Pivotal

云原生和微服务应用平台之旅

-- Event Driven MicroServices的落地实践

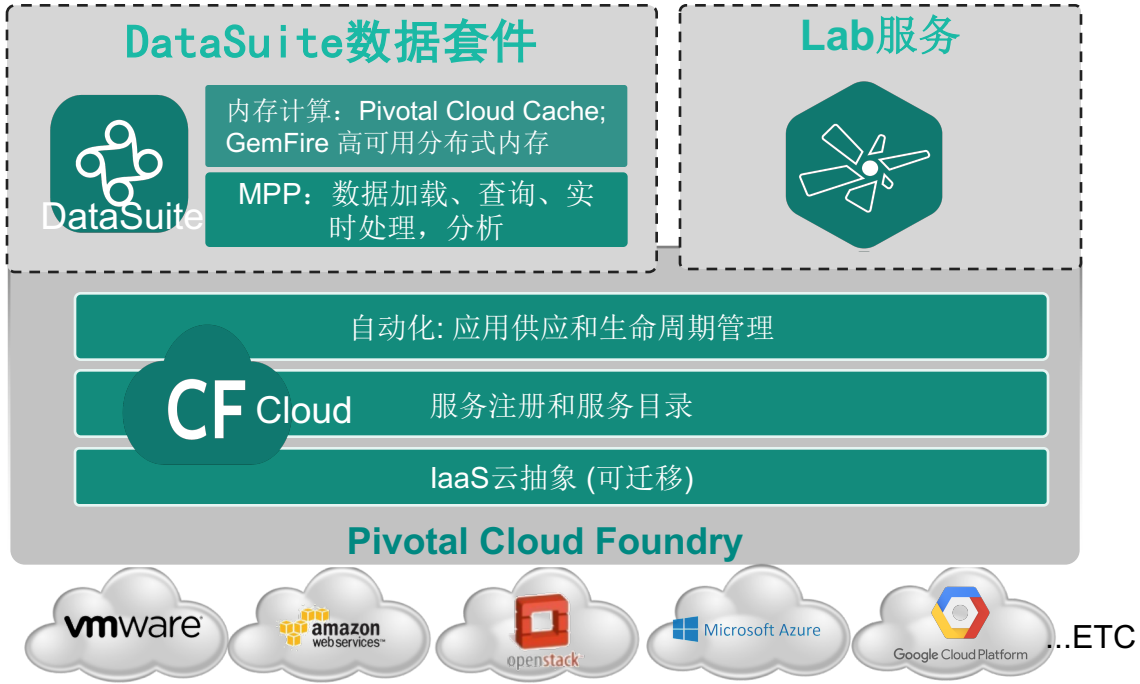
刘鹏，Peng Liu (peliu@pivotal.io)
Pivotal Senior Platform Architect
Mobile: 18500839957
[2019.9]

Agenda

- Pivotal 与云原生PaaS简介
- 微服务与领域驱动设计
- Event Driven与Spring Cloud Stream介绍
- Free Talk

Pivotal助力企业云原生之旅
WHY/ WHAT/HOW

Pivotal公司介绍



DELL vmware GE

Microsoft Ford

2018.4 IPO

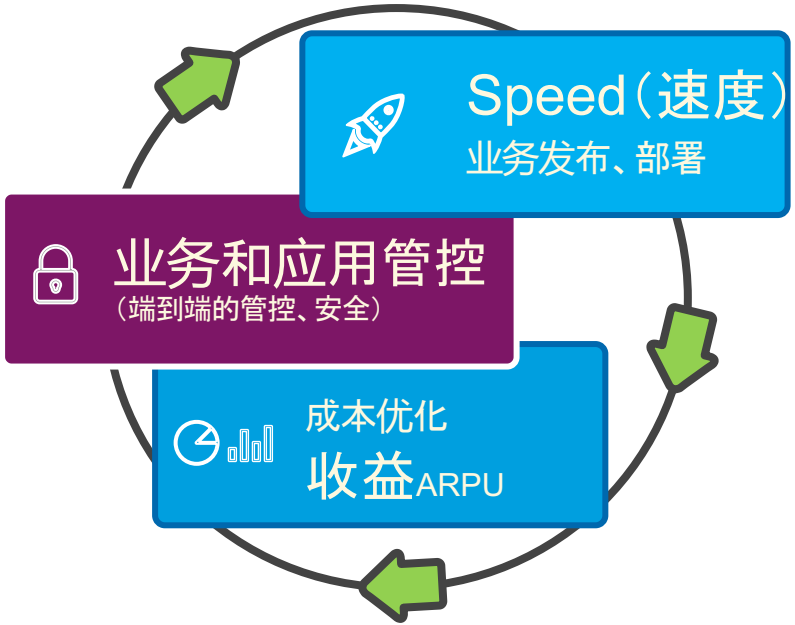
纽交所上市公司
硅谷软件公司

Pivotal对开源支持的承诺

- 所有的软件均开源，同时提供有增强的商业版。
- 开源和商业版重叠部分完全一样。
- 把互联网成功实践技术推广到企业用户
- 支持客户对技术的自主可控

Pivotal

业务系统面临的变化与挑战



- 业务部署分钟级，服务启动时间秒级；
- 业务部署频率是以往的x100倍
- 容器数量是VM的nx10倍
- 新技术转化/交货期 – 核心竞争力
- 组织将安全和合规性作为选择平台的标尺
- 组织必须考虑多云策略
- 开发与运维的比例

