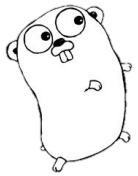
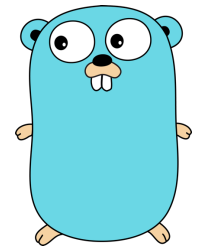
Go语言与微服务

李文周

微服务概述

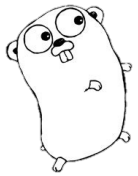
七米（ liwenzhou ）

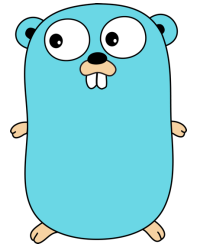




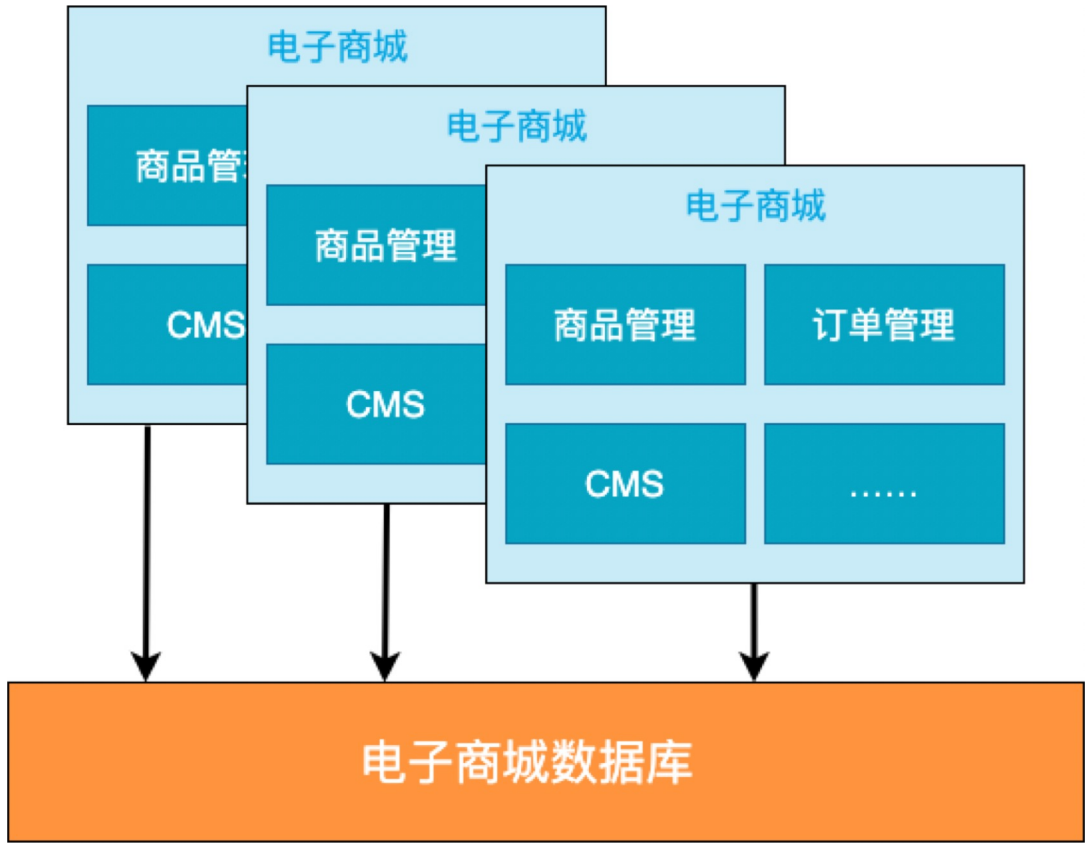
互联网架构演进之路

单体架构 → 垂直架构 → SOA架构 → 微服务架构

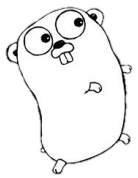


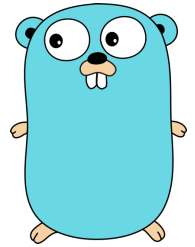


单体架构（电商）

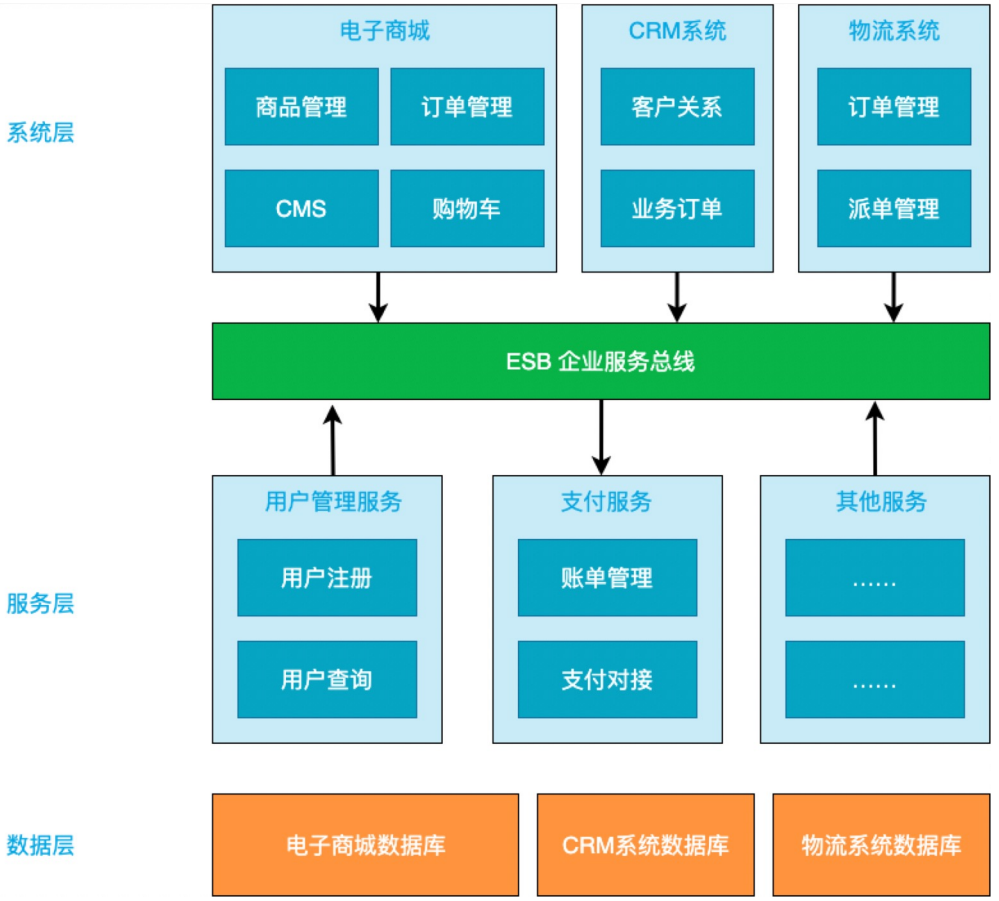


项目初期
CRUD一把梭

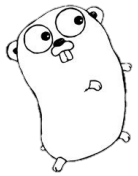


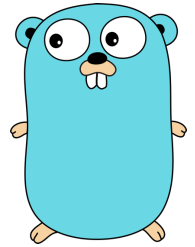


