荔枝微课

专注大众知识分享的平台

荔枝微课基础架构的演进与实践

演讲人：王诚强

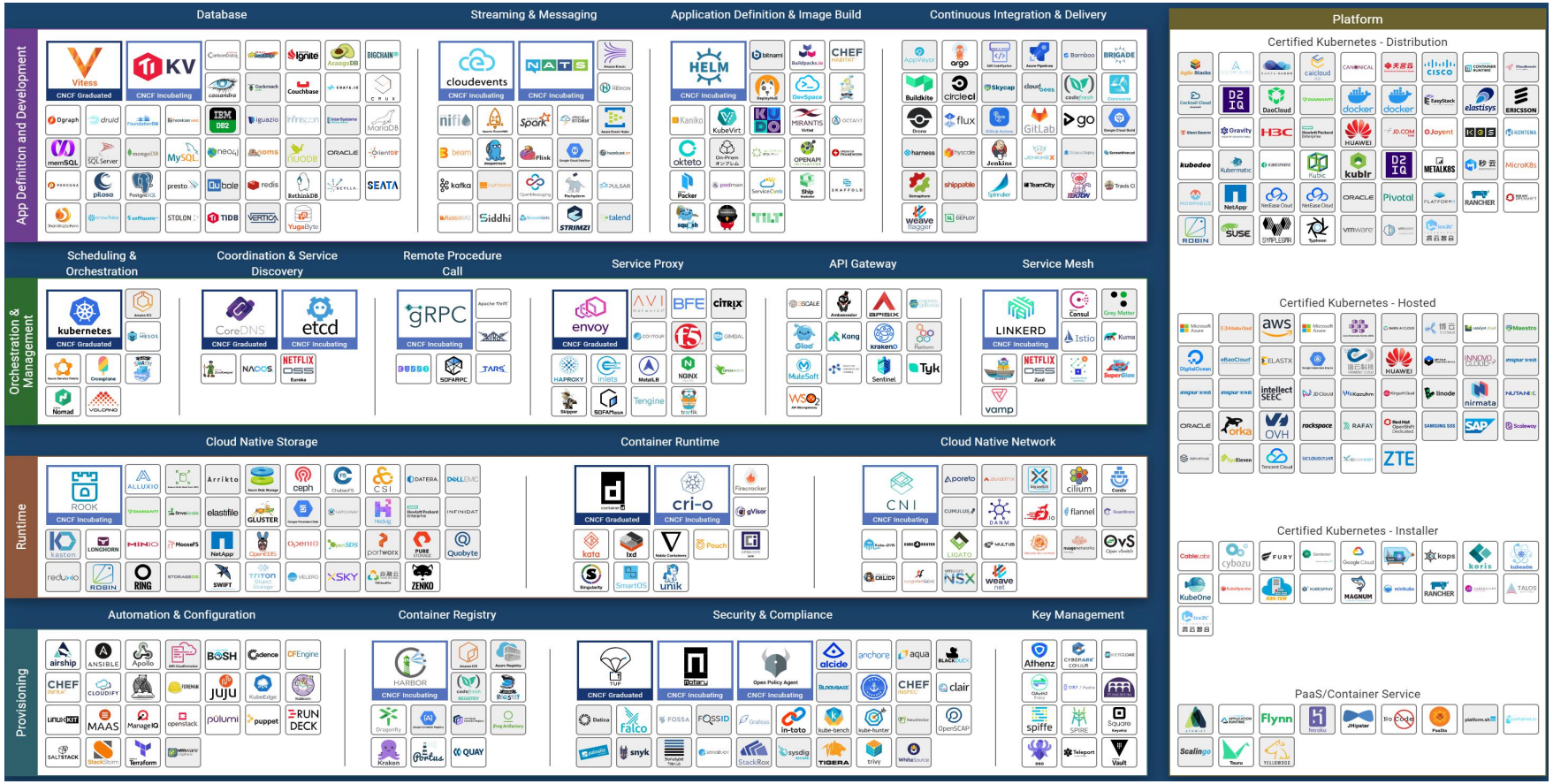
2020.8.8

个人介绍

2018年加入荔枝微课，现为基础架构负责人，主要从事基础技术研究开发、基于云原生的基础架构设计以及基础架构团队的管理建设。致力于在云原生理念下，以微服务搭建中台。

曾负责蒲公英知识商城项目研发；
负责基础架构团队的管理建设；
负责DevOps平台的架构设计与开发；
负责监控告警分析体系与值班、演练制度的建设；
负责集群分布式压测系统的设计与开发；
负责配置中心、Kubernetes、Istio等云原生规划建设；
负责老项目集群化改造的统筹规划；
负责效能平台的架构设计与开发；

