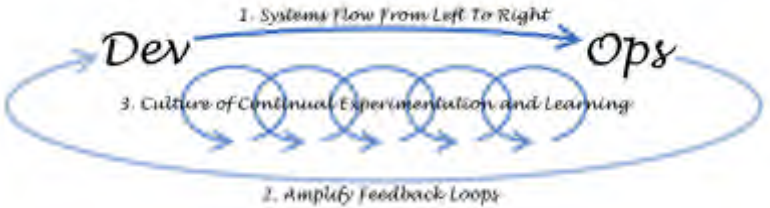
Cloud Native的核心特征 - 工程

DevOps 与 CALMS

- **C**ulture（文化） - 合作&沟通的信任文化
- **A**utomation（自动化） - 自动化“一切”
- **L**ean（精益） - 消除浪费，小批量交付
- **M**easurement（度量） - 收集反馈，持续改进
- **S**haring （分享） - 分享经验 & 知识

Cloud Native的核心特征 - 工程

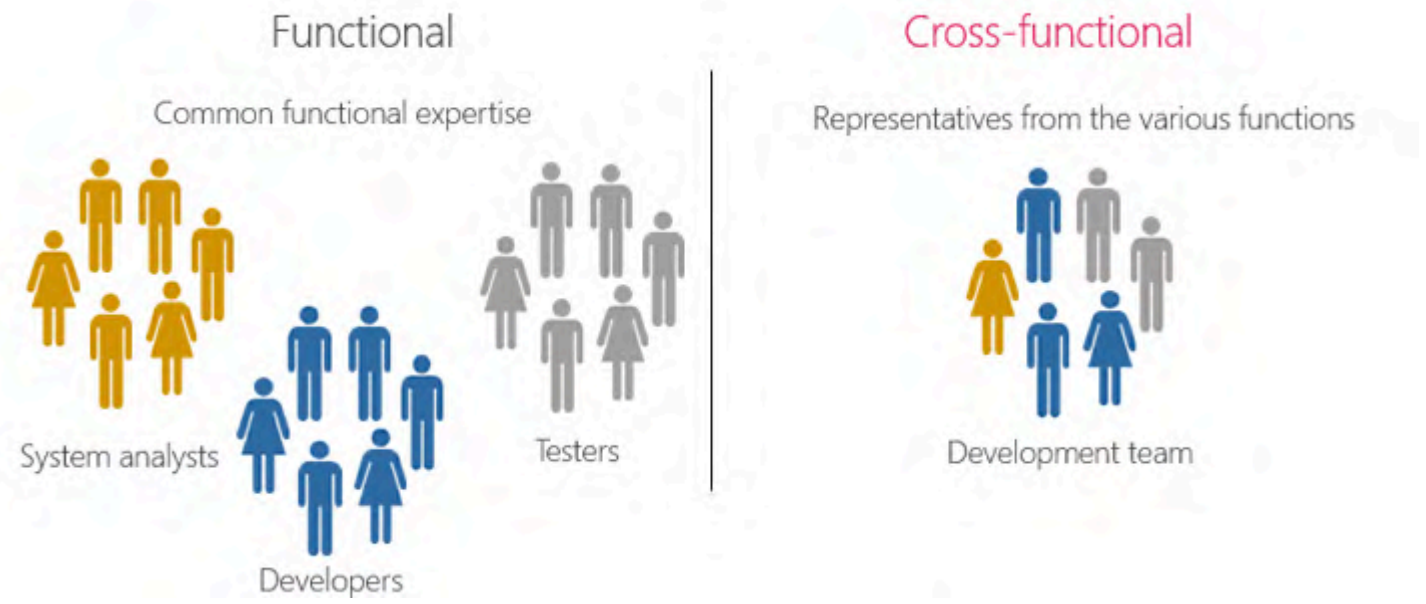
DevOps 与 3 Ways



https://www.icore-ltd.com/devops-three-core-principles/

Cloud Native的核心特征 - 组织

全功能团队



Cloud Native的核心特征 - 组织

全栈工程师

目录

一个真实的演进案例

Spring MVC

MyBatis

JSP

Oracle

以营促销的业务系统

40W+

20+Member

开发与运维分离

3 Month

20 Minute

手动测试为主

多人共享一套环境

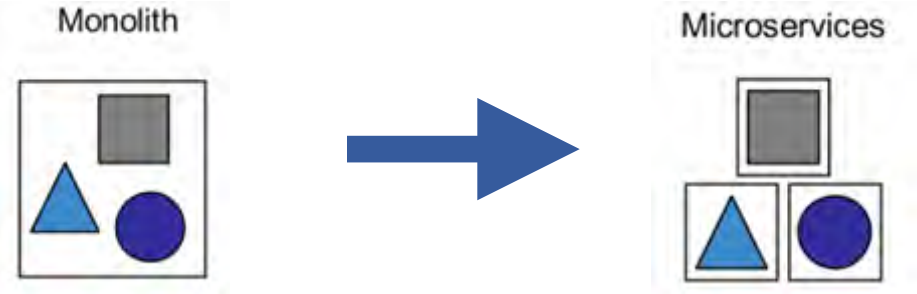
期望

- 2/3周完成特性上线
- 资源按需水平伸缩
- 系统具备高可用性
- 独立的全功能团队

现实

- 版本发布周期3个月
- 单体应用/伸缩成本高
- 停机发布窗口/宕机
- 团队协作成本高

最初的数个月里.....



对应用层架构做了解耦

收益

- 代码（库）解耦
- 基于业务的边界
- 技术栈的多样性
- 开发责任田清晰

挑战

- 交付周期无明显变化
- 无法独立运行部署
- 团队组织无变化
- 数据依然耦合

如何应对？

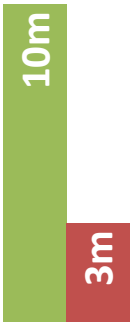
- 优化服务的拆分原则
- 增加服务实现的数量
- 引入更多的支撑组件
- ……

接下来的3个月里，
引入一系列CloudNative实践....