企业希望利用成熟的平台和新技术来应对

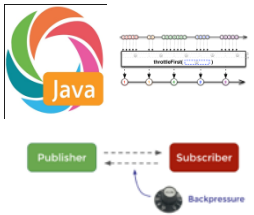
Eg. 网银，12306，个税系统mobile app快速升级和响应业务需求变化

现代IT技术的发展

随着硬件特别是多核处理器的发展和价格的下降，多任务处理已经是所有操作系统必备的一项基本功能。多核CPU的强大运算能力没有得到充分利用。

由大数据驱动的包括软硬件在内的各种分布式存储技术的快速发展，以及分布式数据库的发展与挑战

云计算技术的逐渐成熟，IaaS层随着互联网的发展，网络带宽成倍增长，虚拟化技术成熟使得按需分配计算能力成为可能。容器技术的发展让PaaS的标准化也成为可能

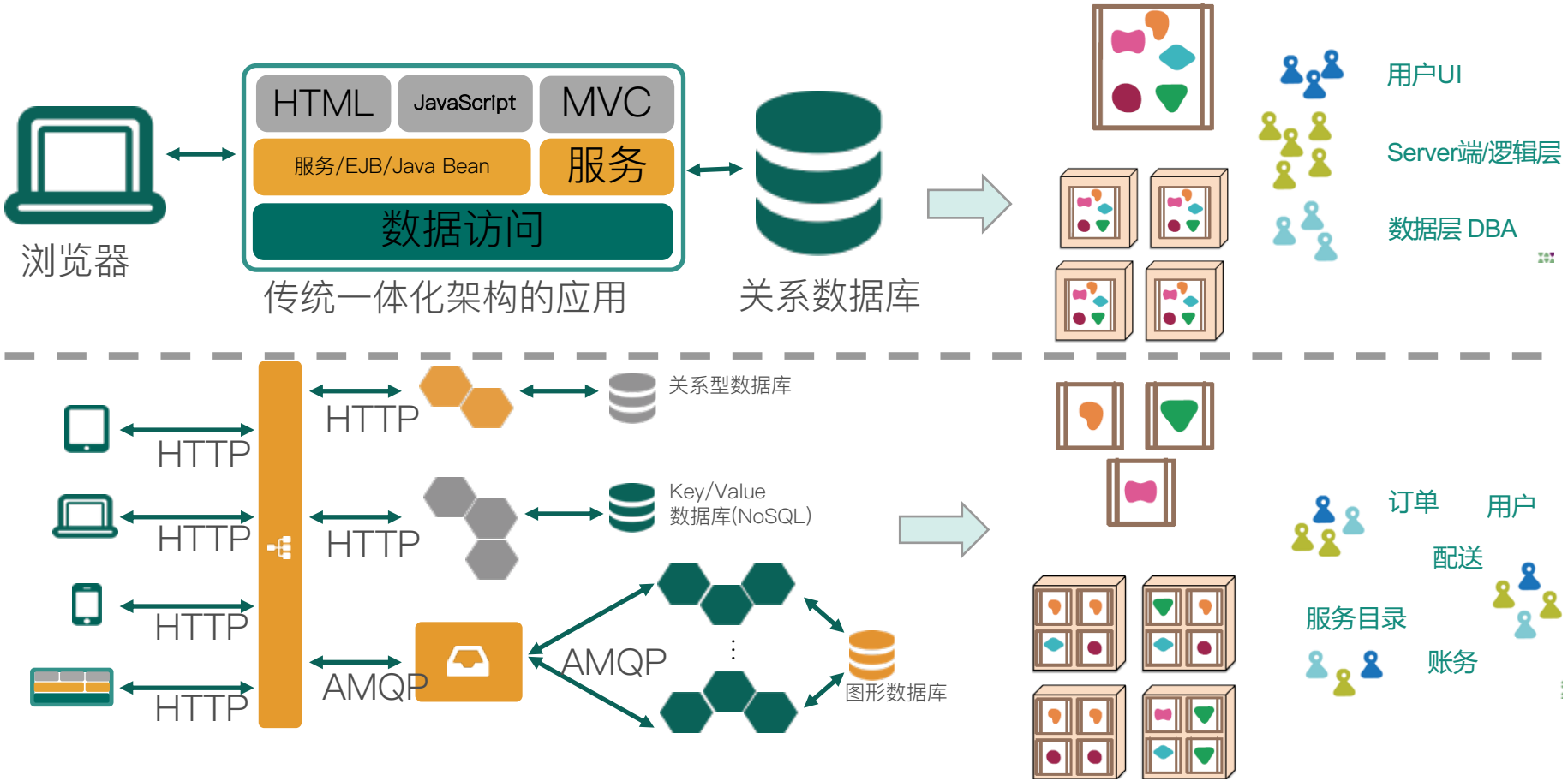


大规模分布式计算的成熟，集群/网格技术的成熟，微服务及类似架构的兴起，各种分布式组件的应用。

软硬件解耦、虚拟化，云计算，大数据，大规模分布式计算…

业务驱动促使IT从交易型向交互型转换：
To 'Build', not just 'Buy'

传统单体应用架构 VS 微服务架构 (I)



传统一体化架构 VS 微服务架构 (II)