SOA架构（电商）

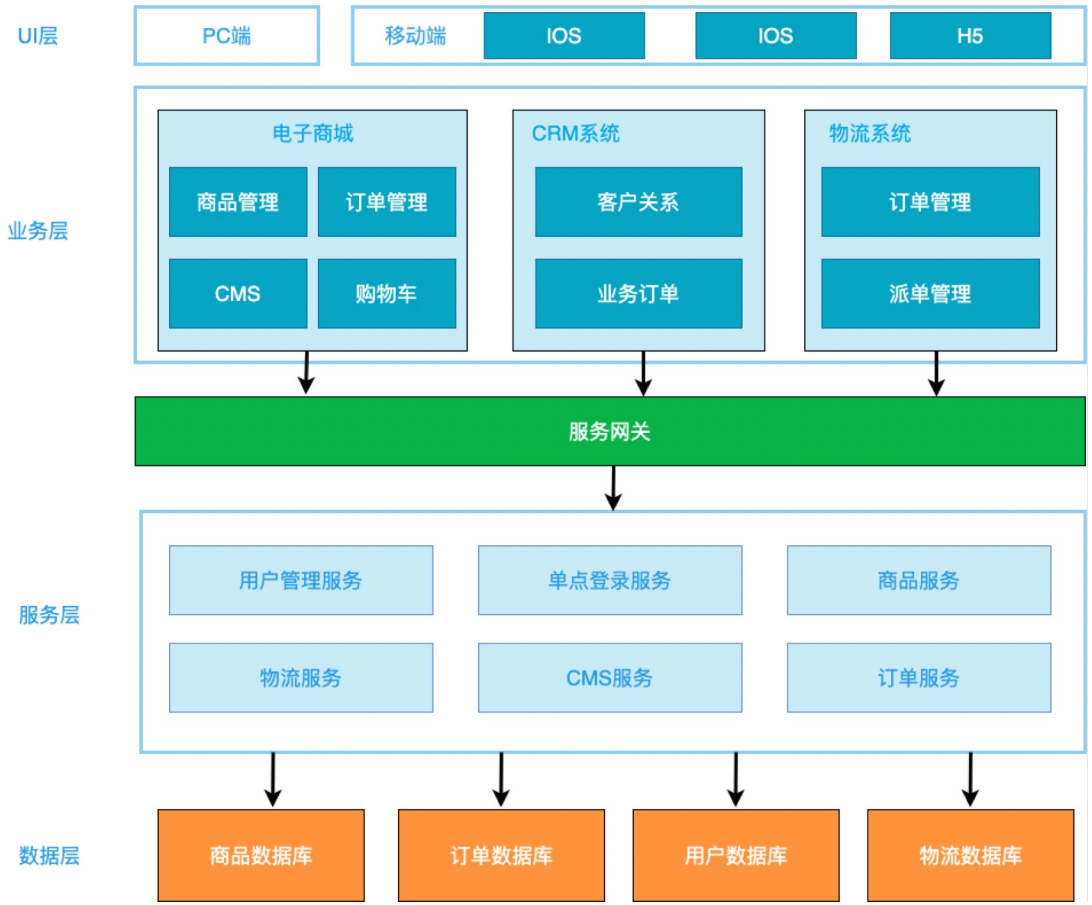


组件化
独立功能

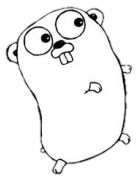


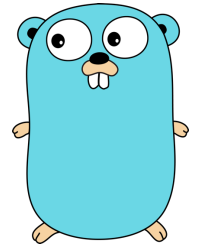


微服务架构（电商）



服务化
独立进程

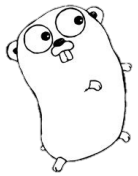


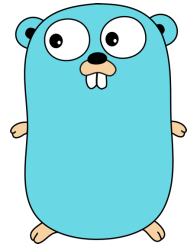


什么是微服务

微服务架构风格 是一种将 **单体应用** 开发为 **一套小型服务** 的方法，每个服务都在 **自己的进程中运行**，并使用 **轻量级的通信机制**(通常是 HTTP 类型的 API)进行通信。这些服务是围绕 **业务能力** 构建的，并且可以通过 **全自动化的部署机制** 进行 **独立部署**。这些服务可以用 **不同的编程语言** 编写，也能使用 **不同的数据存储技术**。

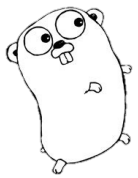
