新架构方案

Huayulei_2003@hotmail.com
2018/05/15

目录

概述：目标和业务需求

整体目标：

1. 规范性：建立服务快速开发、部署、运维管理、持续开发、持续集成的流程规范
2. 自动化：开发代码并提交到代码库，简单配置，服务就会自动集成、自动部署
3. 高可用：服务采用集群部署、多节点同时服务
4. 扩展性：服务易于水平扩容和缩容
5. 高性能：既能满足现有阶段性能需求又能支撑业务量快速发展的下一阶段

兼容现有和未来业务需求

- 支持开发语言的多样性，如：php、golang、python、c/c++等
- 支持数据库的多样性，如：MySQL，Mongodb，Postgresql，Redis，Memcached
- 支持接口协议的多样性，如：http、https、http2、gRPC、websocket
- 支持系统容量和日活跃用户在未来阶段的增长（1000W日活）

概述：高效技术团队

高效技术团队的技术特征：

- 微服务架构

微服务和SOA架构是目前主流，并且微服务更加流行、高加高效

- 容器和容器编排技术

大量使用容器技术和容器编排技术，其中以docker + kubernetes

- DevOps理念深度落地

有效的版本管理、高度敏捷CI/CD体系，高速软件迭代交付

- 自动化程度比较高

大量应用工具和系统自动化管理，尽量减少人力成本并保证高效率和质量

微服务：微服务概述

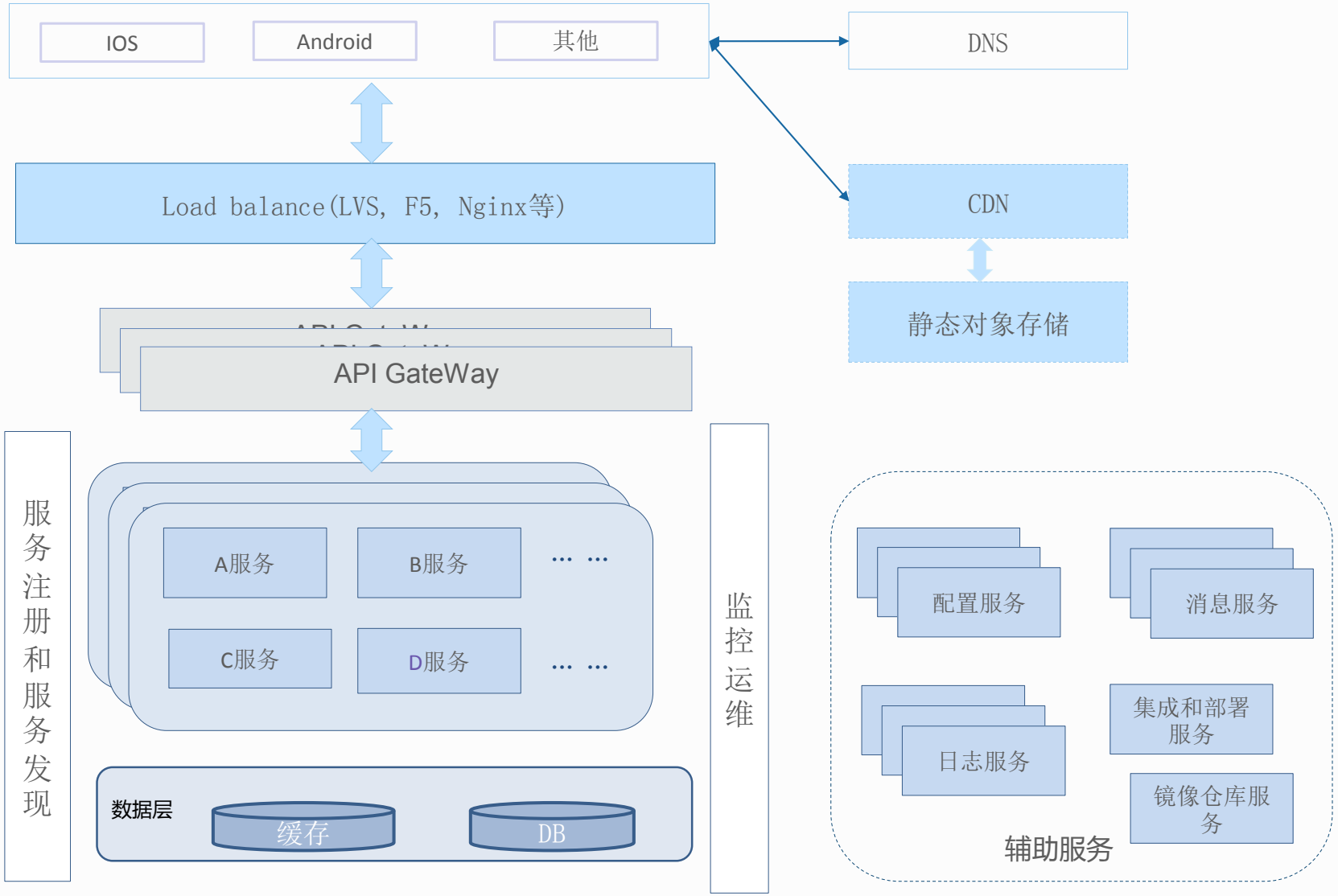
微服务的优点：

① 服务本身逻辑功能比较简单
② 对于一个服务，可以选择最好和最适合的语言和框架来开发
③ 服务之间本质上是松耦合的
④ 多个团队可以同时并行工作在不同的服务中
⑤ 可对某一服务持续发布，而对其他的影响相对较小
⑥ 可以水平扩容和缩容，灵活方便
⑦ 使用集群多点对外提供服务，提高了可用性。

微服务架构的难点：

① 微服务可能增加调用开销：比如网络
② 微服务对监控运维要求比较高
③ 服务间以接口访问、接口设计要求比较高
④ 分库、分表等数据拆分增加了系统的复杂度
⑤ 多重缓存、多数据库带来的数据一致性的挑战
⑥ 服务测试性的成本和复杂度上升

微服务：整体架构概述


IOS
Android
其他
DNS
Load balance(LVS, F5, Nginx等)
CDN
静态对象存储
API GateWay
服务注册和服务发现
A服务
B服务
C服务
D服务
数据层
缓存
DB
监控运维
配置服务
消息服务
日志服务
集成和部署服务
镜像仓库服务
辅助服务

微服务：API Gateway概述

GateWay优点：

① 服务发现与动态负载均衡：自动发现后端拆分、聚合、扩容、缩容的服务集群，当后端服务有所变化的时候，能够实现健康检查和动态的负载均衡。

② 流量控制：为每种类型的请求分配容量，当请求数量超过阀值时抛掉外部请求，限制流量，保护后台服务不被大流量冲垮；当新版本上线时，可以控制流量在新老版本之间的分配。

③ 灰度发布与AB测试：通过配置访问路由，以及访问权重，可实现灰度发布，或者AB测试。同时上线两套系统，通过切入部分流量的方式来测试和验证新系统。

④ 统一的接口协议和规范：使用统一的接口协议格式和接入规范、接入SDK、避免管理混乱造成的不可维护性。

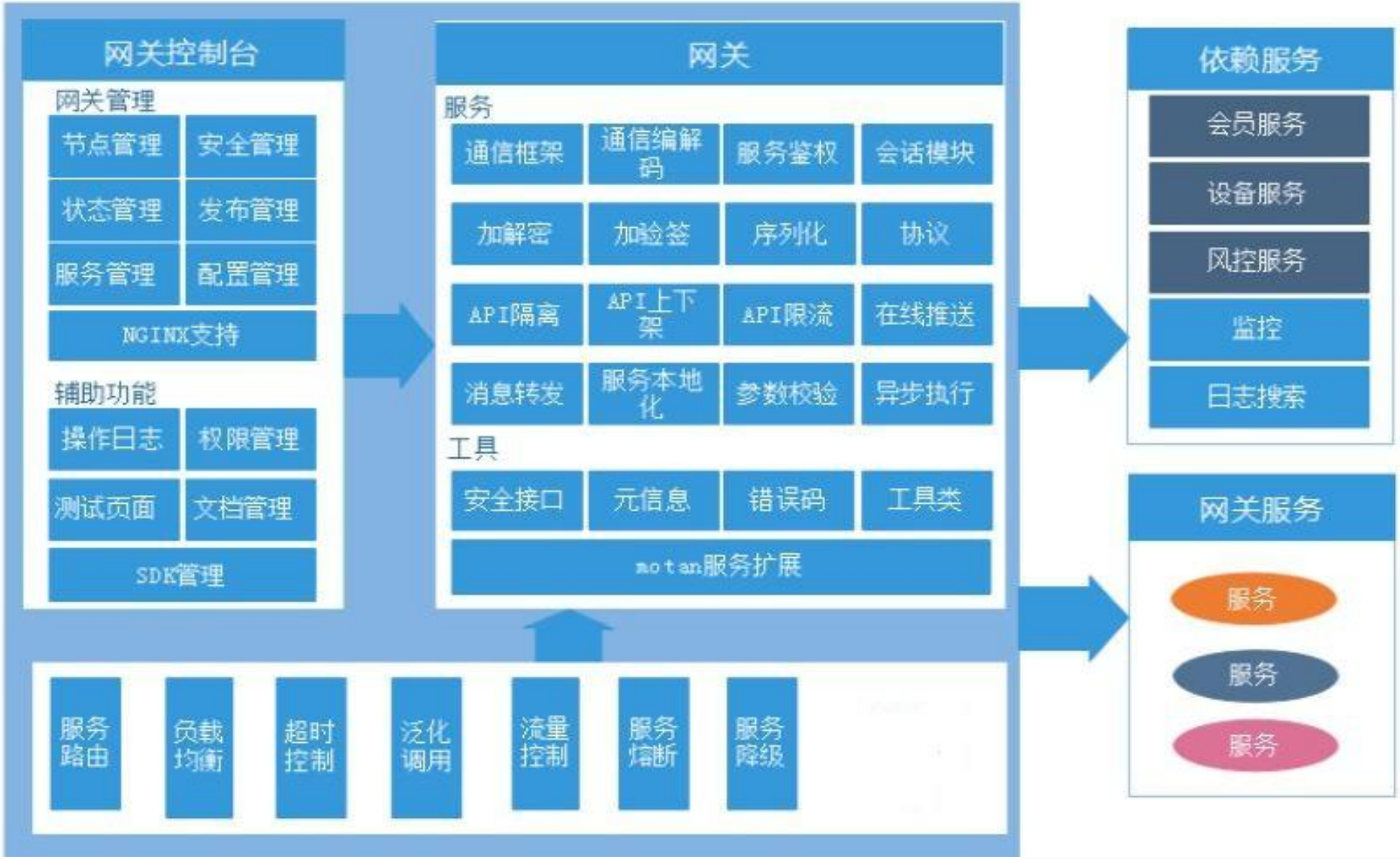
⑤ 身份认证和安全性控制：对每个外部请求进行用户认证，拒绝没有通过认证的请求，还能通过访问模式分析，实现反爬虫功能。