一体化架构	微服务架构
✓ 构建复杂/ 开发相对简单	✓ 构建简单/开发有挑战性
✓ 整个系统是一个**不可分割的整体**	✓ **微粒度**，灵活应变
✓ 模块的**依赖性**由开发语言/框架/中间件决定	✓ **松耦合**，服务调用与开发语言和框架解耦
✓ 应用更新周期紧耦合/ **无法快速迭代和部署**	✓ 应用更新周期松耦合/ 可以**频繁迭代部署**
✓ 伸缩能力有限，效率低下	✓ 有效的**高可伸缩性**
✓ 无法有效应对扩展性开发	✓ 具备**可扩展**性开发
✓ 需要对于选定的**技术路线的长期投资**	✓ **消除**了对于**技术路线的依赖性**

云原生应用定义

2015年Pivotal公司的Matt Stine发布叫做**迁移到云原生应用架构**的文章，第一次提出了云原生（Cloud Native）的概念。Pivotal 不但是云原生应用的提出者，而且还推出了Pivotal Cloud Foundry 云原生应用平台和 Spring 开源 Java 开发框架，成为云原生应用架构中先驱者和探路者。
Pivotal最初的定义，其中探讨了云原生应用架构的几个主要特征：
（1）符合12因素应用；（2）面向微服务架构；（3）自服务敏捷架构；（4）基于API的协作；（5）抗脆弱性等。

Pivotal

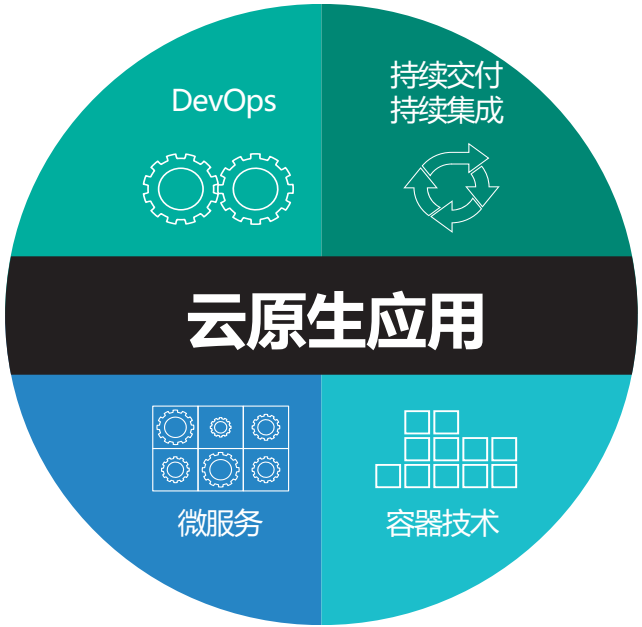
CNCF ToC: CNCF Cloud Native Definition

云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。云原生的代表技术包括**容器、服务网格、微服务、不可变基础设施**和**声明式API。**

这些技术能够构建容错性好、易于管理和便于观察的松耦合系统。结合可靠的自动化手段，云原生技术使工程师能够轻松地对系统作出频繁和可预测的重大变更。

云原生计算基金会（CNCF）致力于培育和维护一个厂商中立的开源生态系统，来推广云原生技术。

何为云原生架构？



借助平台的全面自动化能力
跨多云
构建微服务
持续交付部署生产系统

Building Microservices
To Continuously Deploy on Production
Across Multi-Cloud
Fully Automated by Platform