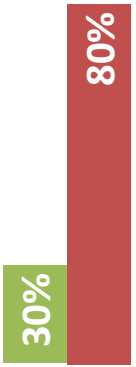
获得的收益

部署频率
+3times

LOC/提交
-63%



构建时间
-70%



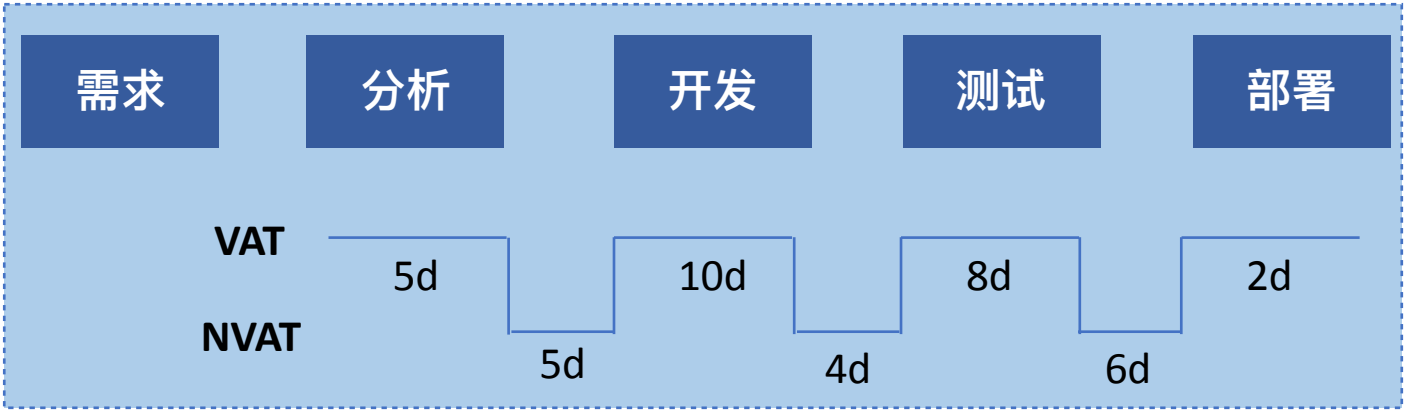
自动化测试覆盖率
+2.7times

主要的实践

1. 梳理价值流映射图
2. 构建全功能交付团队
3. 构建可视化交付流程
4. 打通端到端的流水线
5. 确定清晰的度量机制
6. 持续演进的架构拆分

实践1. 梳理价值流映射图

实践1. 梳理价值流映射图



***VAT**(Value Add Time) = 25 d*

***NVAT**(Non Value Add Time) = 15 d*

***Cycle Time**(VAT + NVAT) = 40 d*

***Efficiency**(VAT/Cycle Time) = 62.5%*

实践1. 梳理价值流映射图

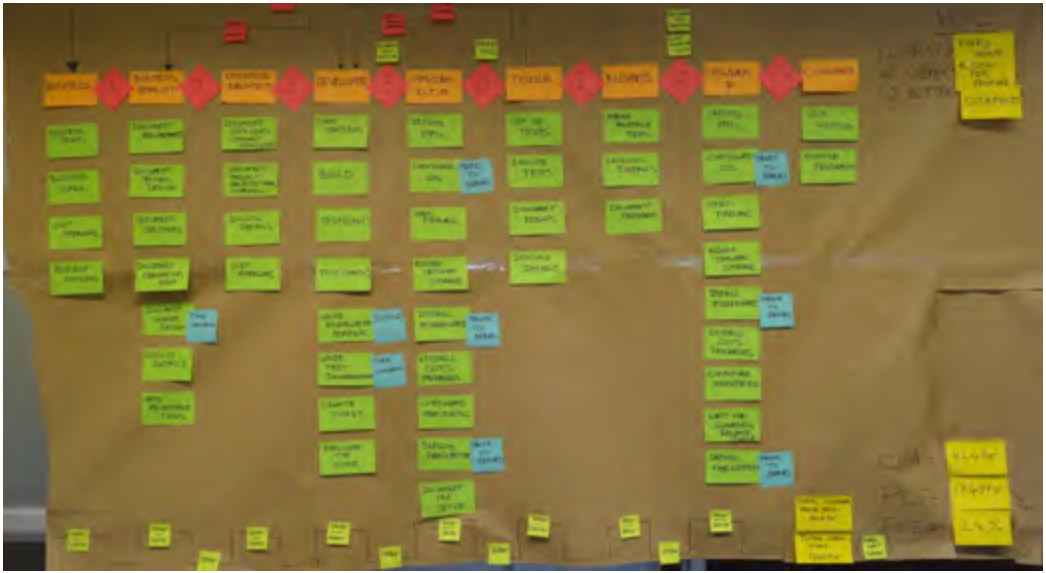
How：

1). 确定交付过程相关角色(分析/架构/开发/测试/运维)