⑥ 数据监控和日志分析：可收集数据和统计，为系统服务和整体架构优化提供数据支持。

GateWay缺点：

① 所有服务都流经Gateway，调用微服务性能有一定损耗。

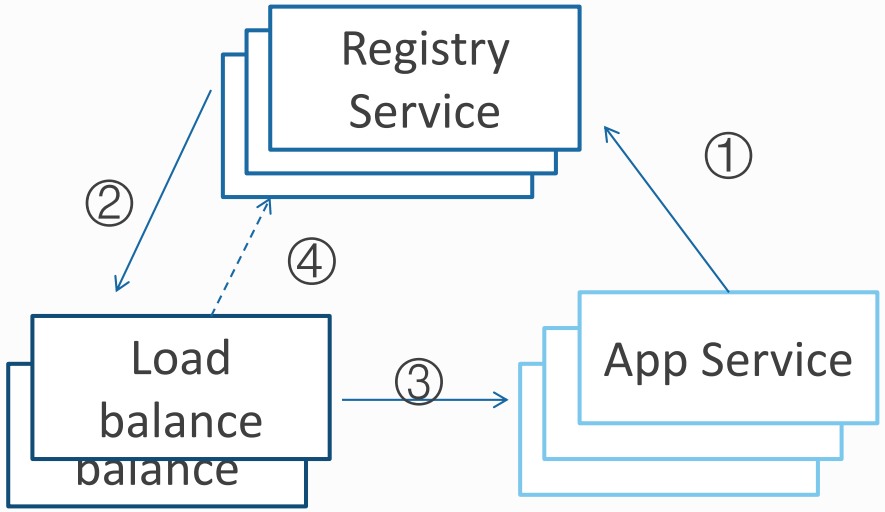
微服务：API Gateway逻辑功能



微服务：API Gateway开源项目

名称	开发语言	支持方	star	网址
zuul和zuul2	java	Netflix	5424	https://github.com/Netflix/zuul/wiki
kong	c+lua	kong	16627	https://konghq.com/
tyk	go	TykTechnologies	3465	http://tyk.io
fabio	go	eBay	4824	https://fabiolb.net
traefik	go		15925	https://traefik.io/

微服务：服务注册发现与负载均衡



●app Service负载提供业务服务
●Registry service负责服务注册和服务健康检查
●Load balance负责服务发现和负载均衡。在存在API GateWay的场景下，该服务也要注册以提供给GateWay服务发现

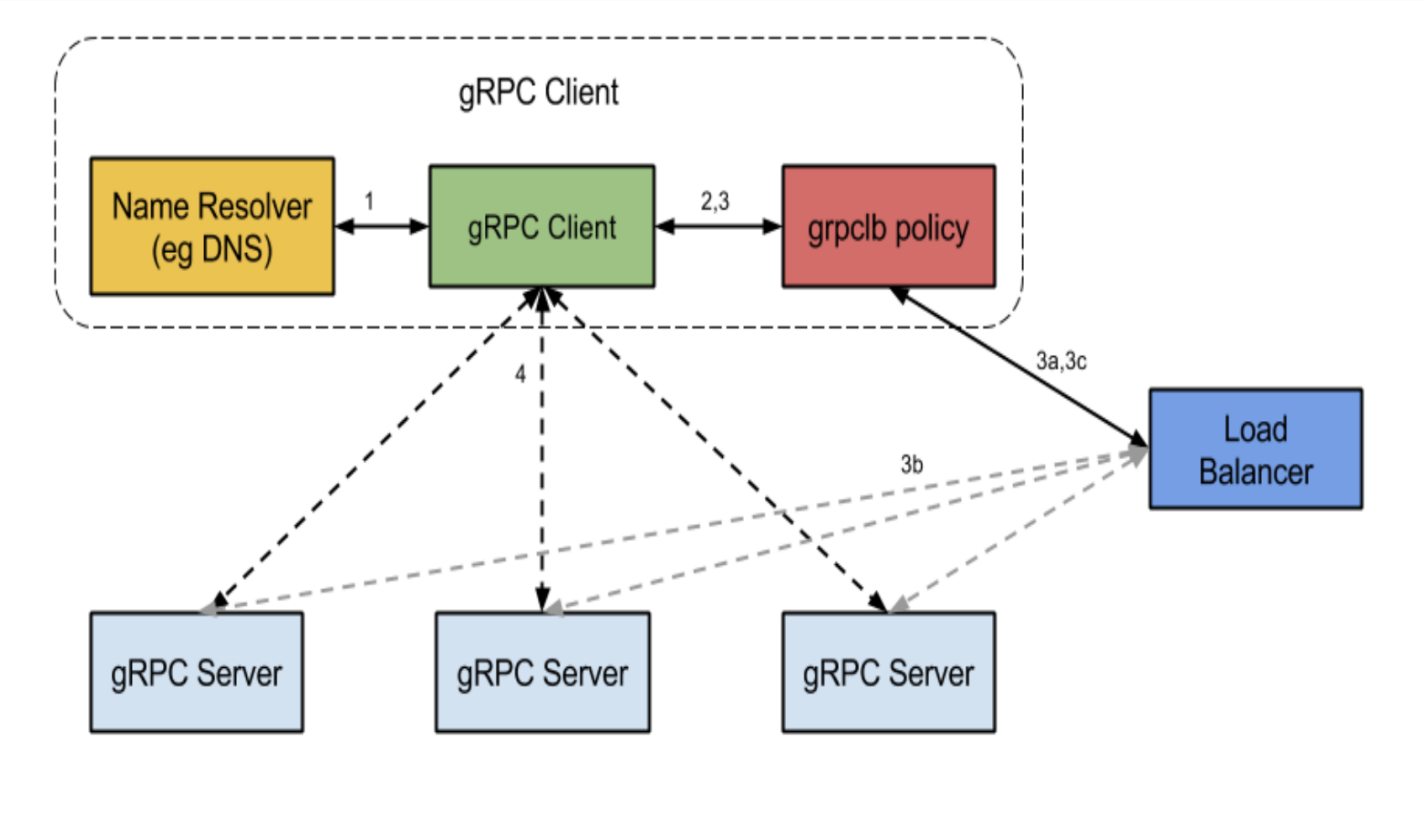
主要优点：

- 可扩展性：任务服务都可以水平扩容和缩容。
- 高可用：出现单点故障，不影响业务。
- 单一服务职责和功能分离：便于升级维护

主要缺点：

- 整体系统调用关系复杂，系统监控和故障恢复管理难度增大
- 服务交互增多，性能有所下降
- 此方案适合业务量级比较大、基础架构成熟的公司。

微服务：gRPC服务注册发现与负载均衡



Load-balancing policies fit into the gRPC client workflow in between name resolution and the connection to the server.

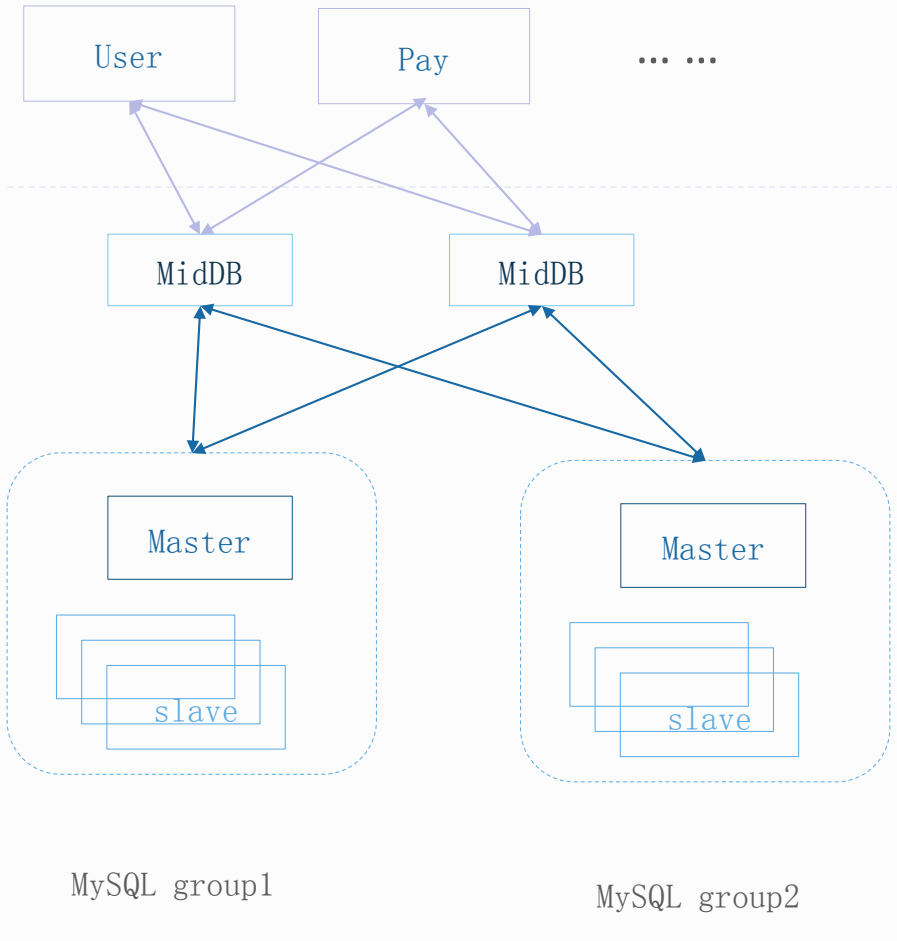
Our solution is: gRPC and Etcd 或 gRPC and consul

https://github.com/grpc/grpc/blob/master/doc/load-balancing.md

微服务：服务注册发现开源项目

名称	开发语言	支持方	star	网址
zookeeper	java	apache	4618	http://zookeeper.apache.org/
consul	go	HashiCorp	12475	https://www.consul.io/
etcd	go	coreos	18845	https://coreos.com/etcd

微服务：数据库中间层



数据库中间层MidDB主要功能：

1. 支持分库分表

① Hash模式

② 支持range模式

2. 支持读写分离

① Master写或强制读

② Slave负载均衡读，带权重

3. 其他功能：

① Master或Salve的上线、下线

② 支持SQL日志和慢日志查询

③ 支持Client连接管理

④ Metric日志监控和错误日志监控

微服务：数据库中间层开源项目

名称	开发语言	支持方	star	网址
Mycat-Server	java		4410	http://mycat.org.cn
cobar	java	alibaba	2251	https://github.com/alibaba/cobar
Atlas	c	Qihoo360	3536	https://github.com/Qihoo360/Atlas
kingshard	go		3564	https://github.com/flike/kingshard
TiDB	go等	pingcap	13749	https://pingcap.com