Pivotal 应用生命周期DevOps, CI/CD工具链和生态圈

自动化.
从测试到构建到部署的集成的工具和自动化的流程

高速.
以小功能更新更频繁的构建产品版本，降低复杂度，加速进入市场

质量.
通过测试驱动的开发减少反馈循环，使得问题浮现的更快，更具响应性

敏捷性.
产品更新形成规律，而且在线升级，无业务停顿时间，改进客户体验，加上产品进入市场

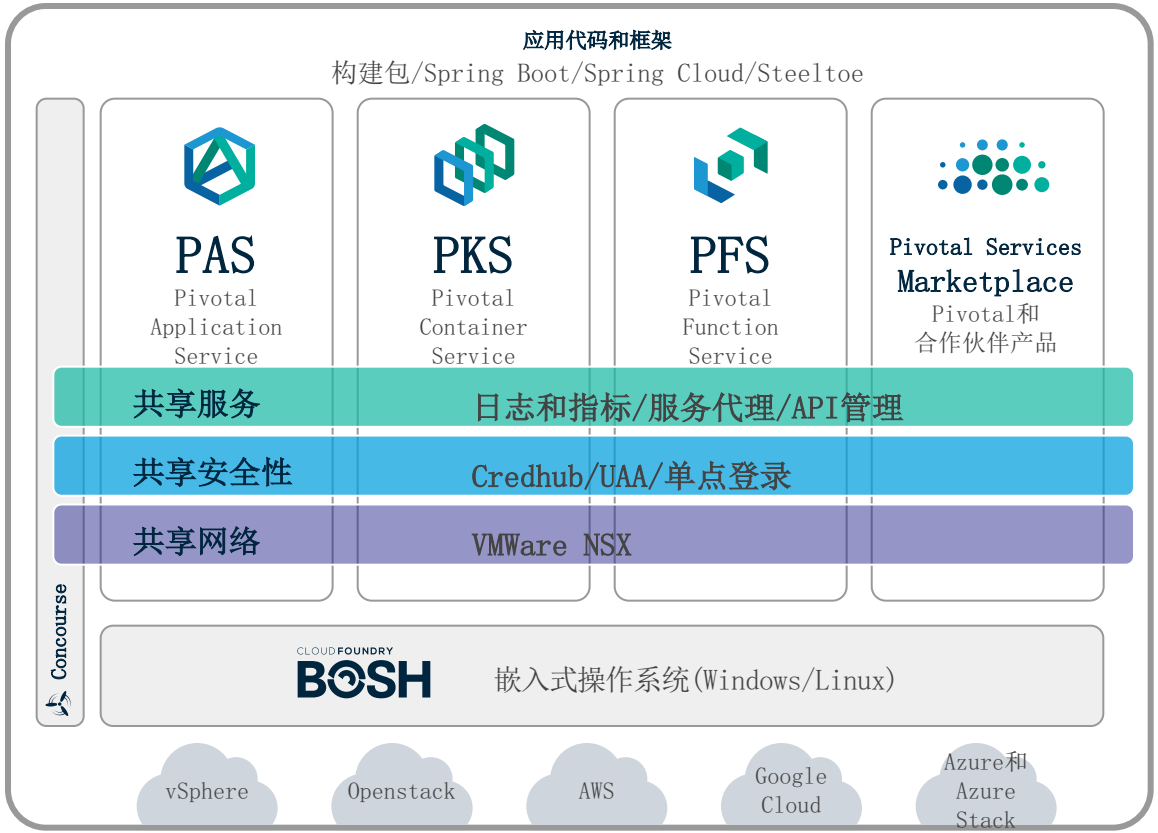
构建管道运维

提交修改的代码	自动的构建和测试(单元测试、静态代码分析)	代码和构建包仓库	自动集成测试	验收测试、性能测试、压力测试	应用在线升级，无业务停顿
GitLab GitHub	concourse Jenkins CloudBees The Enterprise Jenkins Company	JFrog artifactory	Pivotal CF Development	Pivotal CF Test + UAT + Staging	Pivotal CF Production
分布式的版本控制、源代码管理、协调开发	以增量的方式持续的构建和测试软件项目，有数百种插件	共享的二进制库，管理库的生命周期，避免冲突	在同一环境下做开发、测试、QA和生产，开发人员友好的命令行和API,DevOps简化每个应用的运维，内置的生态环境提供大量的服务。跨越多种不同的IaaS		