2). 讨论并绘出交付过程中的活动

3). 识别价值流中的VAT/NVAT/Cycle Time

实践1. 梳理价值流映射图



实践2. 构建全功能团队

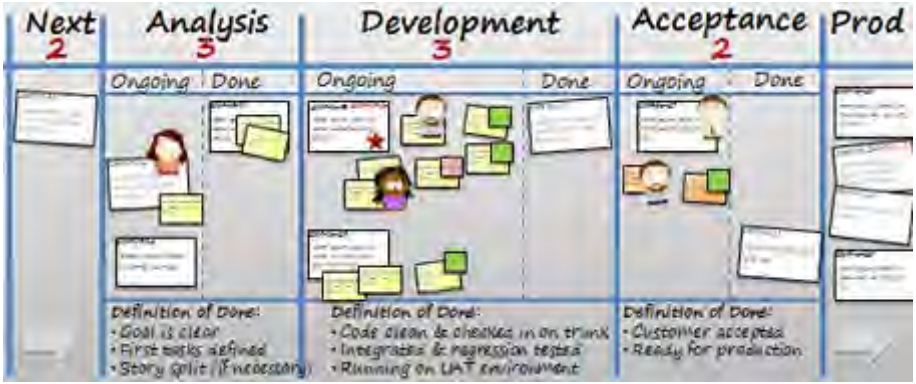
实践2. 构建全功能团队

- SDM
- BA/SE
- DEV
- QA
- Ops

识别**角色**与**成员**的关系

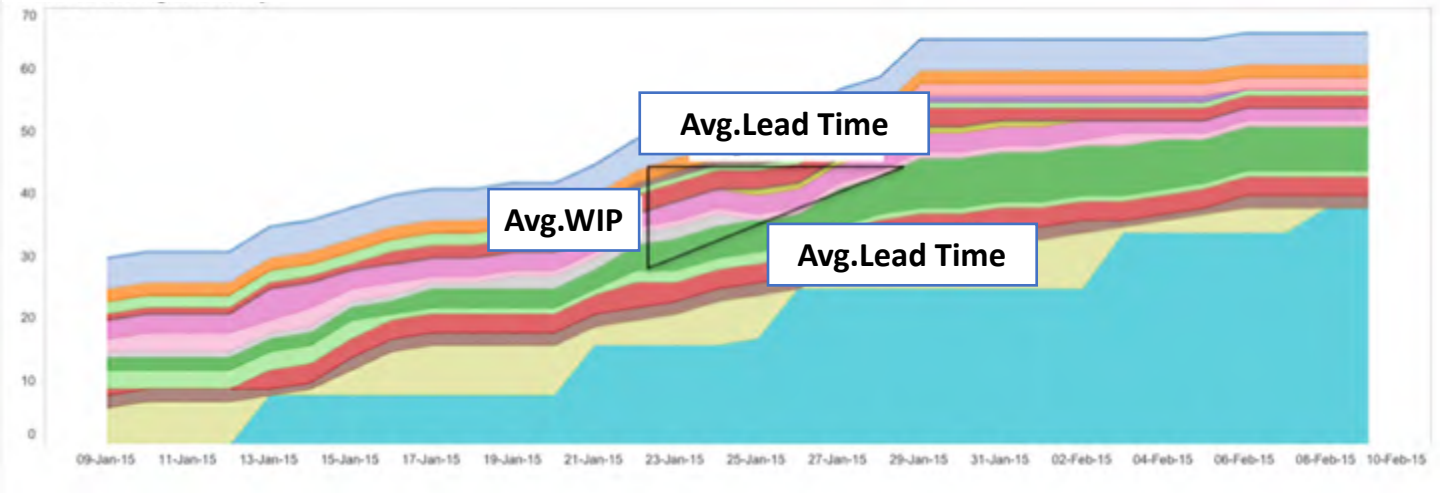
实践3. 使用看板可视化流程