微服务：监控概述

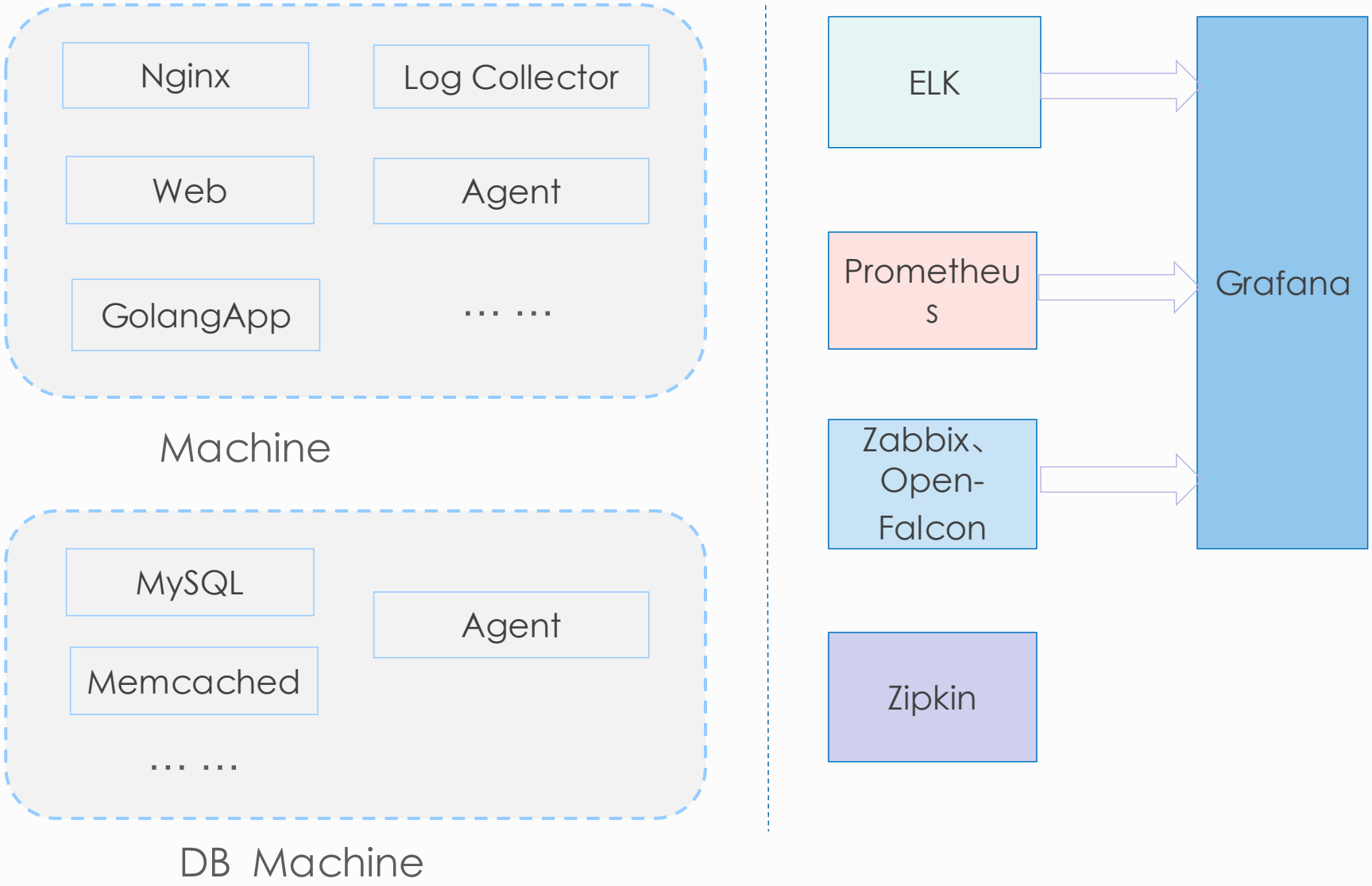
选型原则：

- 稳定性：监控服务不能造成线上服务不稳定
- 性能：监控服务应尽量减少对硬件资源占用，不会影响线上服务性能
- 效率：尽量使用成熟，很多企业实践过的方案，文档和社区支持比较完善

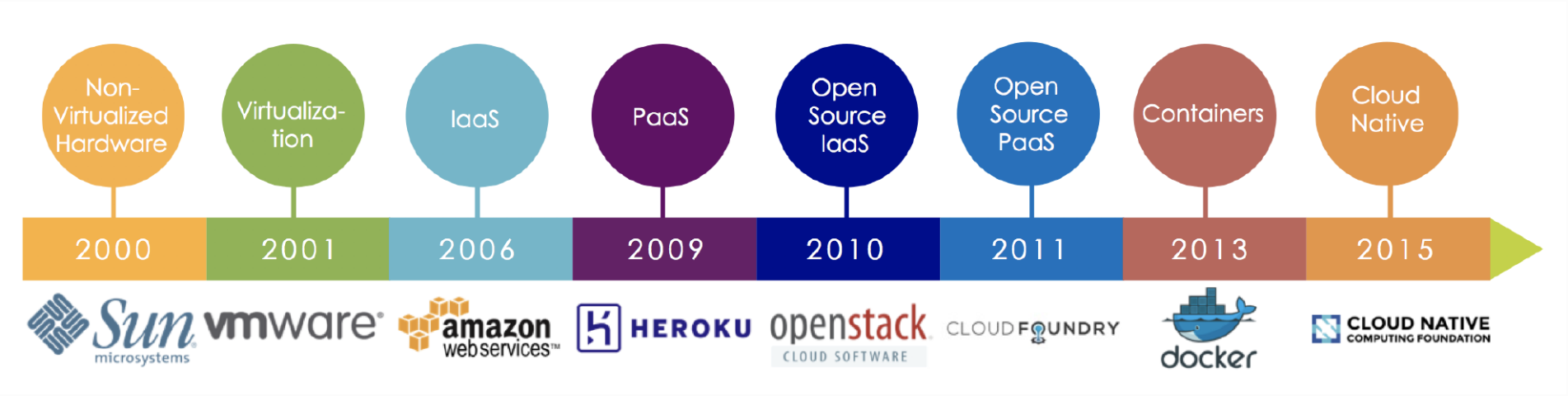
主要描述：

- 基础监控：使用Zabbix服务监控、Open-Falcon服务监控
- Metric监控：使用prometheus监控
- 服务状态监控：使用收集脚本上传Zabbix Agent监控、Open-Falcon Agent服务监控
- 调用链监控（可选）：推荐使用zipkin
- 故障定位监控（可选）：推荐使用sentry

微服务：监控架构


Nginx
Log Collector
Web
Agent
GolangApp
… …
Machine
MySQL
Agent
Memcached
… …
DB Machine
ELK
Prometheus
Zabbix、Open-Falcon
Zipkin
Grafana

容器化技术：Docker概述

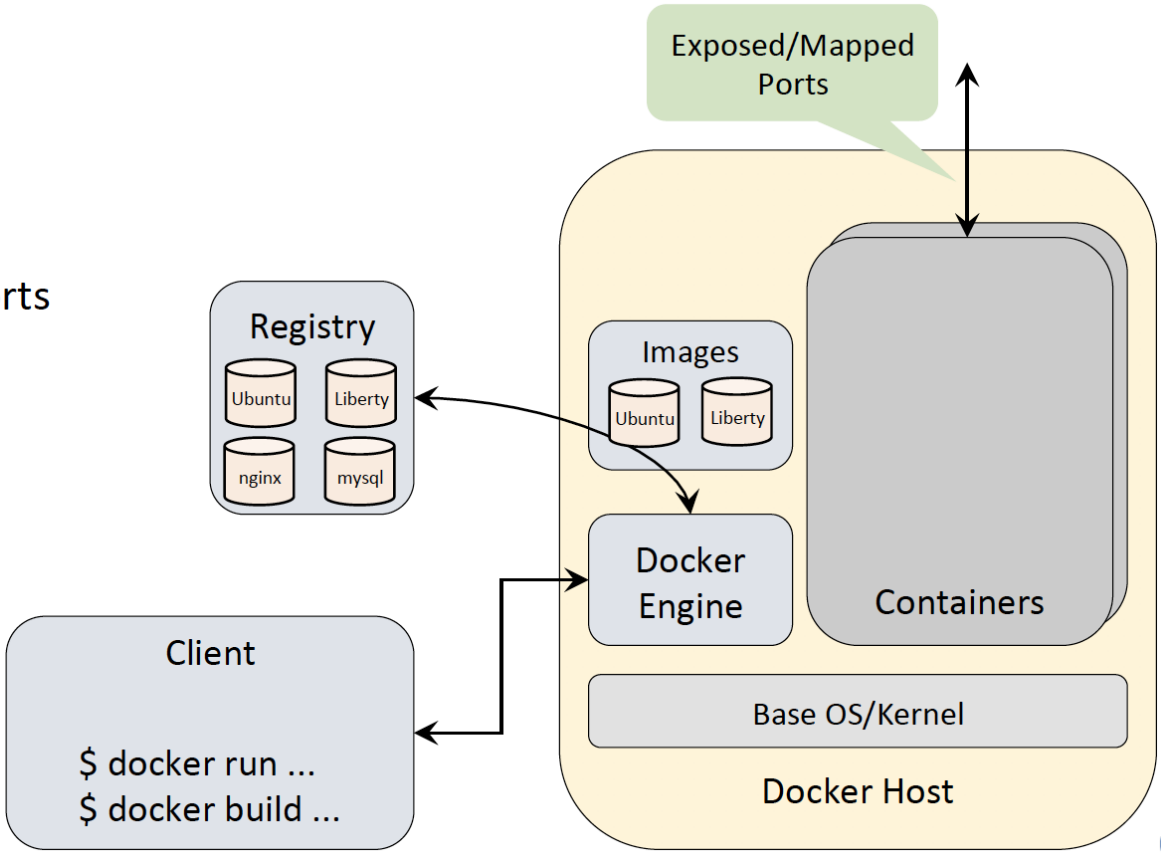


What is Docker:

- Tooling to manage containers: made them easy to use
- Docker creates and manages the lifecycle of containers
  - > Setup filesystem
  - > CRUD container:
    - >> setup networks
    - >> setup volumes / mounts
    - >> Create: start new process telling OS to run it in isolation

容器化技术：Docker 组成概述

- Docker Engine
  - Manages containers on a host
  - Accepts requests from clients
    - REST API
  - Maps container ports to host ports
    - E.g. 80 → 3582
- Images
- Docker Client
  - Drives daemon
  - Drives "builder" of Images
- Docker Registry
  - Image DB



容器化技术：Kubernetes概述

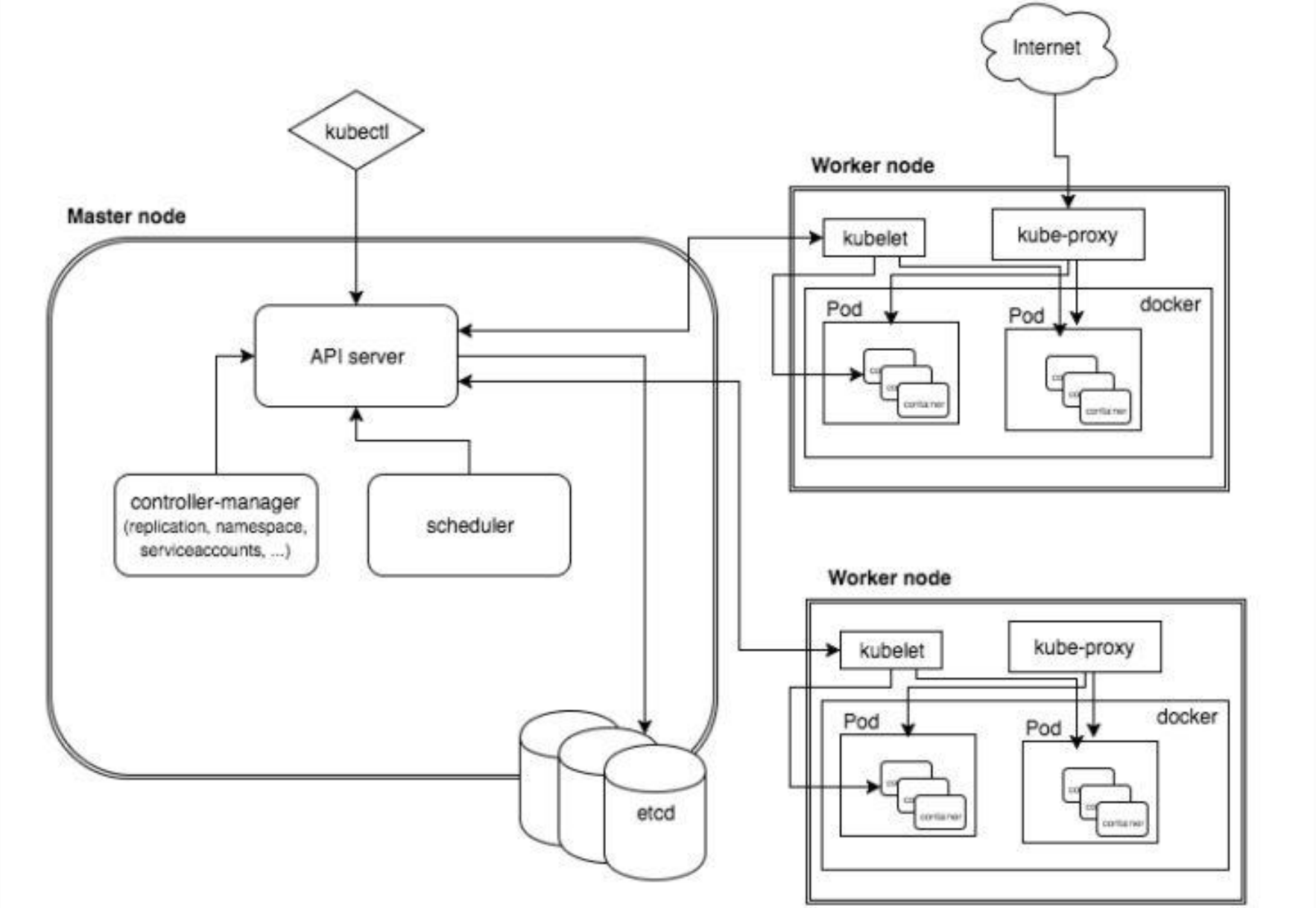
What is Kubernetes：

- Kubernetes is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure.
- Kubernetes is enterprise level container orchestration.