化繁为简 - 成熟先进的平台和技术

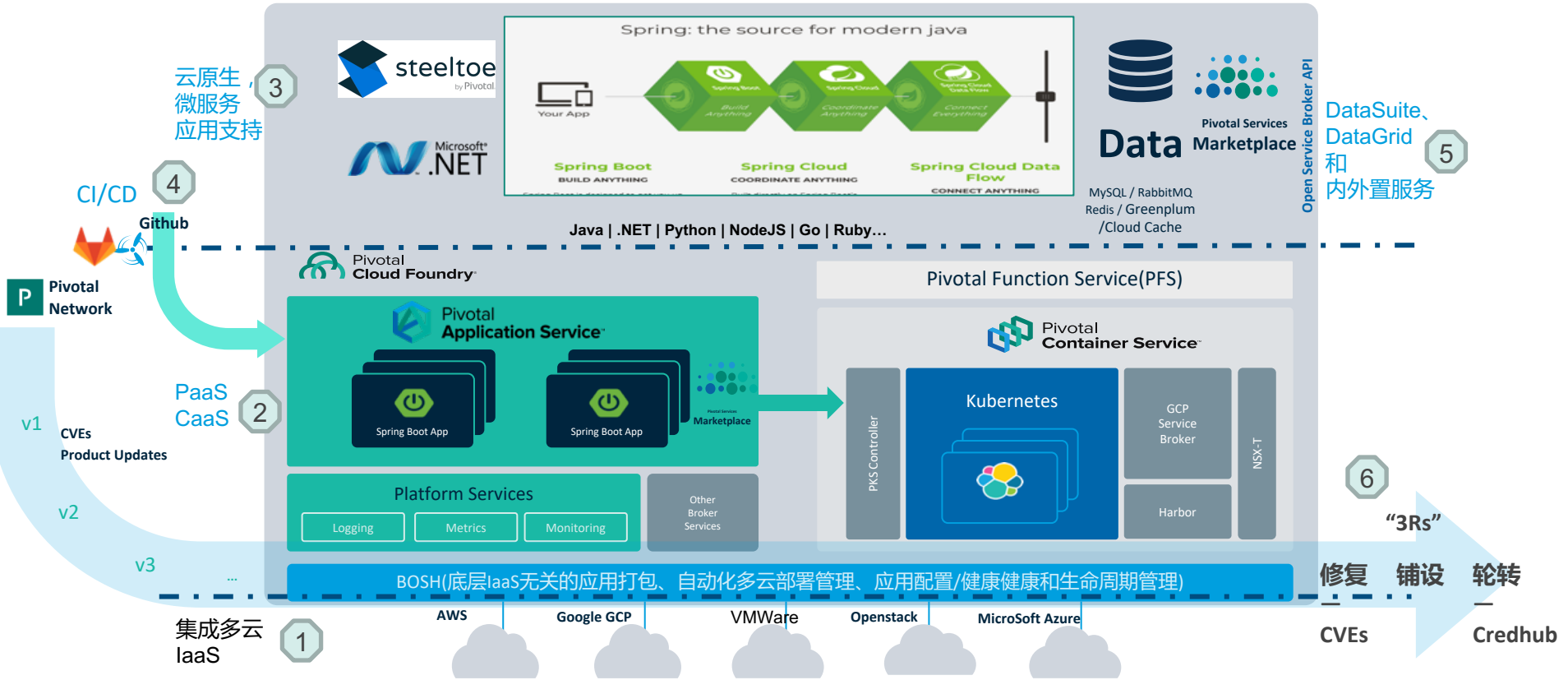
任意应用
每个云
一个平台

Pivotal Cloud Foundry



Pivotal

Pivotal 助力企业快速构建云原生应用



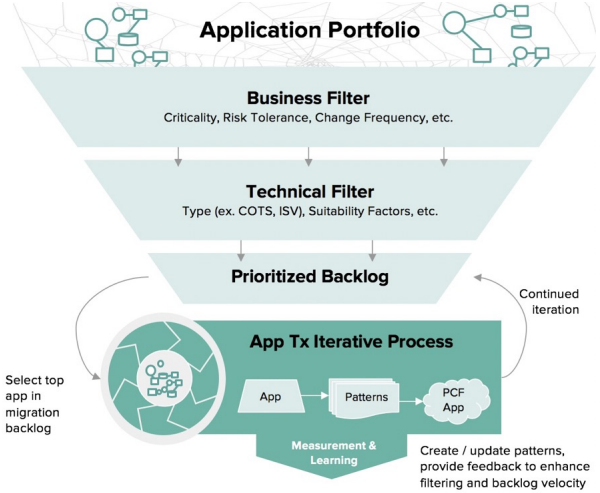
Pivotal

Pivotal App-Tx Service

Application Transformation Funnel

Use of tooling and templates to quickly make effective candidate selection, constantly prioritize work and continuously share feedback to accelerate future efforts