实践3. 使用看板可视化流程



- 任务可视化
- 优化流动率
- 度量与反馈

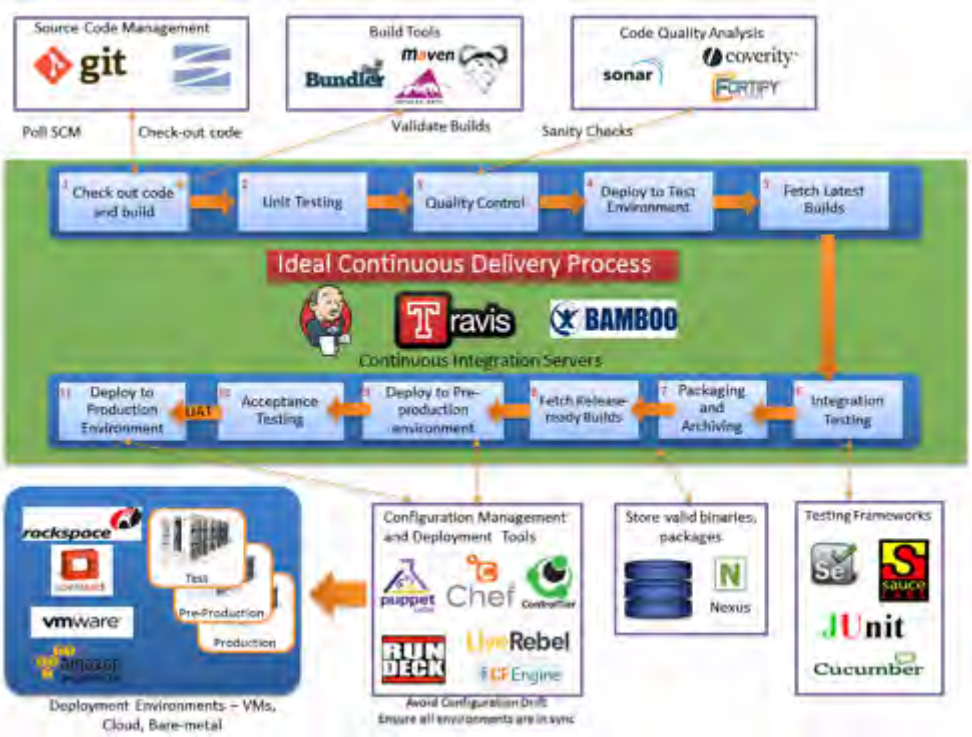
实践3. 使用看板可视化流程



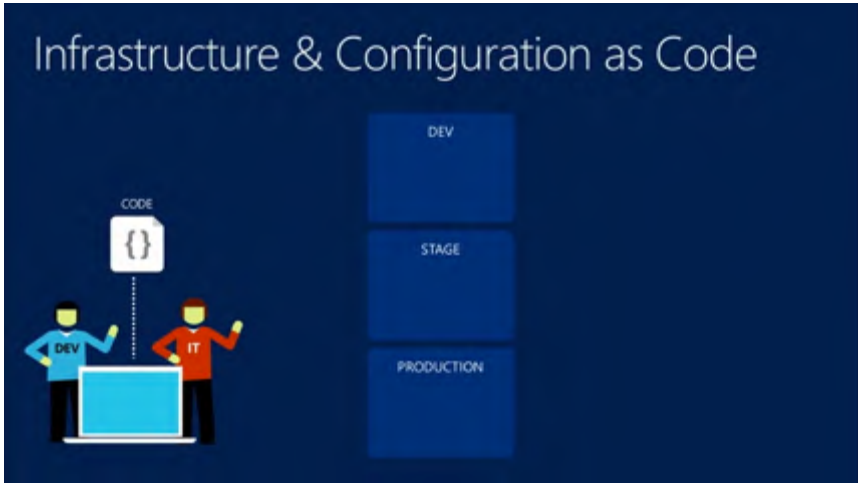
基于数据，反馈并改进

实践4. 打通端到端交付流水线

实践4. 打通端到端交付流水线



实践4. 打通端到端交付流水线



- 基于代码创建环境(Alpha/Beta/Gamma)