With Kubernetes, you are able to quickly and efficiently respond to customer demand:

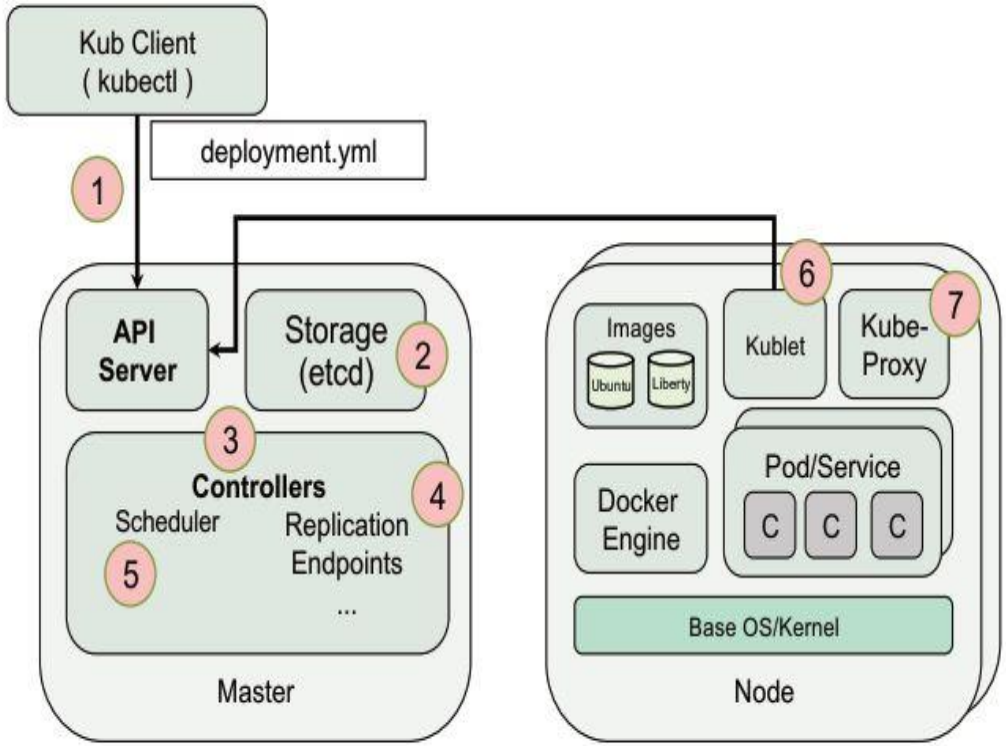
- Deploy your applications quickly and predictably.
- Scale your applications on the fly.
- Seamlessly roll out new features.
- Optimize use of your hardware by using only the resources you need
- Manage infrastructure resources needed by applications : volumes, networks, secrets, and more

容器化技术：Kubernetes架构



容器化技术：Kubernetes部署应用

1. User via "kubectl" deploys a new application
2. API server receives the request and stores it in the DB (etcd)
3. Watchers/controllers detect the resource changes and act upon it
4. ReplicaSet watcher/controller detects the new app and creates new pods to match the desired # of instances
5. Scheduler assigns new pods to a kubelet
6. Kubelet detects pods and deploys them via the container runing (e.g. Docker)
7. Kubeproxy manages network traffic for the pods – including service discovery and load-balancing



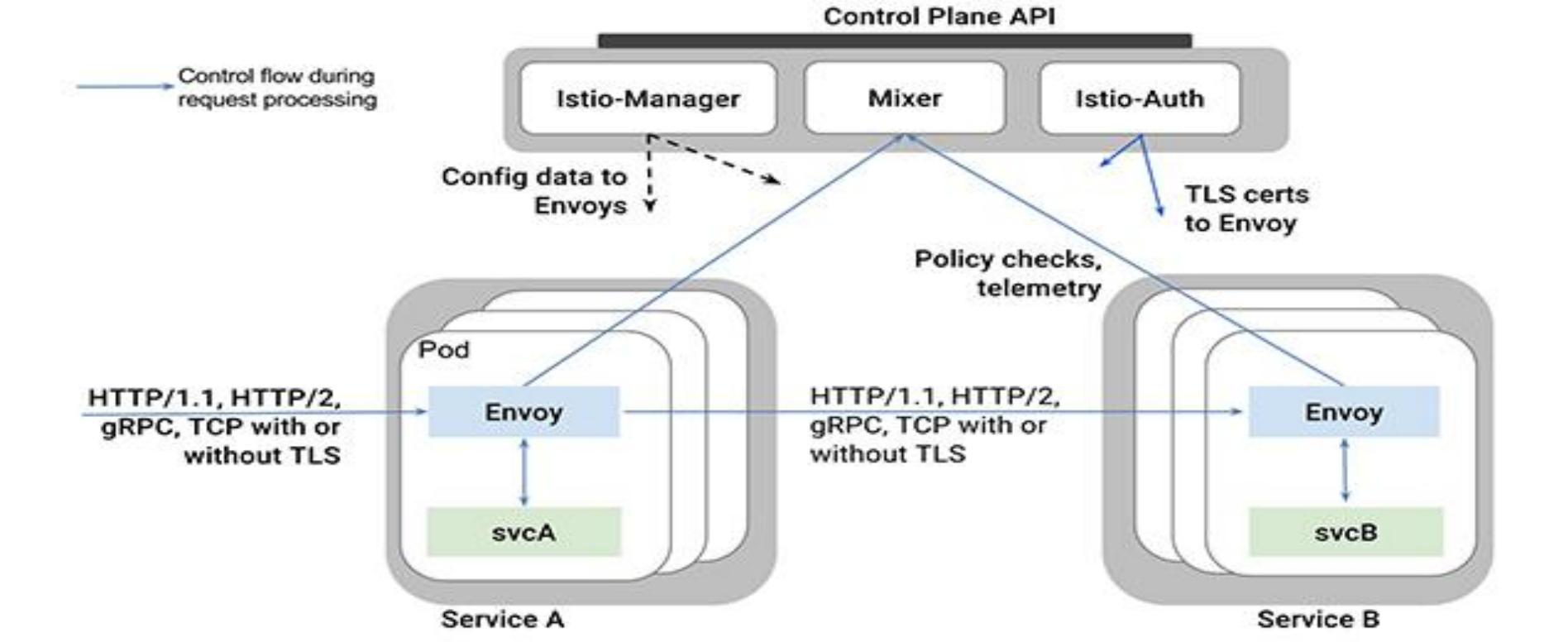
容器化技术：微服务治理istio概述

istio是由IBM，Google，Lyft联合研发的**微服务治理**新项目。Istio提供一种简单的方式来建立已部署服务网络，具备负载均衡、服务间认证、监控等功能，而不需要改动任何服务代码。想要为服务增加对Istio的支持，您只需要在环境中部署一个特殊的边车（sidecar），使用Istio控制面板功能配置和管理代理，拦截微服务之间的所有网络通信。

Istio的功能概述：

- 流量管理。控制服务之间的流量和API调用的流向，使得调用更可靠，并使网络在恶劣情况下更加健壮。
- 可观察性。了解服务之间的依赖关系，以及它们之间流量的本质和流向，从而提供快速识别问题的能力。
- 策略执行。将组织策略应用于服务之间的互动，确保访问策略得以执行，资源在消费者之间良好分配。策略的更改是通过配置网格而不是修改应用程序代码。
- 服务身份和安全。为网格中的服务提供可验证身份，并提供保护服务流量的能力，使其可以在不同可信度的网络上流转。
- 平台支持。Istio旨在可以在各种环境中运行，包括跨云、预置环境、Kubernetes、Mesos等。最初专注于Kubernetes，但很快将支持其他环境。
- 集成和定制。策略执行组件可以扩展和定制，以便与现有的ACL、日志、监控、配额、审核等解决方案集成。

容器化技术：微服务治理istio原理



Istio 通过引入可编程路由和共享管理层，将不同的微服务转换为集成服务网格。通过将 Envoy 代理服务器注入到服务之间的网络路径中，Istio 可以提供复杂的流量管理控制，比如负载均衡和细粒度路由。在该路由网格中，还能够提取关于流量行为的大量指标，可使用它们来执行策略决策，例如运营商可配置的细粒度访问控制和速率限制。这些指标也被发送到监控系统。

容器化技术：微服务治理istio组成

Envoy：

Istio使用Envoy代理的扩展版本，Envoy是以C++开发的高性能代理，用于调解服务网格中所有服务的所有入站和出站流量。Envoy的许多内置功能被istio发扬光大，例如动态服务发现，负载均衡，TLS终止，HTTP/2&gRPC代理，熔断器，健康检查，基于百分比流量拆分的分段推出，故障注入和丰富指标。Envoy被部署为sidecar,和对应服务在同一个Kubernetes pod中。