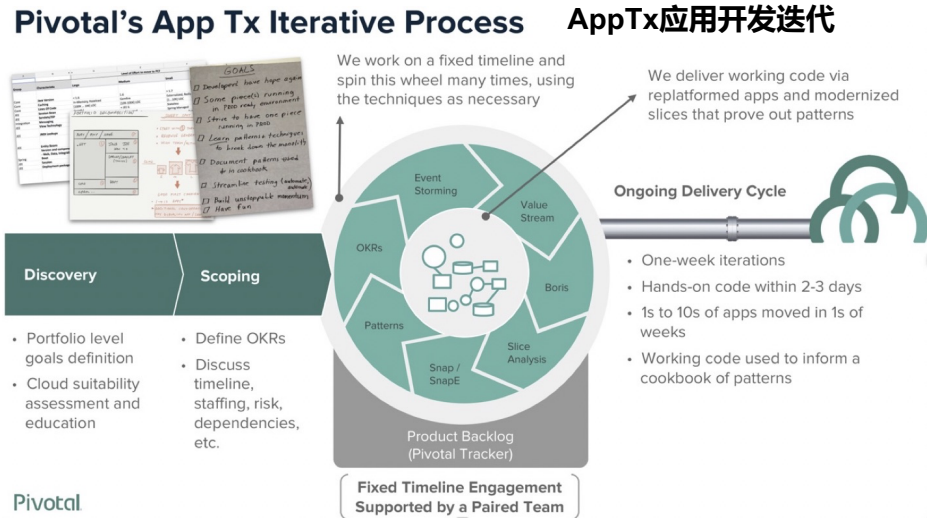
Pivotal



应用转型Funnel

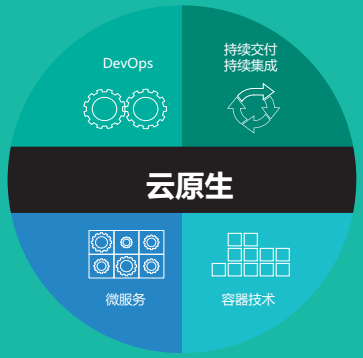
- **业务filter**
- **技术filter**

Start -> Ingrain -> Scale
启动 -> 稳 固 -> 拓展



为企业转型全面助力

组织文化、最佳实践和方法论；
工具和平台



Pivotal

➢技术变革

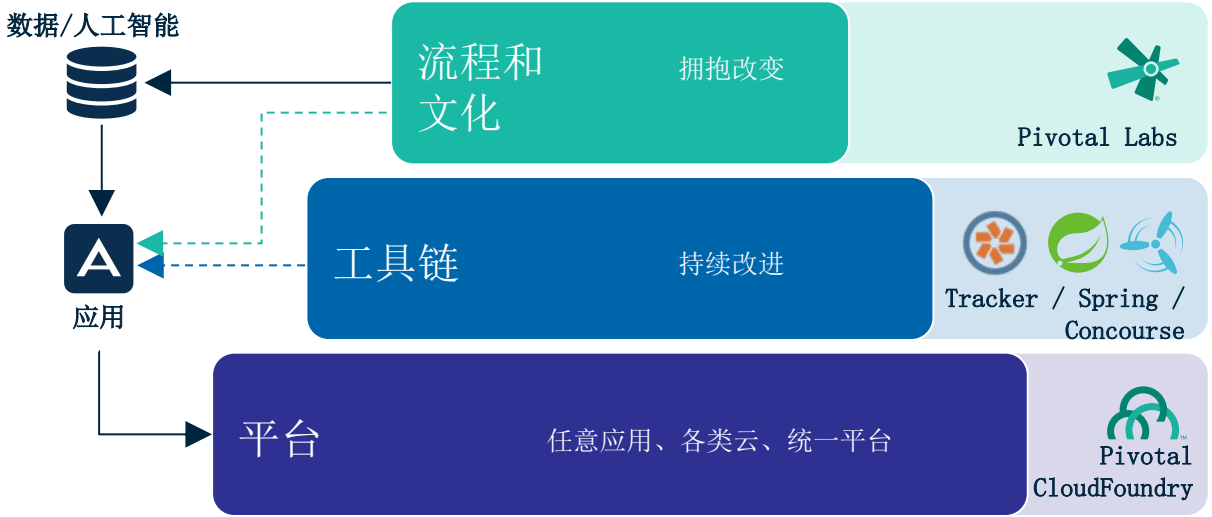
- 分解巨石应用
- 分解的数据
- 容器化
- 解除代码耦合，应用耦合

➢组织变革

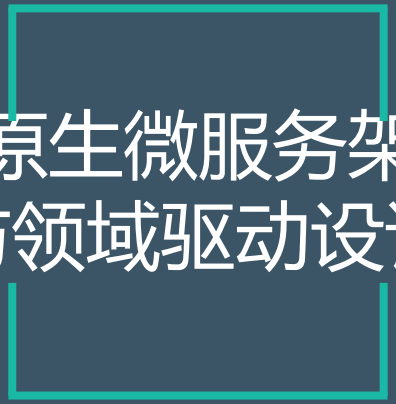
- 按业务功能划分的团队 – 可独立部署的服务
- 平台运营团队

➢文化变革

- 从Silos到DevOps
- 从间断平衡到持续交付
- 集中管理到去中心化自治



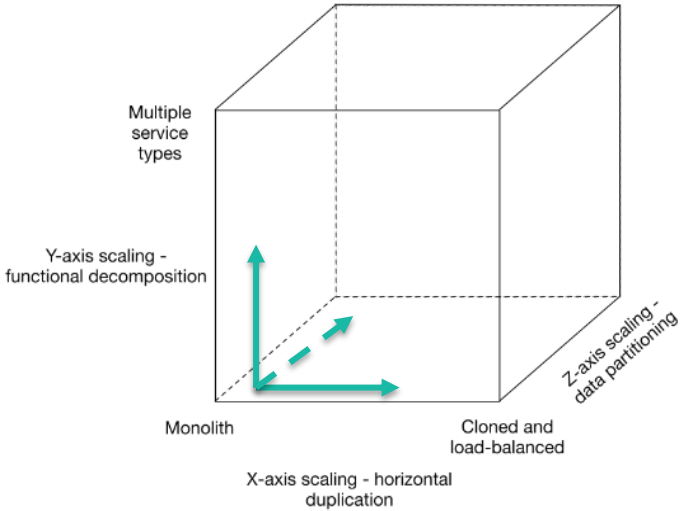
云原生微服务架构与领域驱动设计



Pivotal

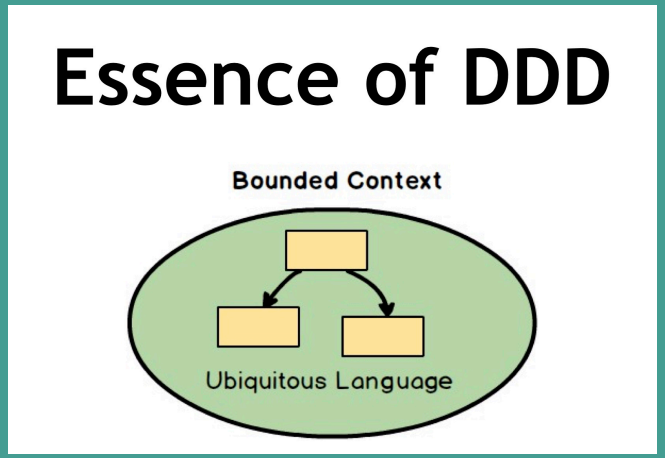
微服务设计核心:业务的扩展性，并行性，快速迭代部署

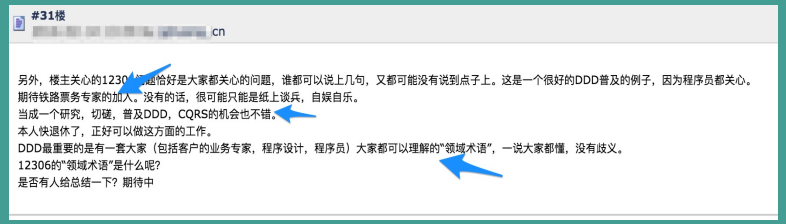
- DDD领域驱动模型 – 业务模型驱动
- 扩展性立方模式
  - X轴 横向扩展模式 – 实例复制扩展
  - Y轴 功能分解模式 – 基于功能的微服务化
  - Z轴 数据分区模式 – 数据分区
- ES（事件源）+ CQRS(读写责任分离)