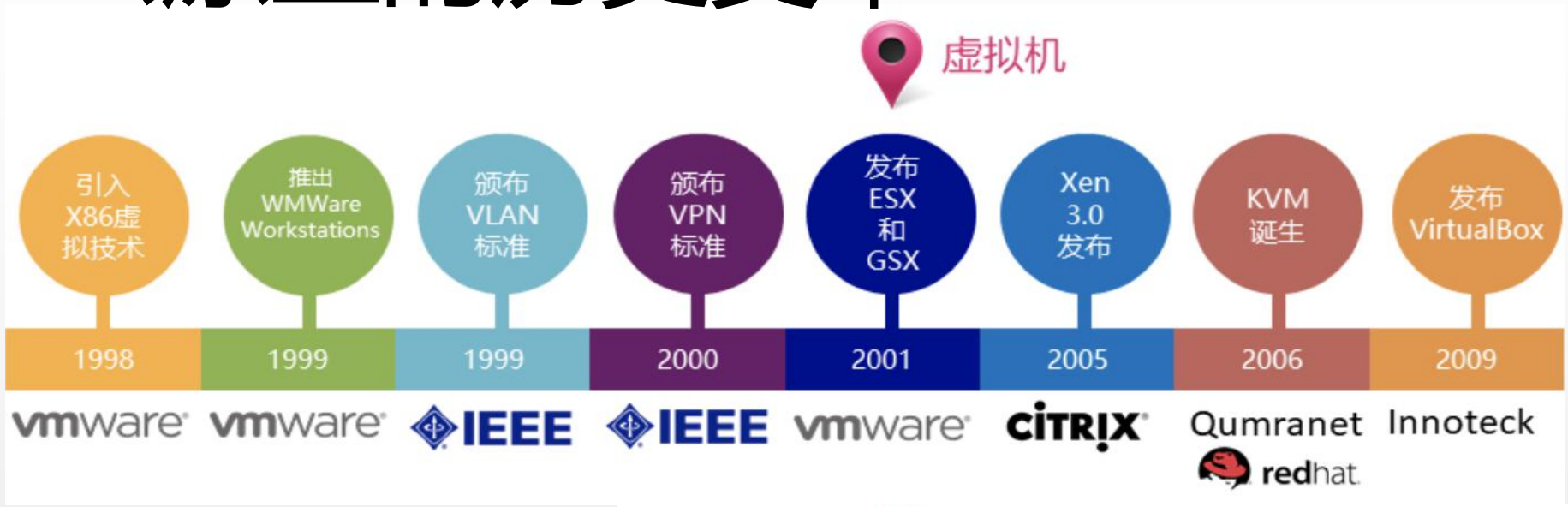
云原生

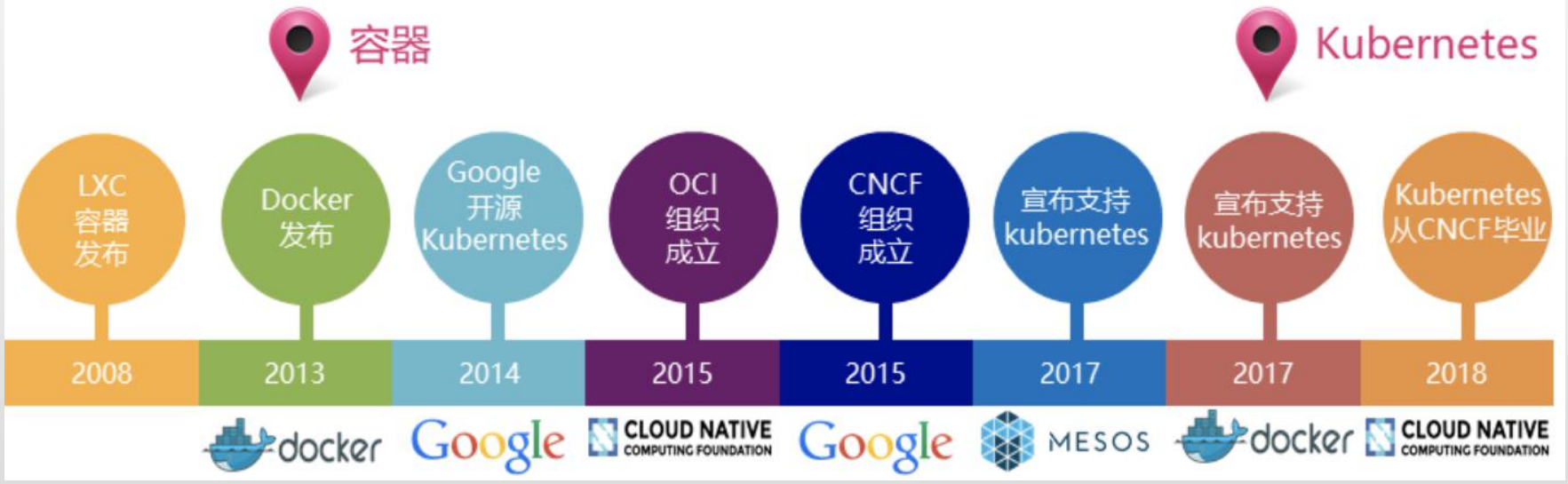


云原生（Cloud Native）是未来架构的演化方向，包含了一组应用的模式，用于帮助企业快速，持续，可靠，规模化地交付业务软件。云原生计算基金会（Cloud Native Computing Foundation，简称CNCF）是致力于云原生的开源软件基金会。

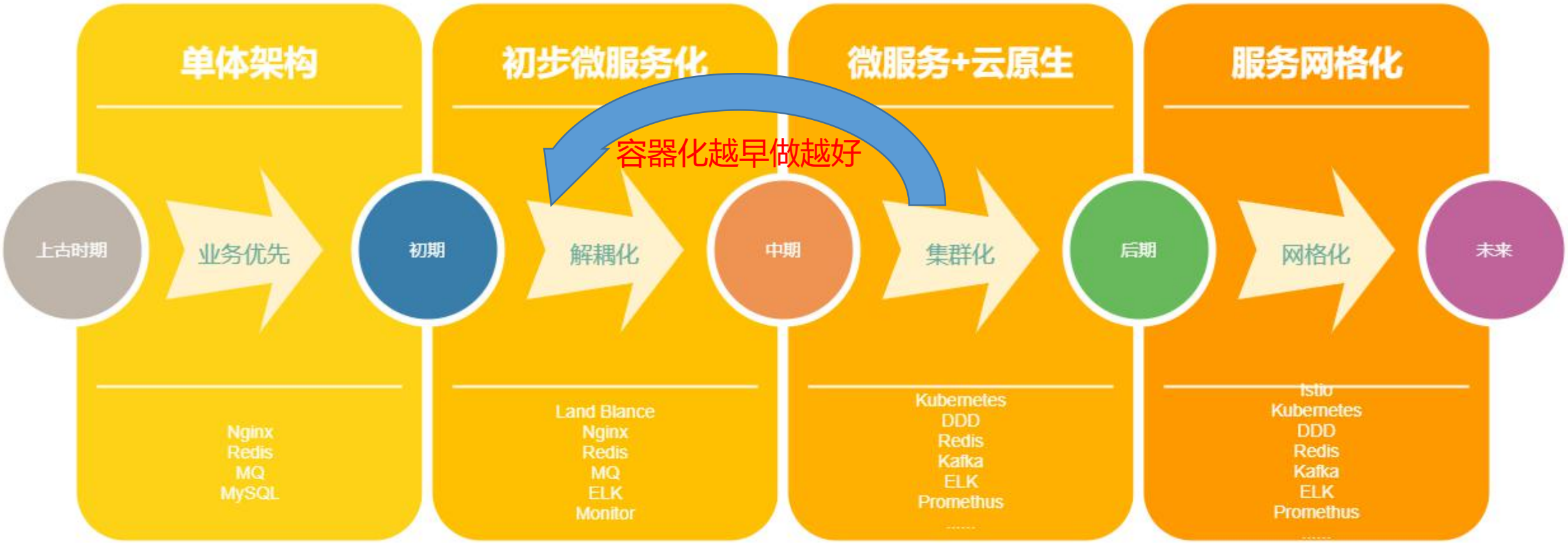
云原生由微服务架构，DevOps和以容器为代表的敏捷基础架构组成。它是一种文化一种理念，也是一种生态，既包括技术（微服务、敏捷基础设施k8s），也包括管理（devops、持续交付），包括的范围极其广泛，总得来讲是一种围绕云计算时代的架构。