Mixer：

Mixer负责在服务网格上执行访问控制和使用策略，并从Envoy代理和其他服务收集遥测数据。代理提取请求级属性，发送到Mixer进行评估。

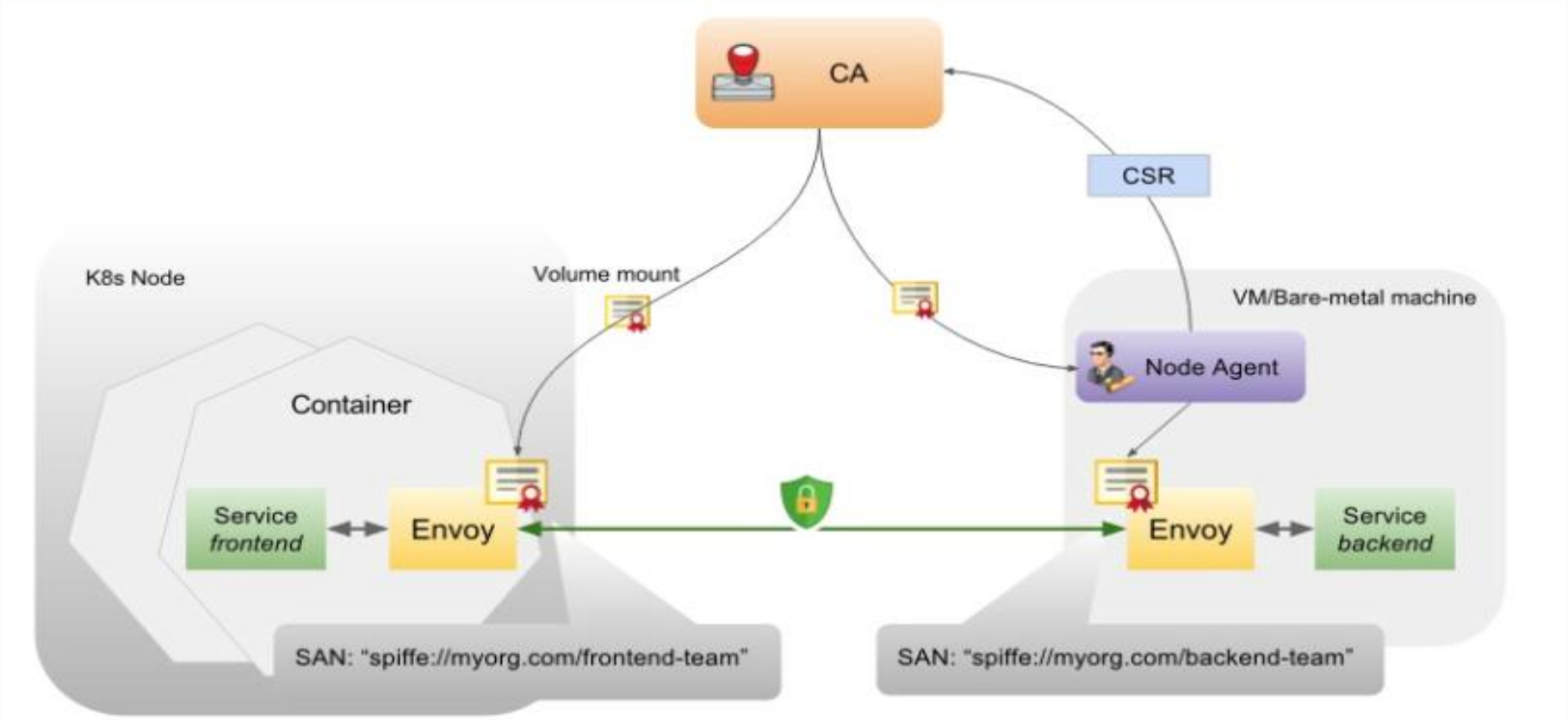
Pilot：

Pilot负责收集和验证配置并将其传播到各种Istio组件。它从Mixer和Envoy中抽取环境特定的实现细节，为他们提供用户服务的抽象表示，独立于底层平台。此外，流量管理规则（即通用4层规则和7层HTTP/gRPC路由规则）可以在运行时通过Pilot进行编程。

Istio-Auth：

Istio-Auth提供强大的服务间认证和终端用户认证，使用交互TLS，内置身份和证书管理。可以升级服务网格中的未加密流量，并为运维人员提供基于服务身份而不是网络控制来执行策略的能力。

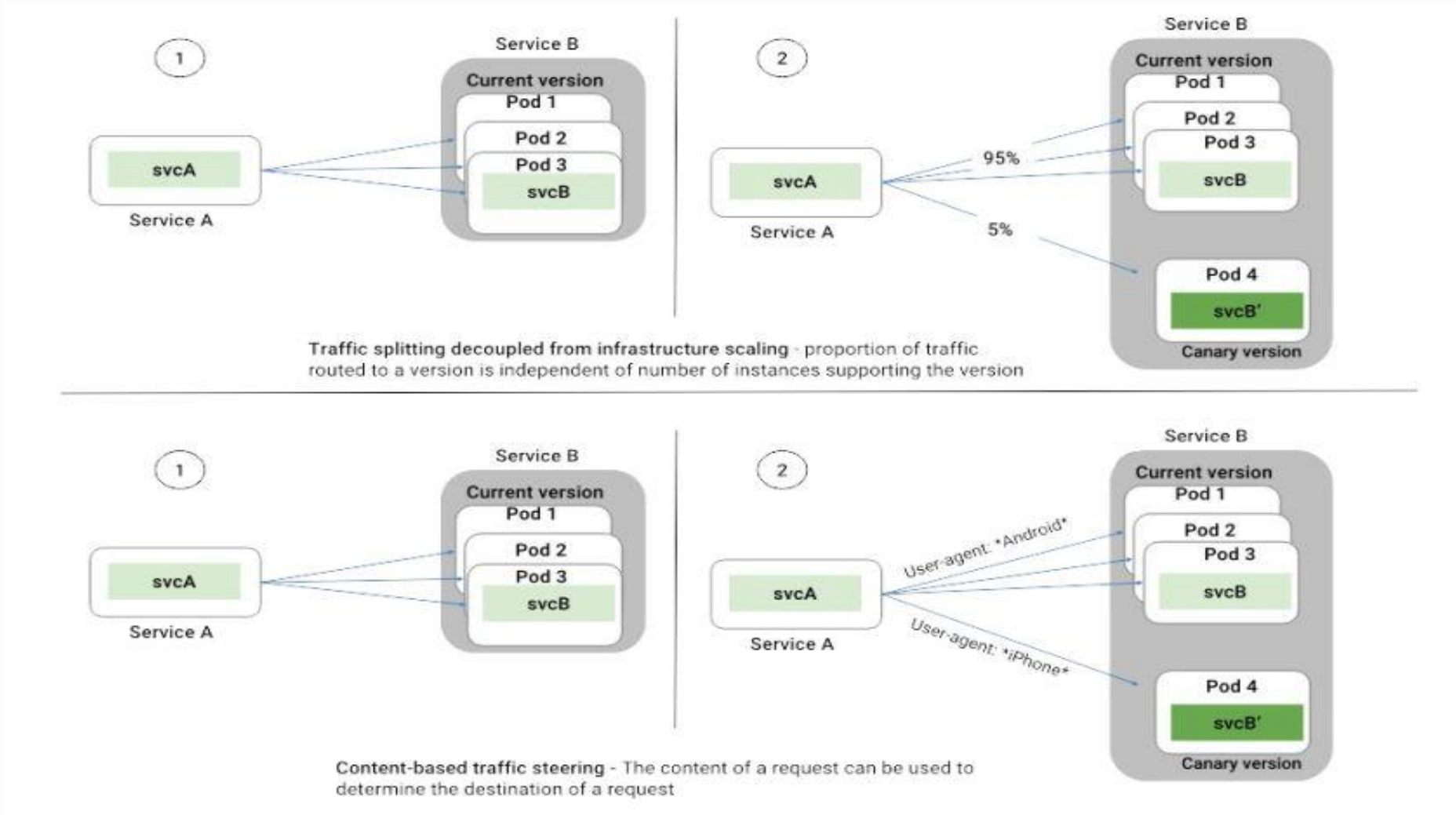
容器化技术：微服务治理istio Auth



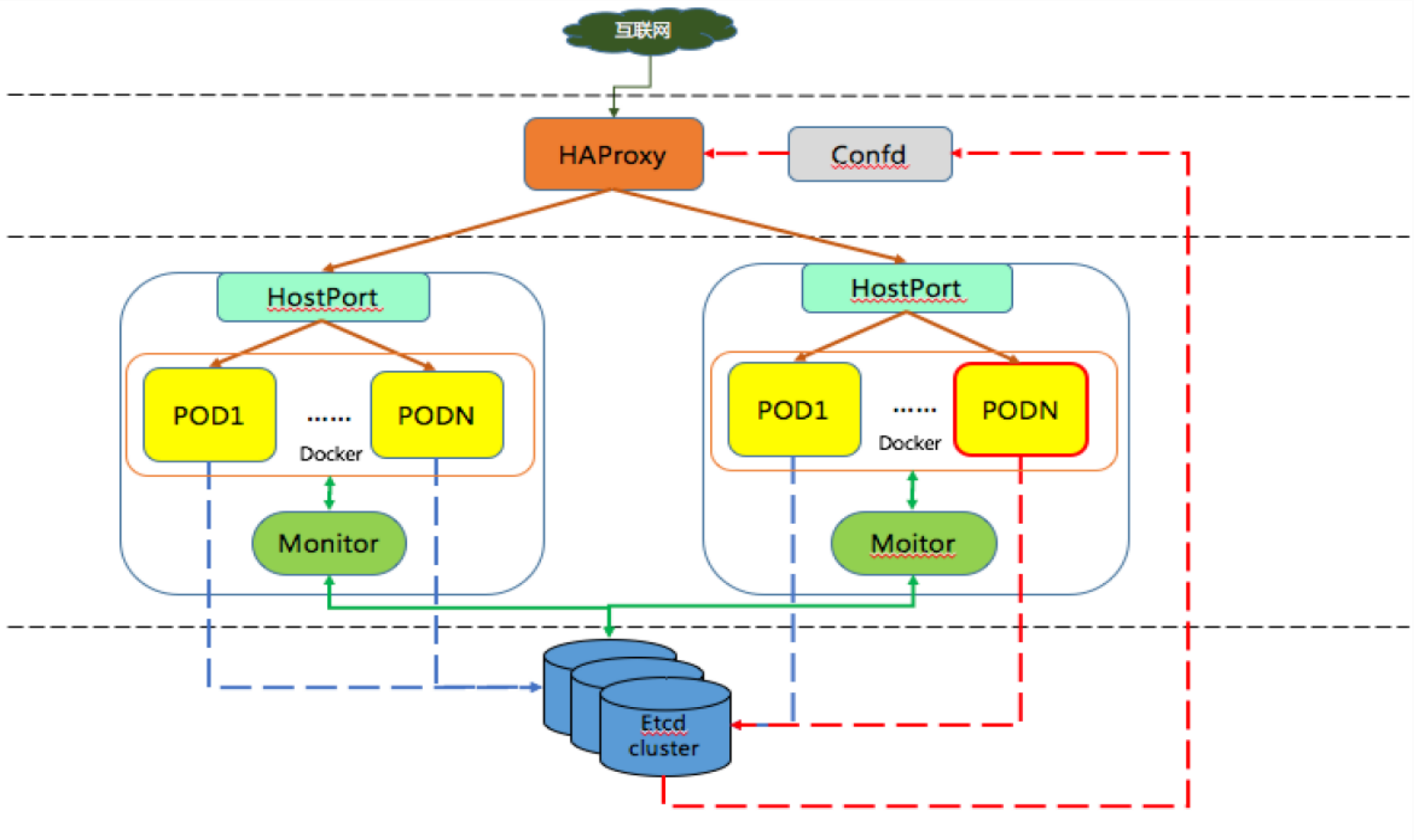
Istio Auth架构，其中包括三个主要组件：身份、密钥管理和通信安全。

Istio Auth利用secret volume mount，从Istio CA向Kubernetes容器传递keys/certs,它在本地生成私钥和CSR（证书签名请求），将CSR发送给Istio CA进行签名，并将生成的证书与私钥一起交给Envoy.