与具有成功经验的合作伙伴共同完成包括组织文化、开发过程管理、应用系统的规划，选择，拆分，构建在内的最佳实践！！

DDD – 领域驱动设计





领域和业务边界

- Domain(领域): 与某个特定问题相关的知识领域和行为。
- Bounded Context：领域边界的上下文。明确定义模型适用的上下文（业务边界）。根据团队组织，应用程序特定部分的使用情况以及代码库和数据库模式等物理表现形式来明确设置边界。保持模型在这些范围内严格一致，但不要被外界的问题分心或混淆。

聚合和聚合根

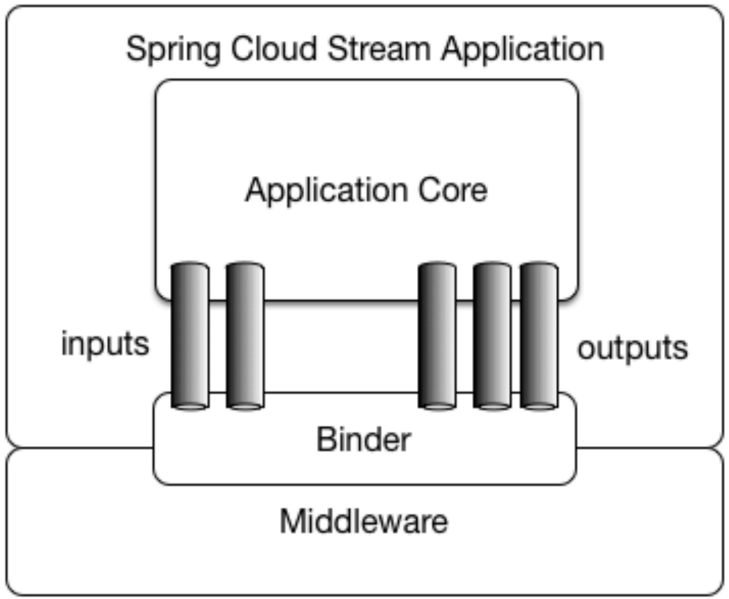
- 聚合：一组具有内聚关系的相关对象的集合，是一个修改数据的最小原子单元。聚合通常使用id访问。
- 聚合根：每个聚合都有一个根对象，根对象管理聚合内的其他子对象（实体、值对象）；聚合之间的交互都是通过聚合根来交互，不能绕过聚合根去直接和聚合下的子实体进行交互。

Event Driven
Spring Cloud Stream介绍

Pivotal

Spring Cloud Stream源语（1）

- **Binder**抽象
- 持久的订阅-发布模型支持
- 消费者组支持
- 分区支持
- 可拔插的**Binder APIs**



- 部署人员可以在运行时动态选择通道连接destination（例如，**Kafka**的**topic**或者**RabbitMQ**的**exchange**），通过外部配置的属性和Spring Boot支持的任何形式来提供（包括应用启动参数、环境变量和application.yml或者application.properties文件）。
- **Spring Cloud Stream**是基于Spring Integration的，Stream完全继承了Integration的基础和基础设施以及组件本身。使用 @StreamListener annotation分发消息到不同方法。
- **Spring Could Stream**还支持使用reactive API，将流入和流出数据作为连续数据流处理。

```
@StreamListener(target = Sink.INPUT, condition = "headers['type']=='foo'")
public void receiveFoo(@Payload FooPojo fooPojo) {      // handle the message    }
@StreamListener(target = Sink.INPUT, condition = "headers['type']=='bar'")
public void receiveBar(@Payload BarPojo barPojo) {      // handle the message    }
```

Spring Cloud Stream源语（2）

- **Binder抽象**
- 持久的订阅-发布模型支持
- 消费者组支持
- 分区支持
- 可拔插的**Binder APIs**

支持持久的发布-订阅模式：应用间的通信遵循发布-订阅模式，在这个模式中，数据通过共享的主题进行广播。可以在图中看到一组相互作用的**Spring Cloud Stream**应用的典型部署。