云原生的历史变革


虚拟机
引入X86虚拟技术
1998
vmware
推出WMWare Workstations
1999
vmware
颁布VLAN标准
1999
IEEE
颁布VPN标准
2000
IEEE
发布ESX和GSX
2001
vmware
Xen 3.0发布
2005
CITRIX
KVM诞生
2006
Qumranet
redhat
发布VirtualBox
2009
Innoteck


容器
Kubernetes
LXC容器发布
2008
Docker发布
2013
docker
Google开源Kubernetes
2014
Google
OCI组织成立
2015
CLOUD NATIVE COMPUTING FOUNDATION
CNCF组织成立
2015
Google
宣布支持kubernetes
2017
MESOS
宣布支持kubernetes
2017
docker
Kubernetes从CNCF毕业
2018
CLOUD NATIVE COMPUTING FOUNDATION

微课架构的演进历程


单体架构
初步微服务化
微服务+云原生
服务网格化
容器化越早做越好
上古时期
业务优先
初期
解耦化
中期
集群化
后期
网格化
未来
Nginx
Redis
MQ
MySQL
Land Blance
Nginx
Redis
MQ
ELK
Monitor
Kubernetes
DDD
Redis
Kafka
ELK
Promethus
Istio
Kubernetes
DDD
Redis
Kafka
ELK
Promethus

上古时期

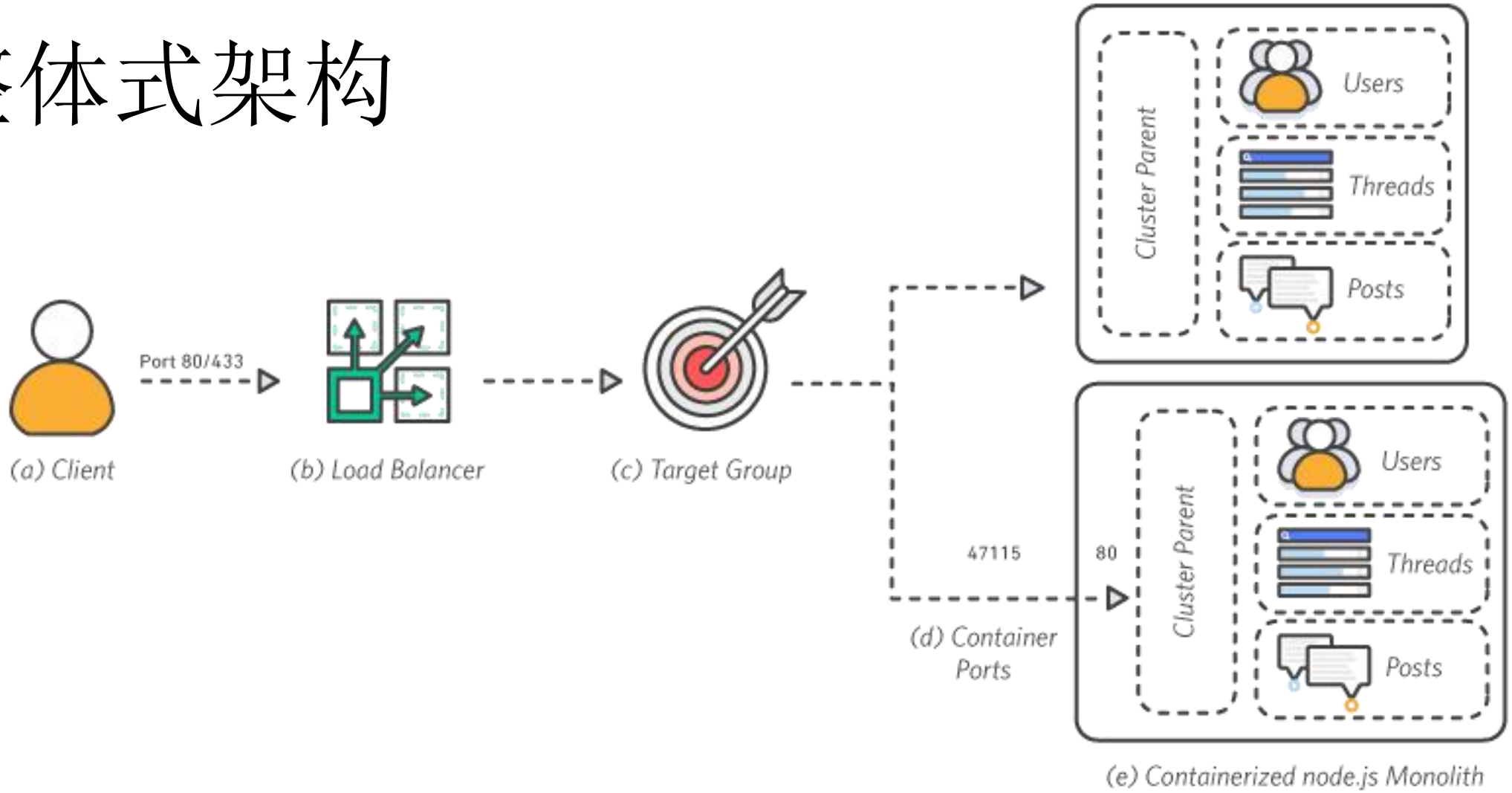
优点：

1. 业务起步快；
2. 部署维护简单；

缺点：

1. 项目越来越复杂，耦合度高；
2. 人员要求高，新人不易快速上手；
3. 不易扩展新功能；
4. 局部BUG影响整体；
5. 重复造轮子；

整体式架构


Port 80/433
(a) Client
(b) Load Balancer
(c) Target Group
47115
80
(d) Container Ports
Cluster Parent
Users
Threads
Posts
Cluster Parent
Users
Threads
Posts
(e) Containerized node.js Monolith