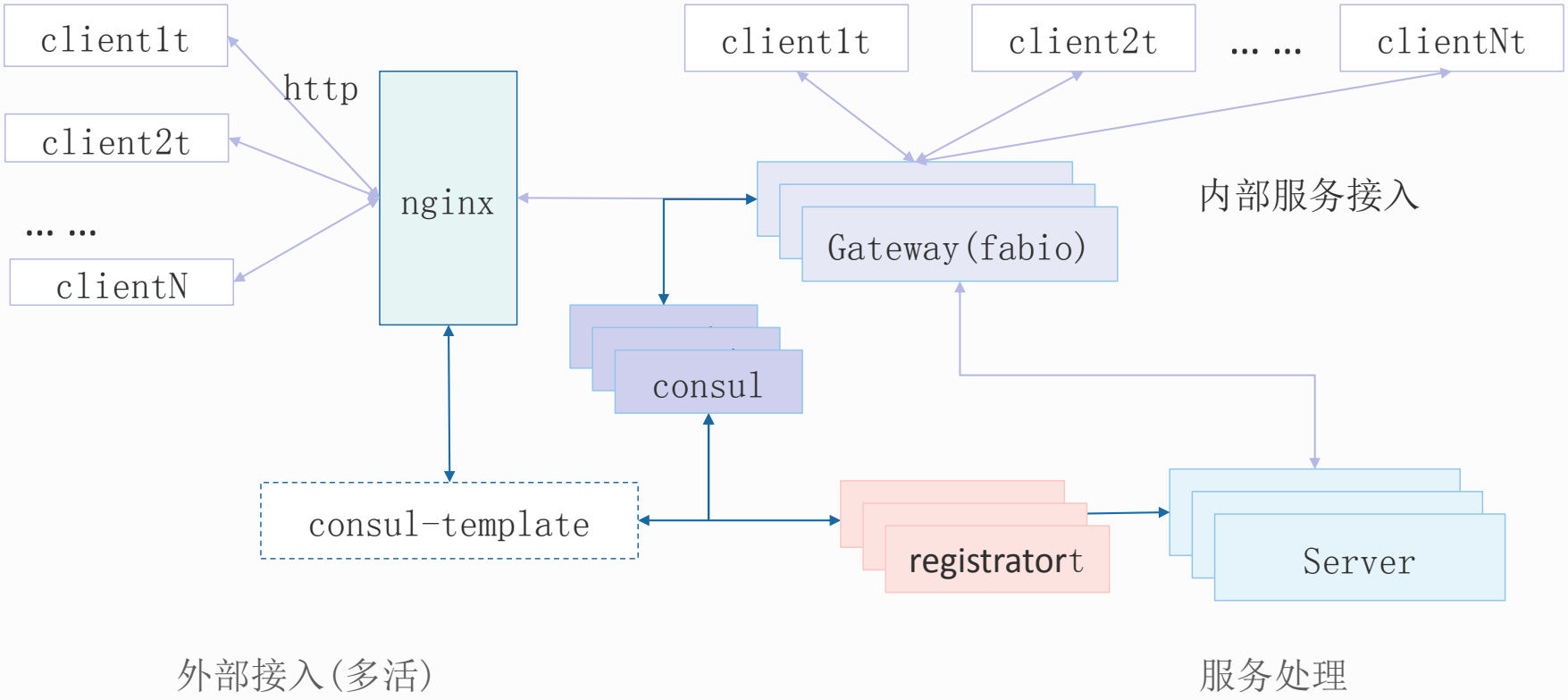
容器化技术：微服务治理istio流量管理



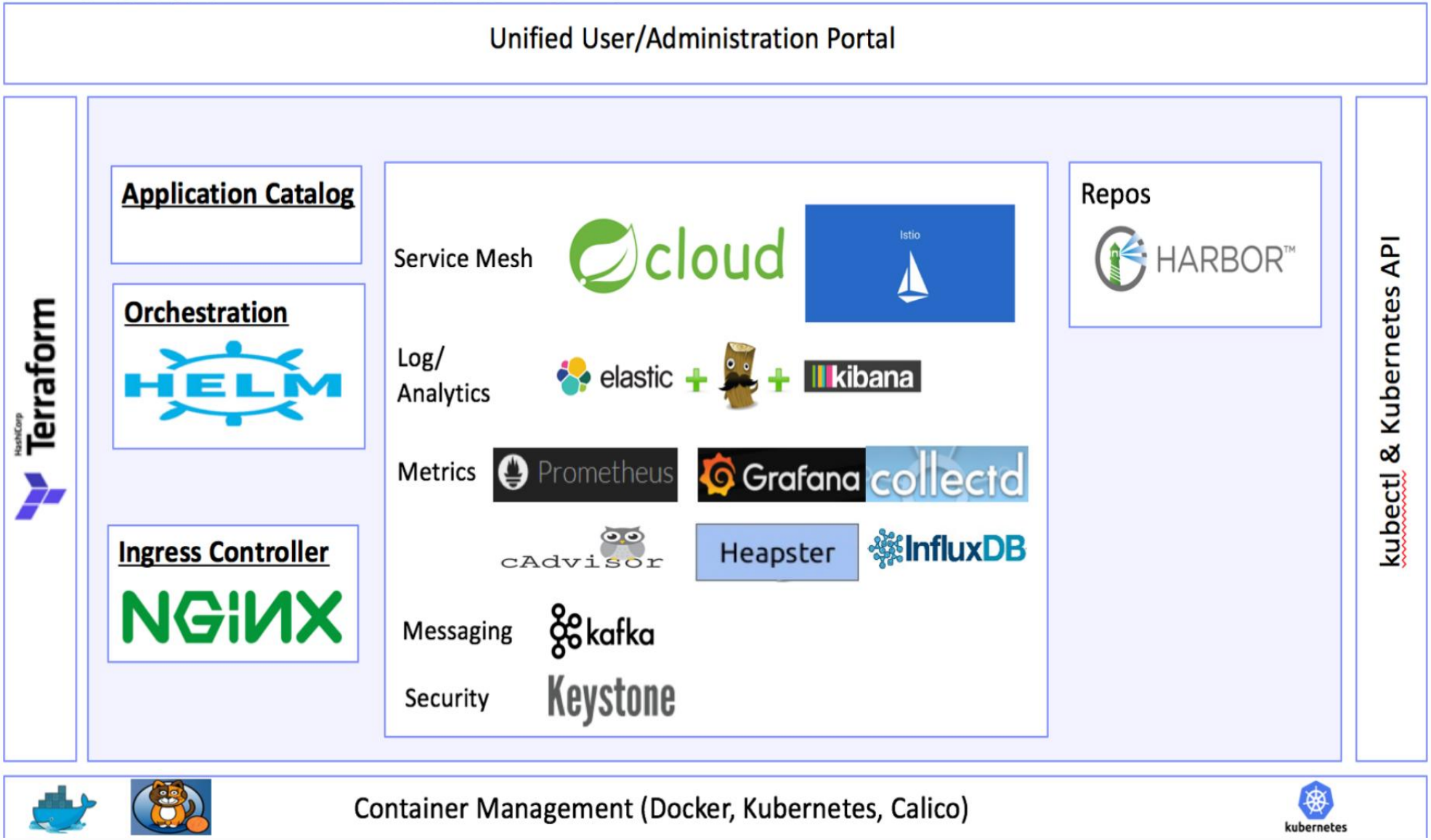
容器化技术：部署架构方案1



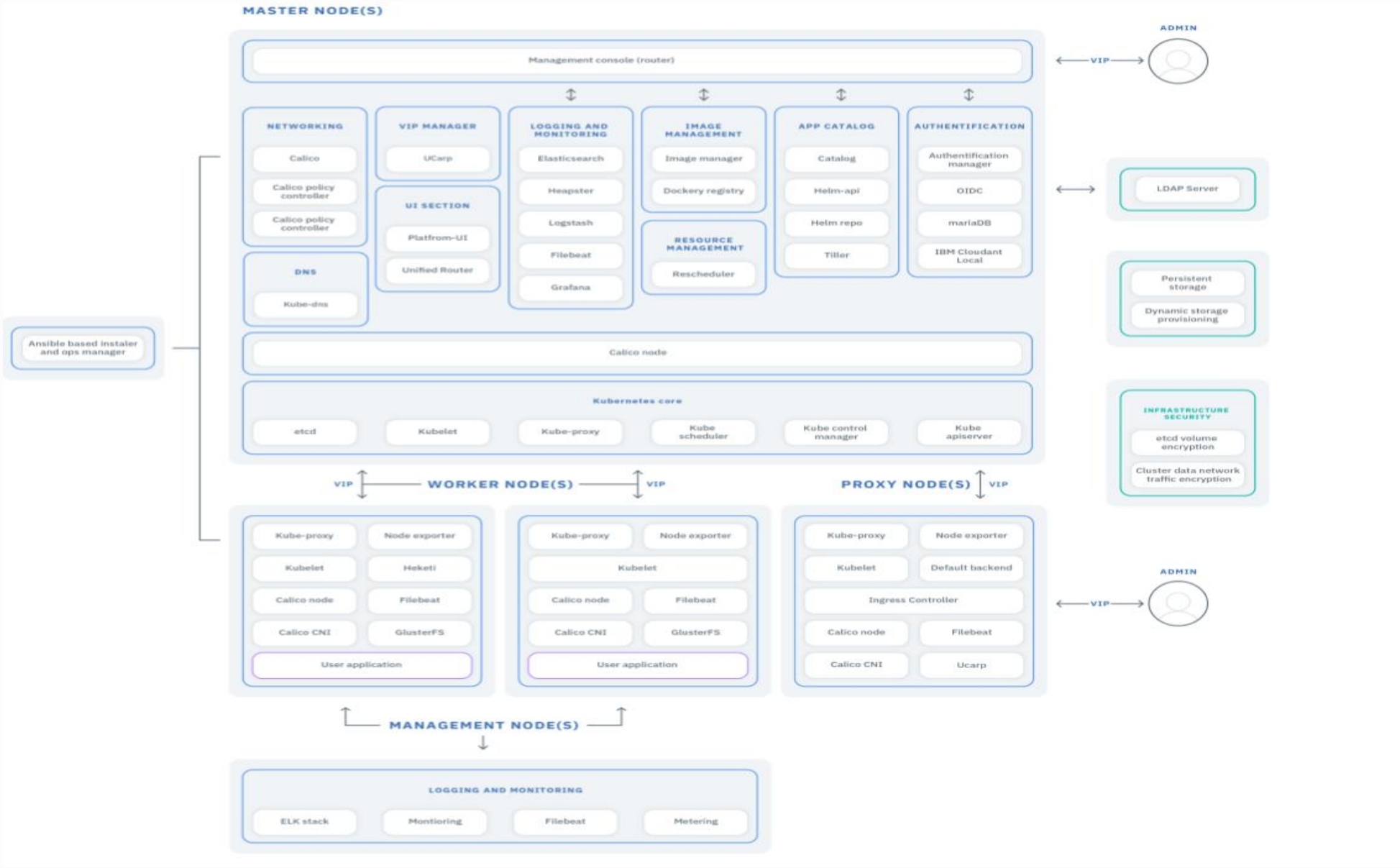
容器化技术：部署架构方案2



容器化技术：部署架构方案3(1/2)



容器化技术：部署架构方案3(2/2)