—— James Lewis 和 Martin Fowler (2014)

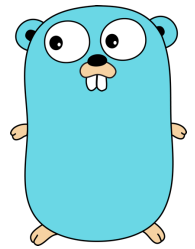




微服务架构优势

- ✓ 快：更注重敏捷开发、持续交付
- ✓ 准：服务粒度小、服务质量精准可控
- ✓ 狠：适用于互联网时代，产品迭代周期更短





微服务架构带来的挑战

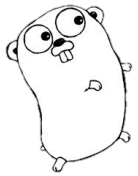
分布式系统的复杂性

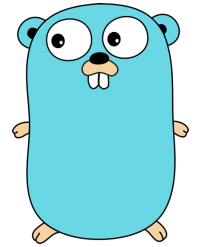
服务依赖管理

数据的一致性保障

测试更加艰难

对DevOps等基础设施的高要求

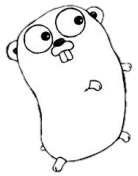


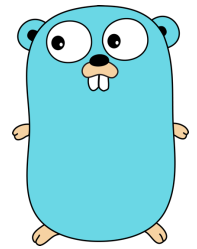


互联网架构演进之路

“

架构的选型，永远只有合适与不合适，
永远没有哪个更好的说法。”



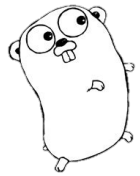


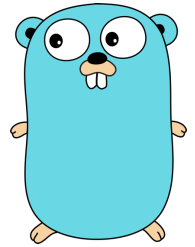
如何进行服务划分?