初期

要不要拆，怎么拆的问题
拆分后又会面临哪些问题

优点：

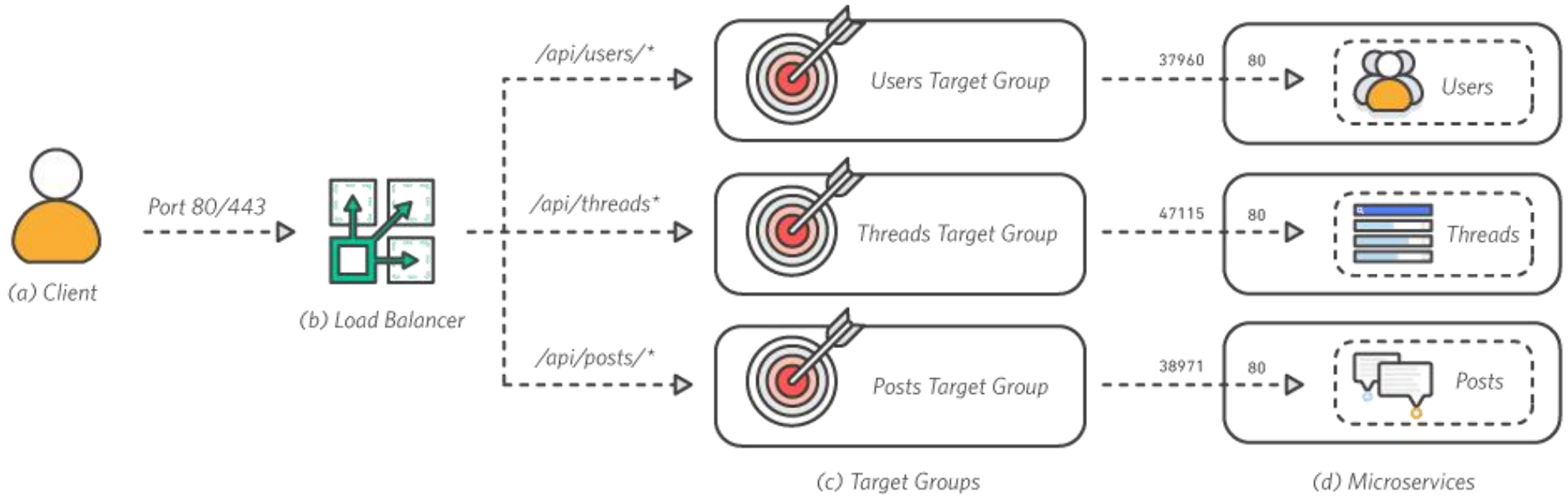
1. 松耦合，新人易上手；
2. 具备一定的监控能力；
3. 具备一定的日志收集能力；
4. 具备一定的错误收集能力；
5. 具备一定的告警能力；
6. 具备一定的分析定位能力；
7. 具备一定的资源扩缩能力；

缺点：

1. 业务项目多、产品迭代快；
2. 团队人员变更频繁，组织松散；
3. 多个技术栈；
4. 部署方式五花八门；
5. 代码前期遗留问题；
6. 服务关系不明确；
7. 配置和代码混合；
8. 重复开发；

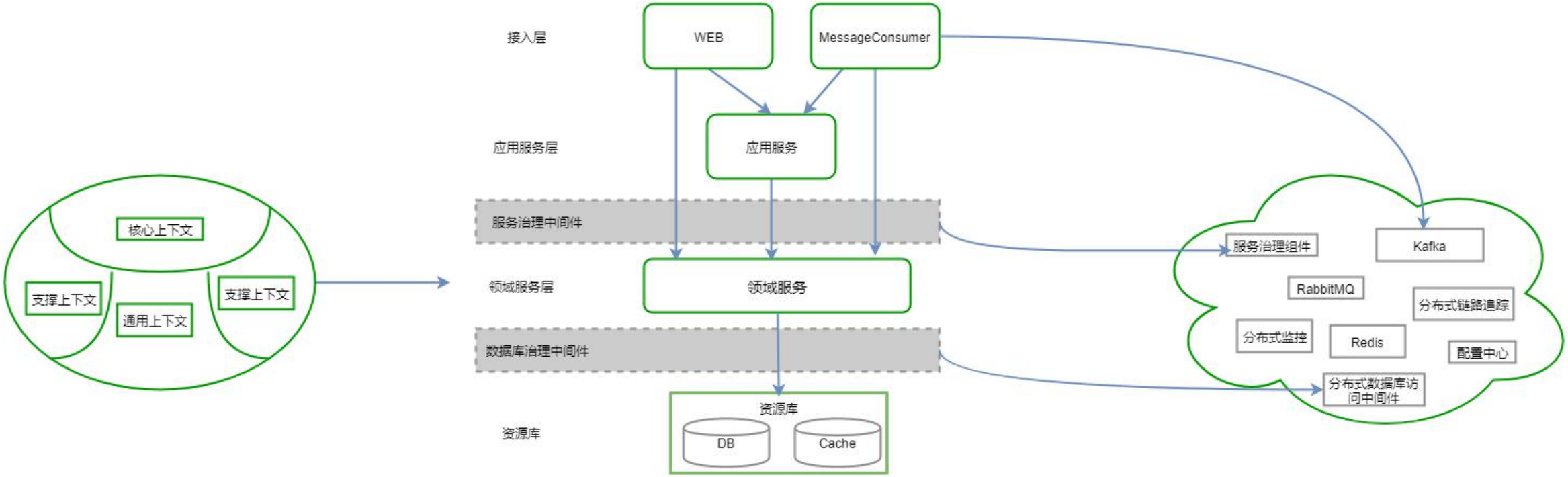
即有一定的兜底能力，但没有容器化，没有统一起部署方式
可以看到拆分后带来的第一个麻烦是对服务的部署、监控、管理有更高要求