软件开发过程：DevOps概述

DevOps（*英文 Development 和 Operations 的组合*）是一组过程、方法与系统的统称，用于促进开发、测试和运维之间的沟通、协作与整合。

目标：在软件交付和部署的过程中沟通合作，以持续和整体化的方式提供更快速更高质量的产出成果。

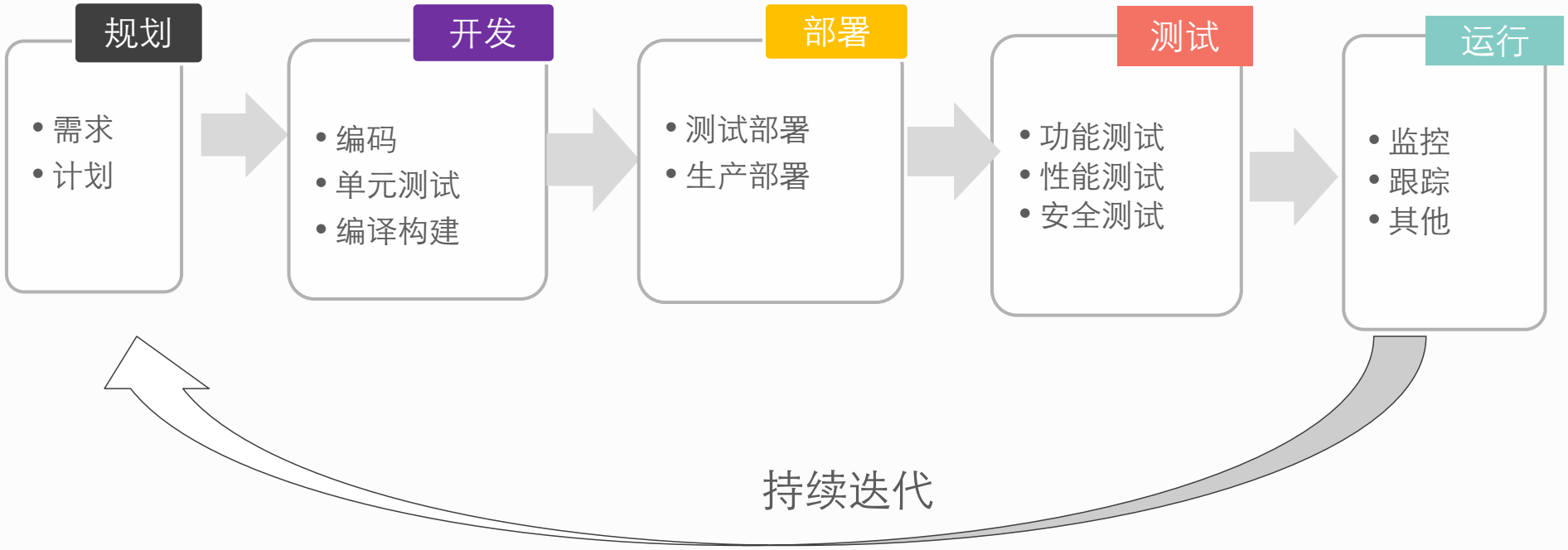
主要体现在：

1. 加速软件交付；
2. 平衡速度、成本、质量和风险；
3. 减少获得反馈的时间。

本质认识：

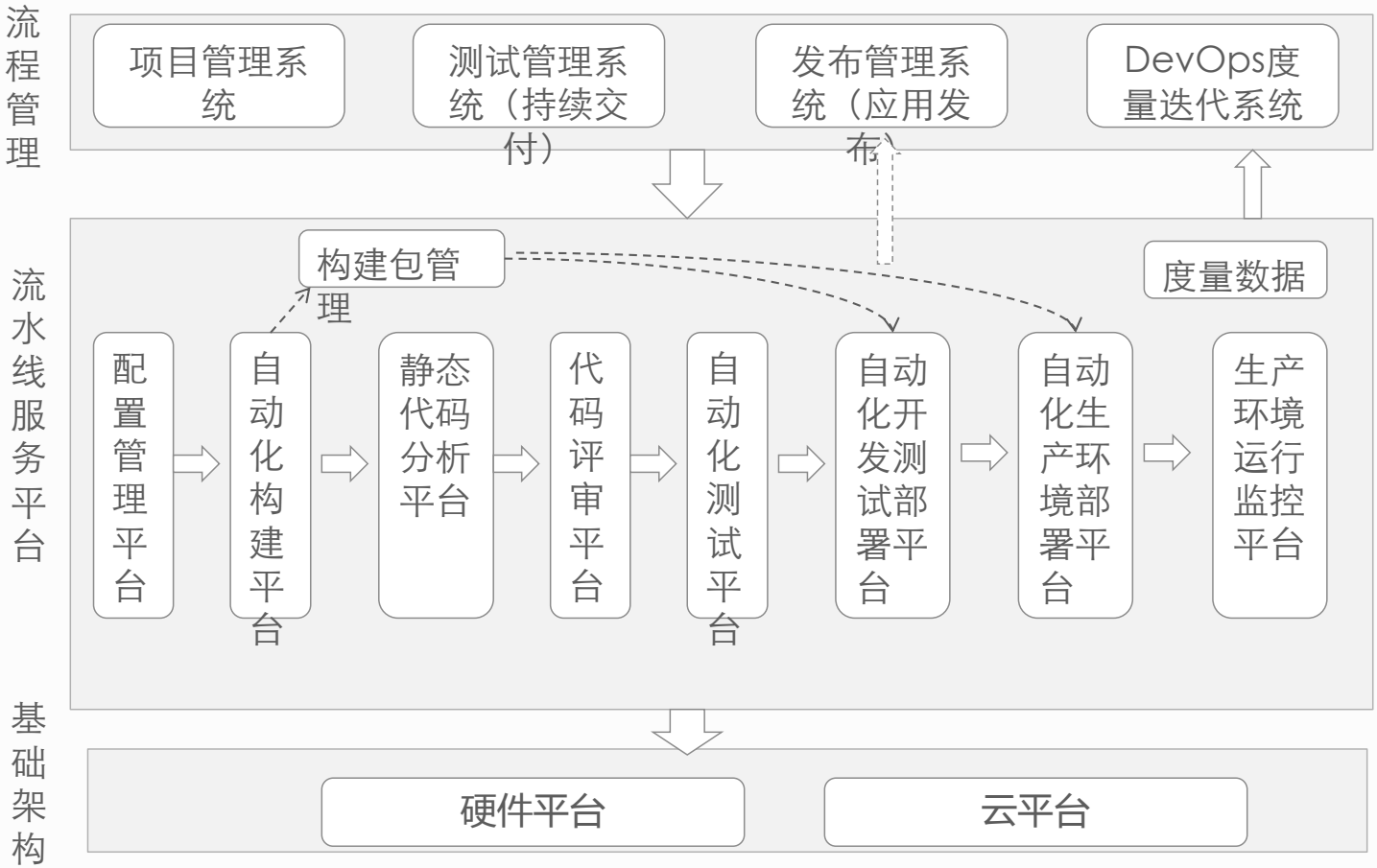
① devops是一种思想，没有成熟的实践方法
② 标准化的程度决定了devops发展的程度
③ devops实施方不但包含运维团队，还涉及开发和测试

软件开发过程：DevOps组成

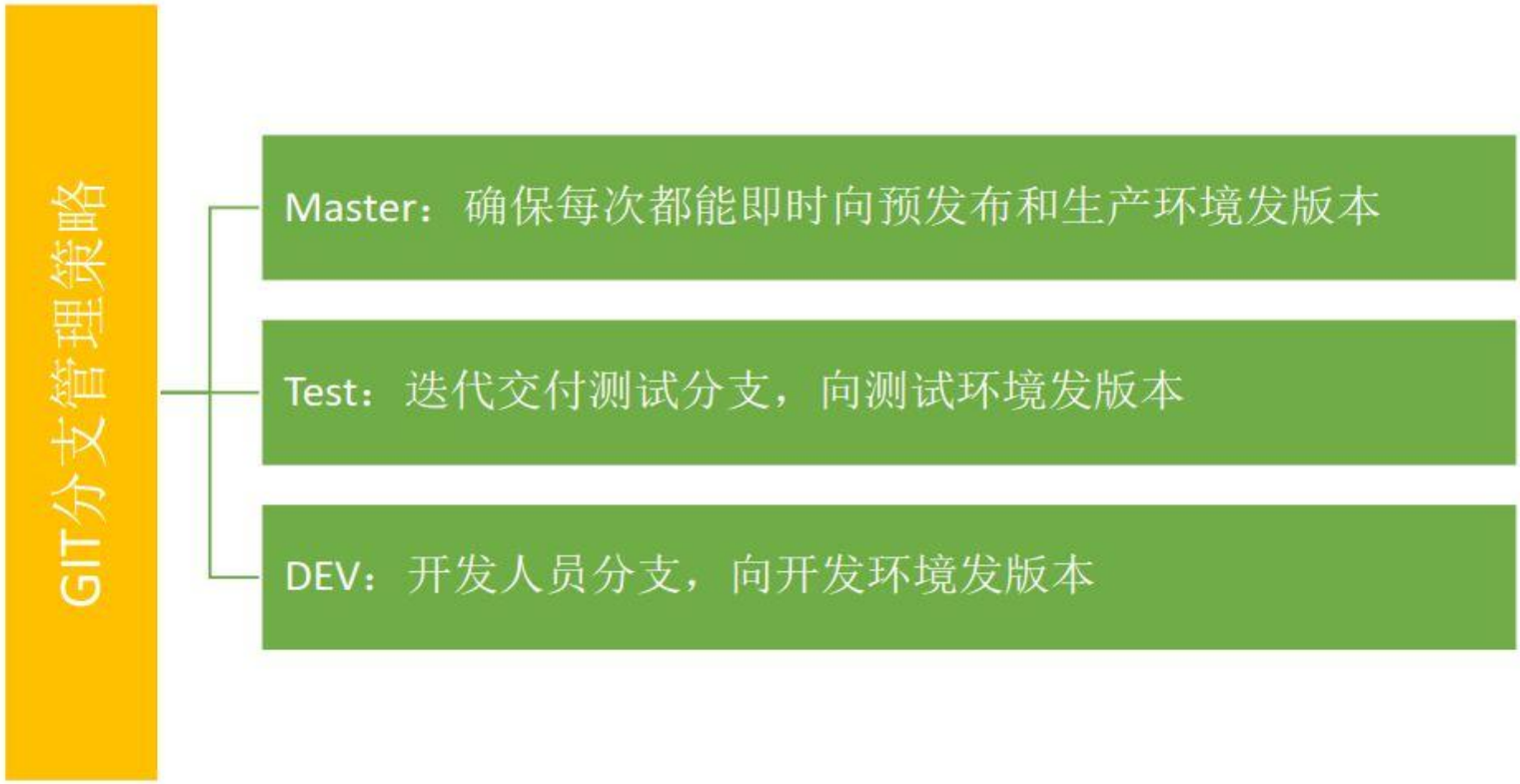


- **沟通协作**：去除壁垒，了解各相关方需求；整体协作，及早获得各相关方的建议和反馈
- **敏捷高效**：强调敏捷开发和自动化测试；强调自动化发布部署
- **持续改进**：持续业务规划；持续测试优化；持续审计监控