实践5. 清晰的度量机制

实践5. 清晰的度量机制

端到端的交付效率

代码提交频率　构建失败率

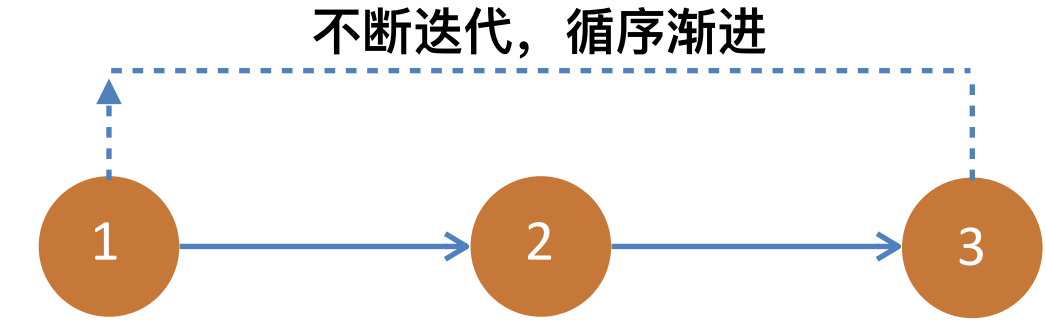
代码提交行数　服务启动时间

CI构建时间　单元测试占比

过程中的局部效率

实践6. 持续演进的服务架构

实践6. 持续演进的服务架构



定义服务
- 业务逻辑梳理
- 服务的拆分

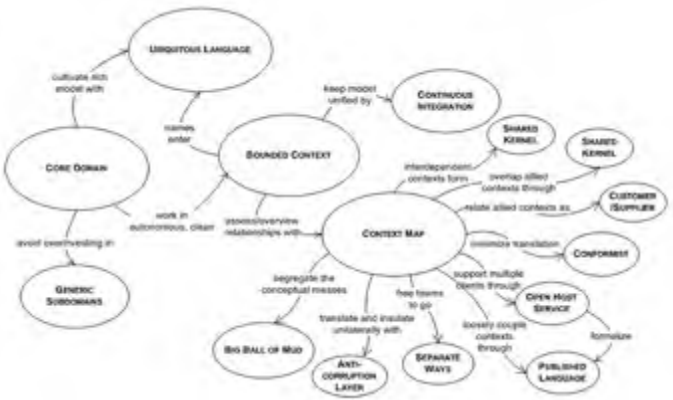
构建服务
- 技术栈选择
- 数据的解耦

双模IT运行
- 注册/发现机制
- 监控&告警信息

实践6. 持续演进的服务架构

领域驱动设计(DDD)

- **领域语言(Ubiquitous Language)**
- **领域模型(Domain Model)**
- **特定语境(Bundle Context)**
- **领域事件(Domain Event)**
- **聚合根(Aggregation Root)**



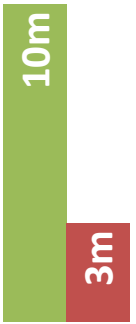
实践6. 持续演进的服务架构

1. 梳理业务流程
2. 识别领域事件
3. 识别外部命令
4. 识别聚合根
5. 拆分服务

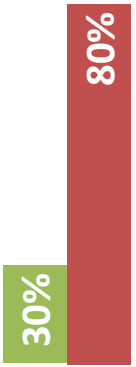
获得的收益

部署频率
+3times

LOC/提交
-63%



构建时间
-70%



自动化测试覆盖率
+2.7times

下一步……

- 增加服务实现的数量
- 引入更多的支撑组件
- 基于容器/云化基础设施

推荐阅读


持续交付


DevOps
Handbook


Release It!
Design and Deploy
Production-Ready Software


O'REILLY
Migrating to
Cloud-Native
Application
Architectures


微服务

谢谢~

Thanks

荣誉出品

高效运维社区

国际最佳实践管理联盟

想第一时间看到

高效运维社区公众号

的好文章吗?

请打开高效运维社区公众号，点击右上角小人，如右侧所示设置就好