拆分整体式架构


Port 80/443
(a) Client
(b) Load Balancer
/api/users/*
/api/threads*
/api/posts/*
Users Target Group
Threads Target Group
Posts Target Group
(c) Target Groups
37960
47115
38971
80
80
80
Users
Threads
Posts
(d) Microservices

领域驱动设计与微服务

领域驱动设计（Domain-Driven Design，等称DDD）是一种由域模型（来驱动着系统设计的思想，不是通过数据库等数据源来驱动系统设计（Model-Driven Design，简称MDD）。领域模型是对业务模型的抽象，DDD是把业务模型翻译成系统架构设计的一种方式。



拆分后的麻烦

1. 服务变多难以部署、难以管理；
2. 资源规划复杂，特别是微服务期望分表分库后有专用数据库；
3. 分布式事务；

中期-改革进入了深水期

手段：

1. 统一配置中心，分离代码和配置；
2. 统一开发测试流程；
3. 统一持续集成持续部署方式；
4. 容器化、集群化改造；
5. 更为全面的监控告警能力；

缺点：

1. 需要基础架构的视野，而不仅仅是软件架构；
2. 需要更多的跨部门沟通；
3. 改造带来的风险；
4. 新知识学习成本；

微服务和云原生

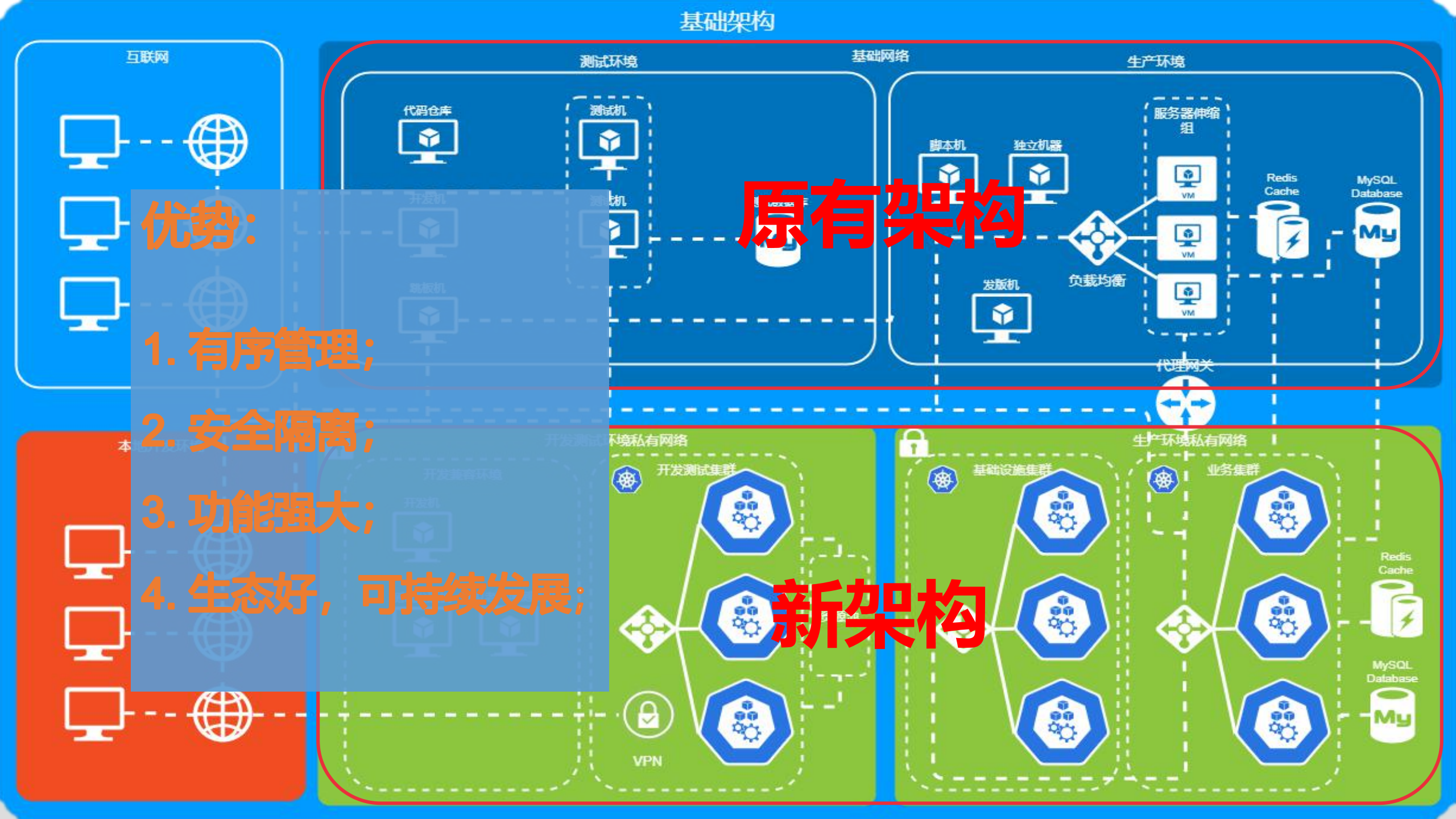
1. 引入K8S以解决服务管理、资源管理问题，并进入云原生生态；

2. 引入DevOps解决自动化流程问题，包括自动测试、代码质量评估、构建、部署等；

3. 引入Istio解决网关和服务治理问题；

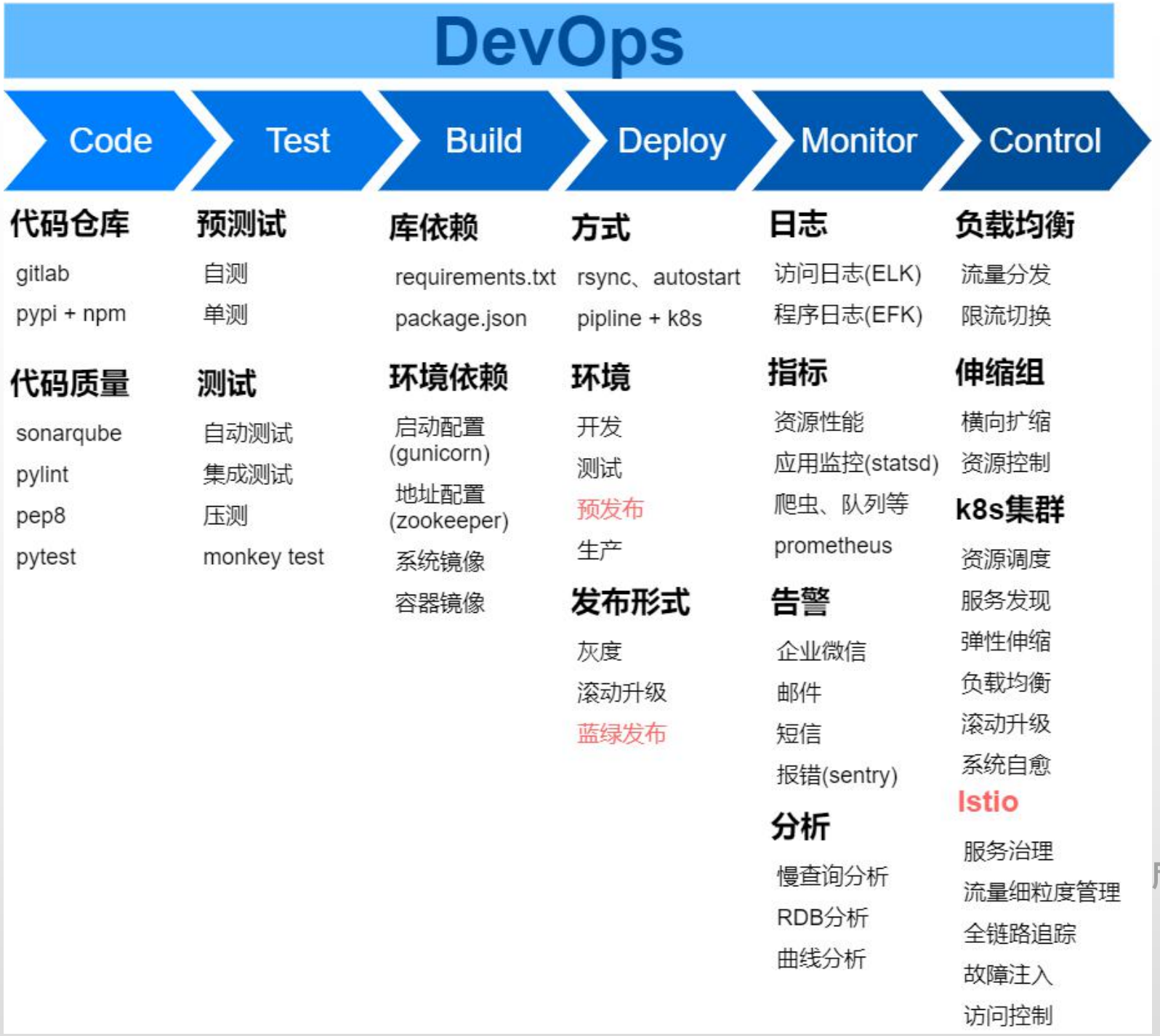
云原生应用与传统应用

云原生应用	传统应用