软件开发过程：DevOps整体架构

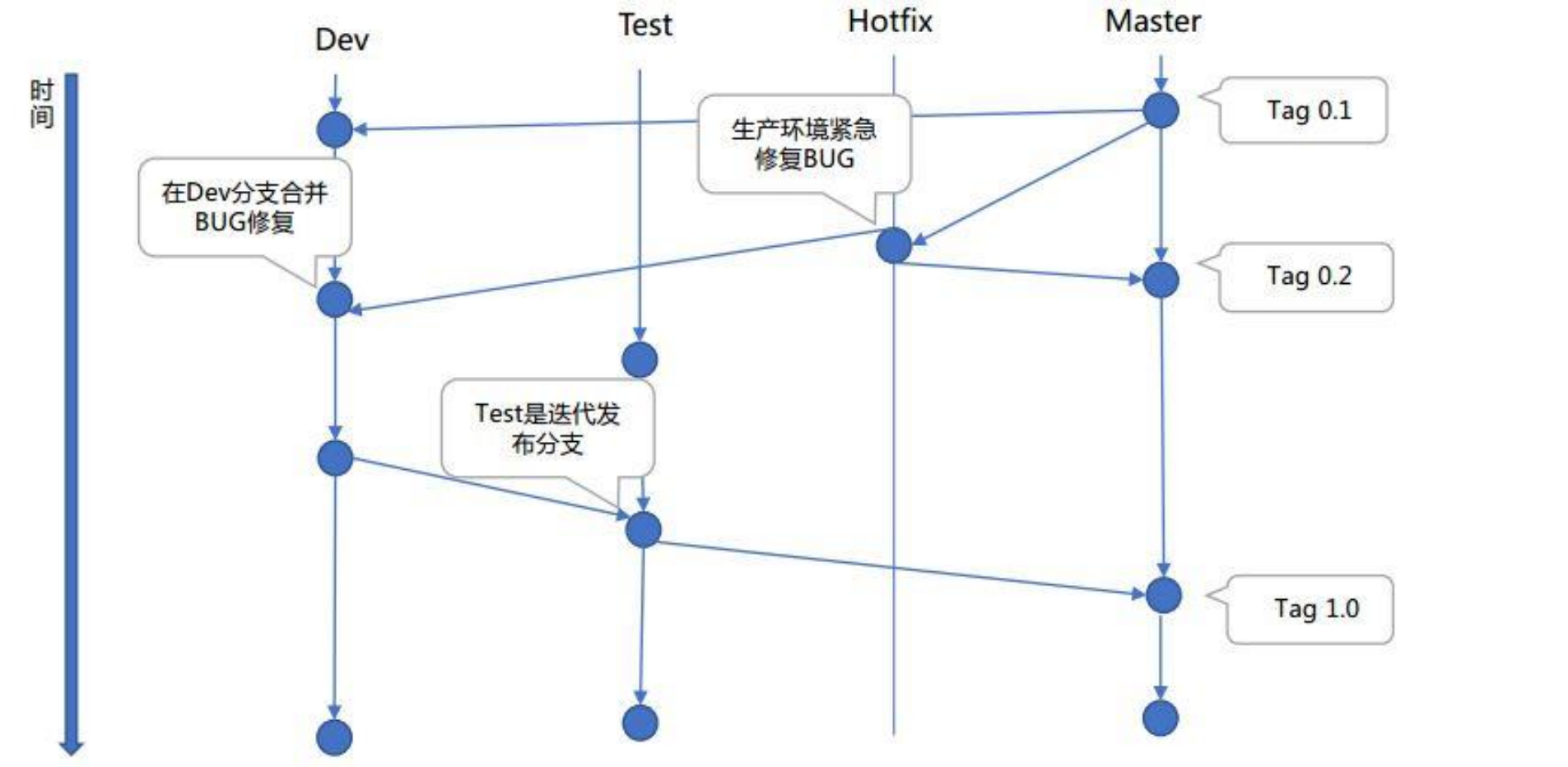


软件开发过程：代码分支规划

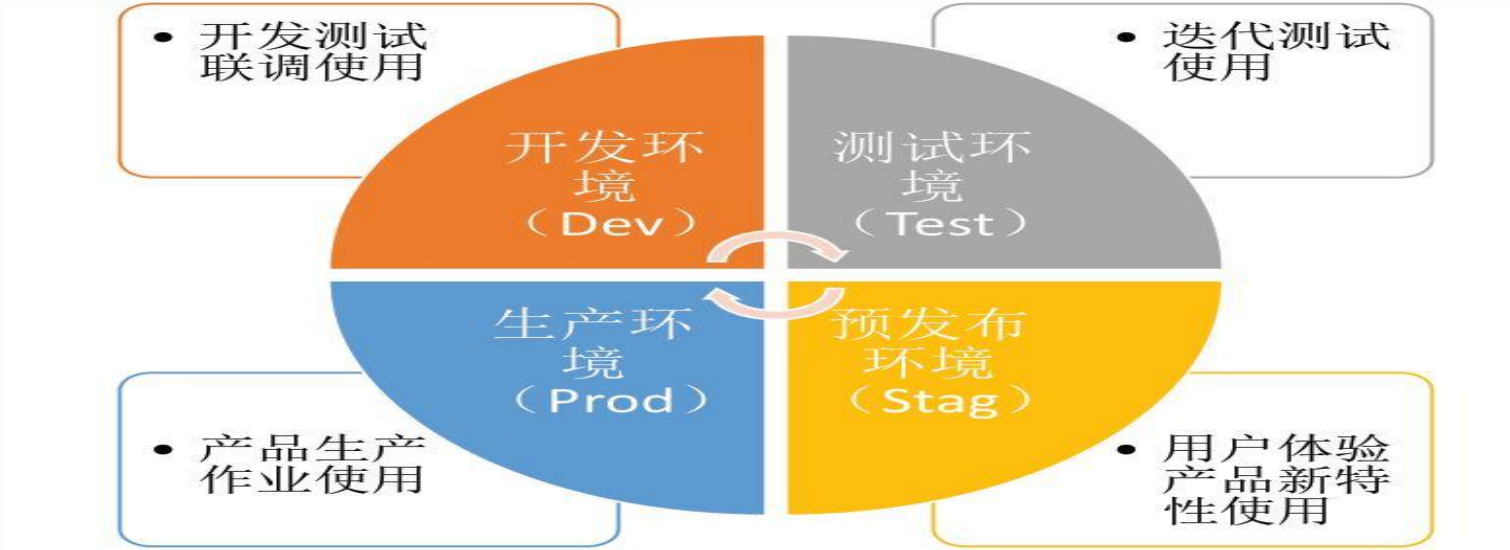


分支类型	命名规范	创建自	合并到	说明
dev	dev/*	master	test	新开发的需求
test	test/*	dev	dev 和 master	迭代新版本的发布
hotfix	hotfix/*	master	develop 和 master	生产环境中发现的紧急 bug 的修复
master	master/*	test/hotfix		预发布环境和生产环境发版本

软件开发过程：代码分支管理

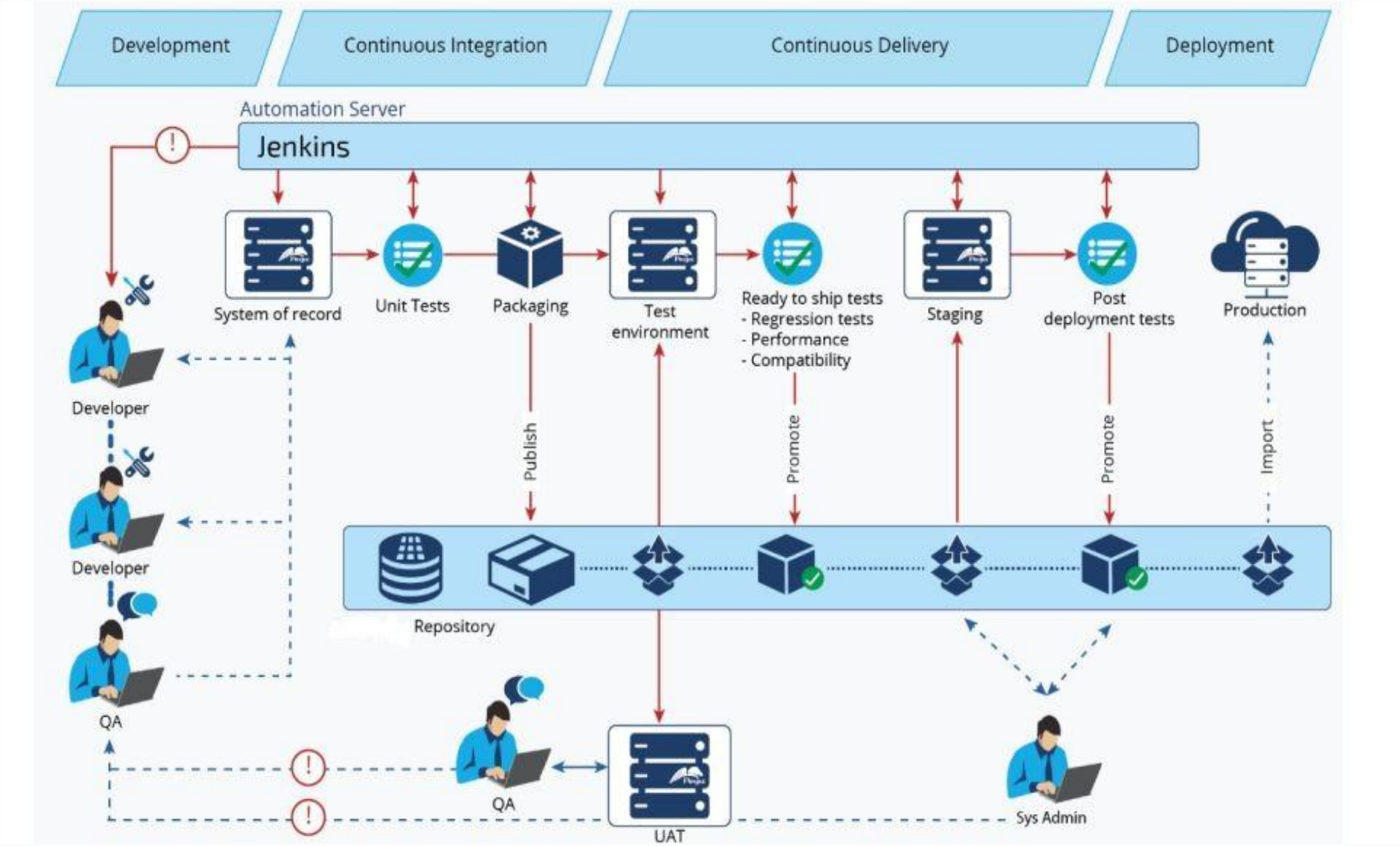


软件开发过程：代码分支和环境



分类	用途	目标代码来源	构建策略	Job责任人	Job权限配置
开发环境（dev）	主要提供给开发人员对代码进行调试	dev分支	轮询版本管理系统dev分支代码变更进行自动构建 手动触发构建	开发leader	开发leader可配置 匿名可执行构建
测试环境（test）	主要提供给测试人员对功能、性能进行测试	test分支	轮询版本管理系统test分支代码变更进行自动构建 手动触发构建	开发leader	开发leader可配置 测试人员可执行构建 匿名可查看
预发布环境（stag）	主要用于正式上线前的测试验证和产品验收	master分支	根据测试人员预发布通知，手动触发构建	DevOps工程师	DevOps工程师可配置 开发leader可查看
生产环境（prod）	提供生产作业的正式环境	master分支	根据测试人员上线通知，手动触发构建	DevOps工程师	DevOps工程师可配置 开发leader可查看

软件开发过程：DevOps理念下的CI/CD



软件开发过程：DevOps实践目标

1. 构建流水线平台，减少人工操作

构建一个持续交付的流水线平台是最基础也是最迫切的，只有通过流水线平台的自动化和持续流动，才能保证在不同阶段、不同节点上产品发布的一致性和稳定性，同时，也才能消除由于人工操作所引入的人为风险，同时提高效率

2. 优化现有开发模式及现有产品架构，消除流动阻碍

需要对现有的开发模式及产品架构做进一步的优化，使整个流水线是顺畅地流动起来。

3. 简化开发测试以及发布部署流程

在通过工具自动化的方式实现产品的持续交付后，由于人工操作的减少，自动化及流水线操作的提高，包括操作过程可追踪性的实现，快速自动回滚操作的实施等，这个时候，在完整的开发测试交付流程中，有些管控步骤可能就是多余的，是可以优化的。因此，实施的第三步就是对整体开发测试发布流程进行优化，去掉冗余的人工评审步骤，从而实现企业级的 DevOps 持续交付流水线

Q&A

Thank you very much !