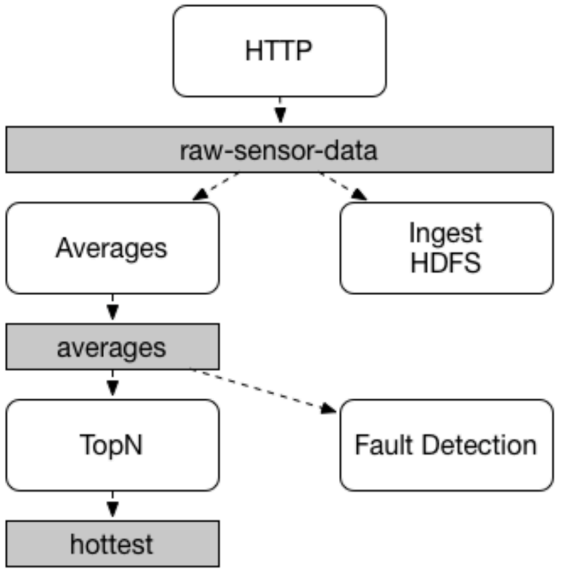


Figure 2. Spring Cloud Stream Publish-Subscribe

Spring Cloud Stream源语（3）

- **Binder抽象**
- 持久的订阅-发布模型支持
- 消费者组支持
- 分区支持
- 可拔插的**Binder APIs**

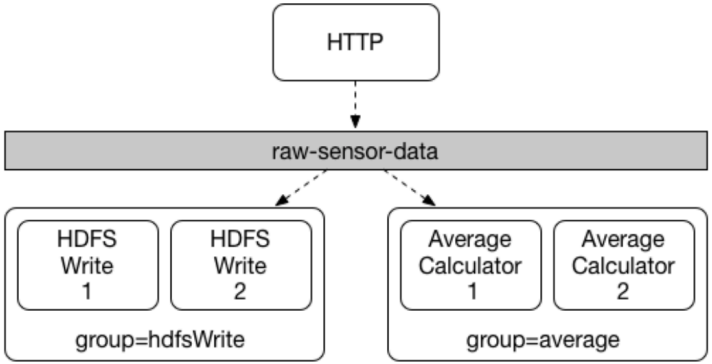


Figure 3. Spring Cloud Stream Consumer Groups

消费者组支持：

spring.cloud.stream.bindings.<channelName>.group=hdfsWrite

或者

spring.cloud.stream.bindings.<channelName>.group=average

的消费者。

订阅给定**destination**的所有分组会接收到发布消息的一个副本，但是在每个组中，只有一个成员会从**destination**接收到这个消息。默认情况下，没有指定组的时候，**Spring Cloud Stream**会将应用程序分配到一个匿名、独立且单一的消费者组，这个消费者组与所有其他消费者组都处于同一个发布-订阅关系中

消费者组订阅是持久的。也就是说，**binder**实现确保组订阅是持久的，一旦一个组中创建了一个订阅，就算这个组里边的所有应用都挂掉了，这个组也会收到消息

Spring Cloud Stream源语（4）

- **Binder抽象**
- 持久的订阅-发布模型支持
- 消费者组支持
- 分区支持
- 可拔插的**Binder APIs**

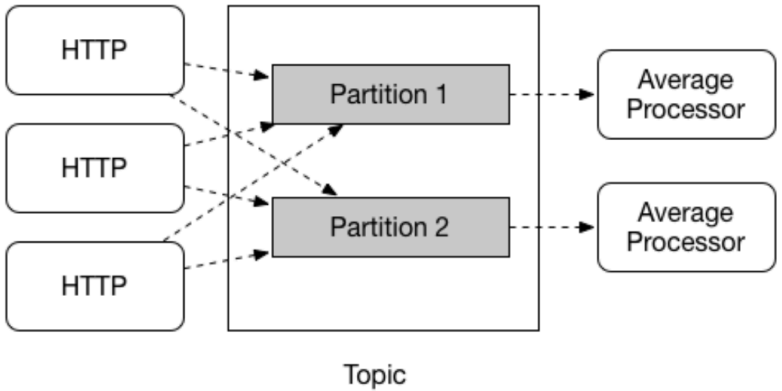
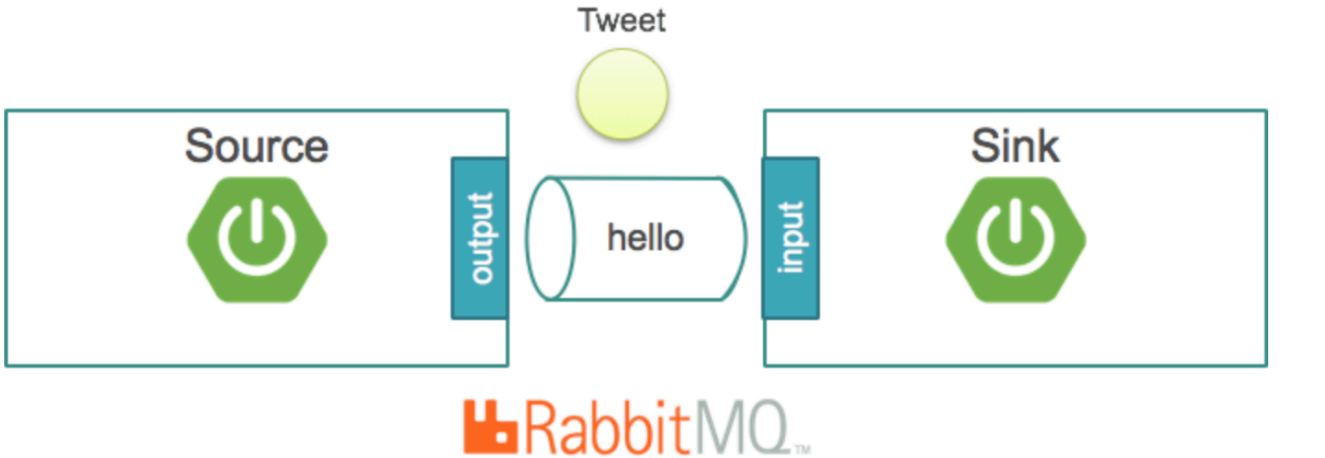


Figure 4. Spring Cloud Stream Partitioning

分区支持：

Spring Cloud Stream支持在一个应用程序的多个实例间分区数据。在分区场景中，物理通信媒介（例如：代理topic）被视为是结构化的多个分区。一个或多个生产者应用实例发送消息到多个消费者应用实例，并确保通过共同特征标识的数据由相同的消费者实例处理。

示例



Reference books:

- Domain-Driven Design: Tackling Complexity in the Heart of Software 1st Edition
- Building Microservices: Designing Fine-Grained Systems 1st Edition

Transforming How The World Builds Software