可预测。 云原生应用符合旨在通过可预测行为最大限度提高弹性的框架或“合同”。	不可预测。 通常构建时间更长，大批量发布，只能逐渐扩展，并且会发生更多的单点故障。
操作系统抽象化。	依赖操作系统。
资源调度有弹性。	资源冗余较多，缺乏扩展能力。
团队借助DevOps更容易达成协作。	部门墙导致团队彼此孤立。
敏捷开发。	瀑布式开发。
微服务各自独立，高内聚，低耦合。	单体服务耦合严重。
自动化运维能力。	手动运维。
快速恢复。	恢复缓慢。


基础架构
互联网
测试环境
基础网络
生产环境
代码仓库
测试机
开发机
跳板机
脚本机
独立机器
服务器伸缩组
VM
VM
VM
Redis Cache
MySQL Database
负载均衡
发版机
代理网关
原有架构
优势：
1. 有序管理；
2. 安全隔离；
3. 功能强大；
4. 生态好，可持续发展；
开发测试集群
基础设施集群
业务集群
VPN
新架构

荔枝微课

上线之路



分布式事务

分布式事务，是相对本地事务而言，而数据库本地事务有ACID四大特性；
A：原子性 C：一致性 I：隔离性 D：持久性

分布式又会面临CAP定理，即布鲁尔定理：
C：一致性 A：可用性 P：分区容错性

BASE 是 Basically Available(基本可用)、Soft state(软状态)和 Eventually consistent (最终一致性)三个短语的缩写，是对 CAP 中 AP 的一个扩展。
CAP 中理论没有网络延迟，在 BASE 中用软状态和最终一致，保证了延迟后的一致性

分布式事务处理手段

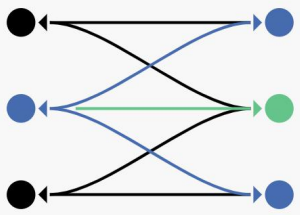
基础原理	实现	优势	必要前提
2PC	分布式数据库	简单	关系数据库
2PC	TCC	不依赖关系数据库	实现TCC接口
2PC	事务消息	高性能	MQ，实现事务检查接口
最终一致性	本地消息表 定时轮询或Binlog触发	去中心化	侵入业务，接口需要幂等性，建议MQ通讯

后期-网格化

服务网格化，是为了提高服务治理能力得到加强；

Istio

连接、保护、控制和观测服务。

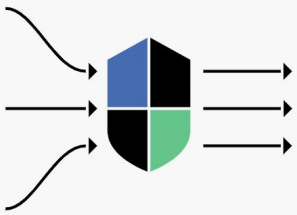


连接

智能控制服务之间的流量和 API 调用，进行一系列测试，并通过红/黑部署逐步升级。

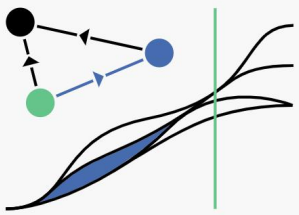
保护

通过托管身份验证、授权和服务之间通信加密自动保护您的服务。



控制

应用策略并确保其执行，使得资源在消费者之间公平分配。

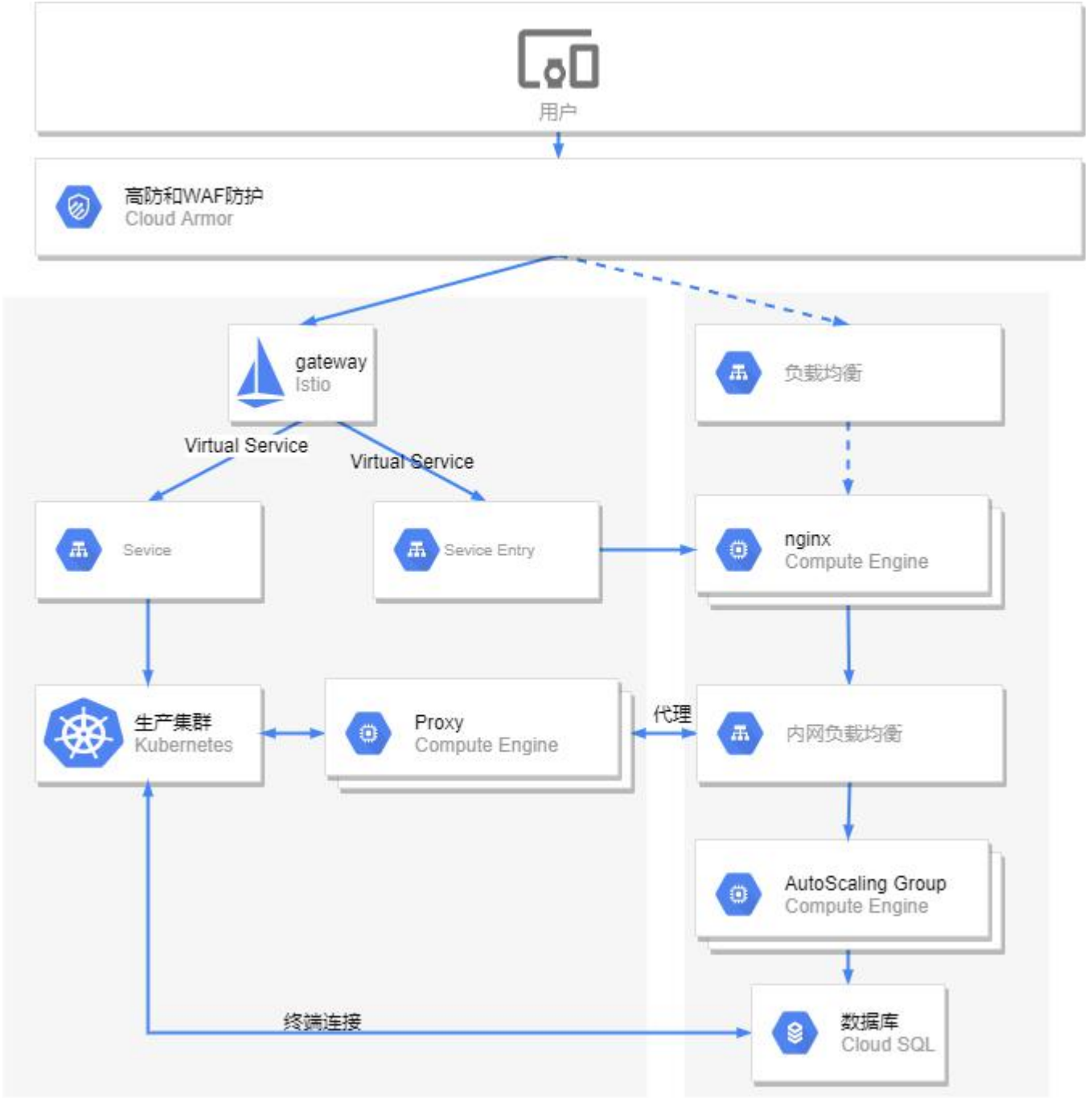


观测

对您的一切服务进行多样化、自动化的追踪、监控以及记录日志，以便实时了解正在发生的事情。

案例-流量转移

一边行驶一边换轮胎，完成新旧项目更替。



未来-持续演进，永无止境

Servceless

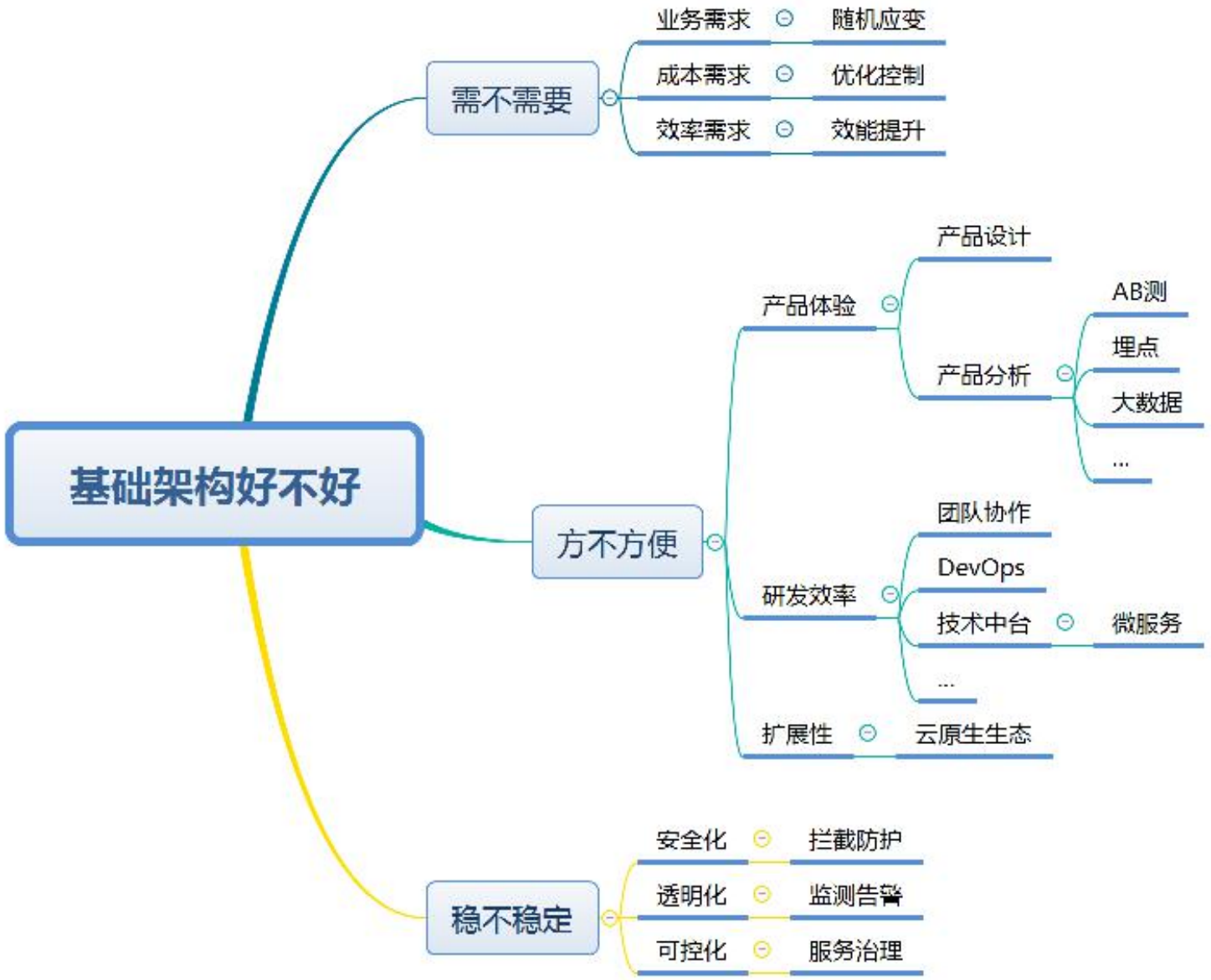
业务中台、基础中台、数据中台

AIOps/NoOps

技术方案、理念千千万，只有适合自己的才是最好的。

架构方向

架构的方向始终是围绕这几点来的。



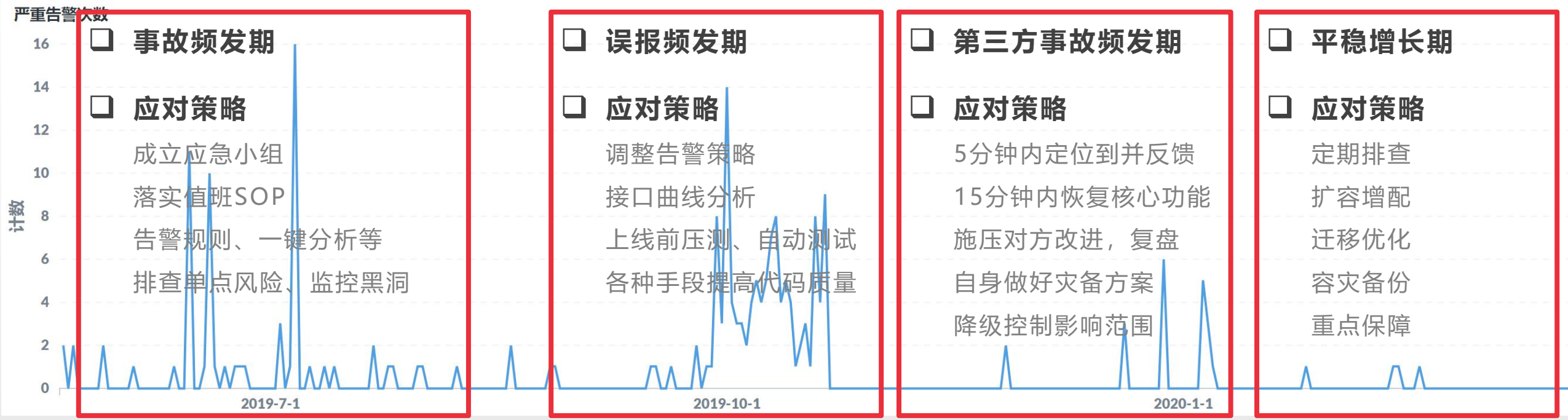
原则

1. 微服务的第一条规则是，不要构建微服务，即不要为了微服务而微服务；
2. 不要在没有DevOps或云服务的情况下进行微服务；
3. 不要通过使它们变得太小来制造太多的微服务；
4. 不要将微服务转变为SOA；
5. 不要尝试成为Netflix：不需要什么都从头开始；

评价方法

1. 性能测试（如网络耗时）；
2. 压力测试；
3. 定期演练；
4. 团队、用户满意程度；

稳定性整体趋势



- ❑ **事故频发期**
- ❑ **应对策略**
  - 成立应急小组
  - 落实值班SOP
  - 告警规则、一键分析等
  - 排查单点风险、监控黑洞

- ❑ **误报频发期**
- ❑ **应对策略**
  - 调整告警策略
  - 接口曲线分析
  - 上线前压测、自动测试
  - 各种手段提高代码质量

- ❑ **第三方事故频发期**
- ❑ **应对策略**
  - 5分钟内定位到并反馈
  - 15分钟内恢复核心功能
  - 施压对方改进，复盘
  - 自身做好灾备方案
  - 降级控制影响范围

- ❑ **平稳增长期**
- ❑ **应对策略**
  - 定期排查
  - 扩容增配
  - 迁移优化
  - 容灾备份
  - 重点保障

注：该告警数不代表事故数，只反映关键接口耗时波动

Thanks！
欢迎相互交流学习