按业务职能(Business Capability)划分。
由公司内部不同部门提供的职能。例如客户服务部门提供客户服务的职能，财务部门提供财务相关的职能。

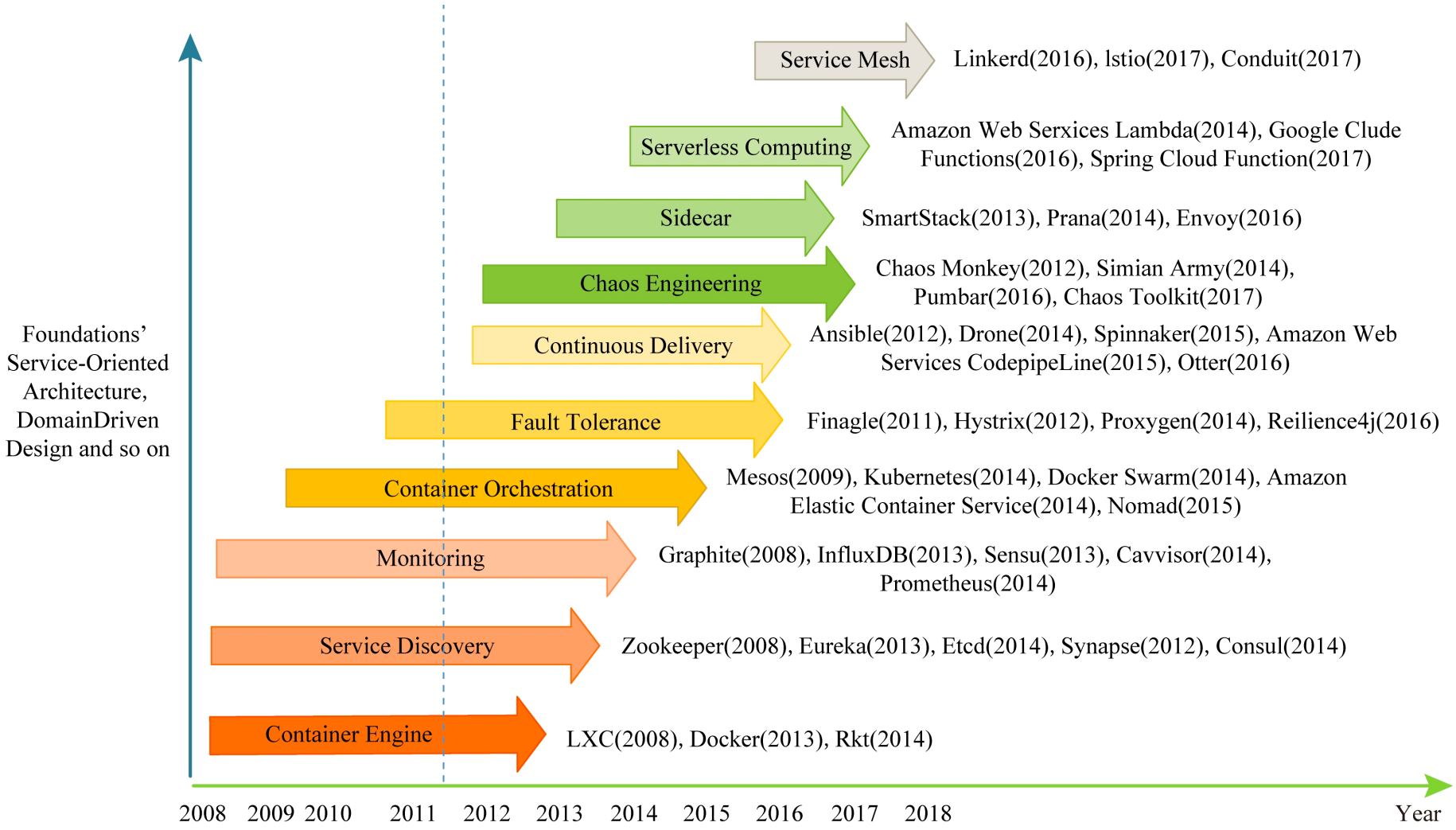
按DDD 的限界上下文(Bounded Context)划分
限界上下文是 DDD 中用来划分不同业务边界的元素，
这里业务边界的含义是“解决不同业务问题”的问题域和对应的解决方案域，
为了解决某种类型的业务问题，贴近领域知识，也就是业务。

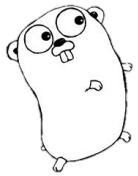
CQRS 将系统中的操作分为两类，即「命令」(Command) 与「查询」(Query)。
命令则是对会引起数据发生变化操作的总称，即我们常说的新增，更新，删除这些操作，都是命令。
而查询则和字面意思一样，即不会对数据产生变化的操作，只是按照某些条件查找数据。
CQRS 的核心思想是将这两类不同的操作进行分离，然后在两个独立的「服务」中实现。
这里的「服务」一般是指两个独立部署的应用。在某些特殊情况下，也可以部署在同一个应用内的不同接口上。

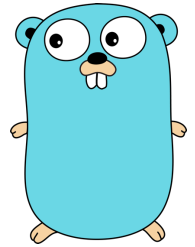




微服务发展历史

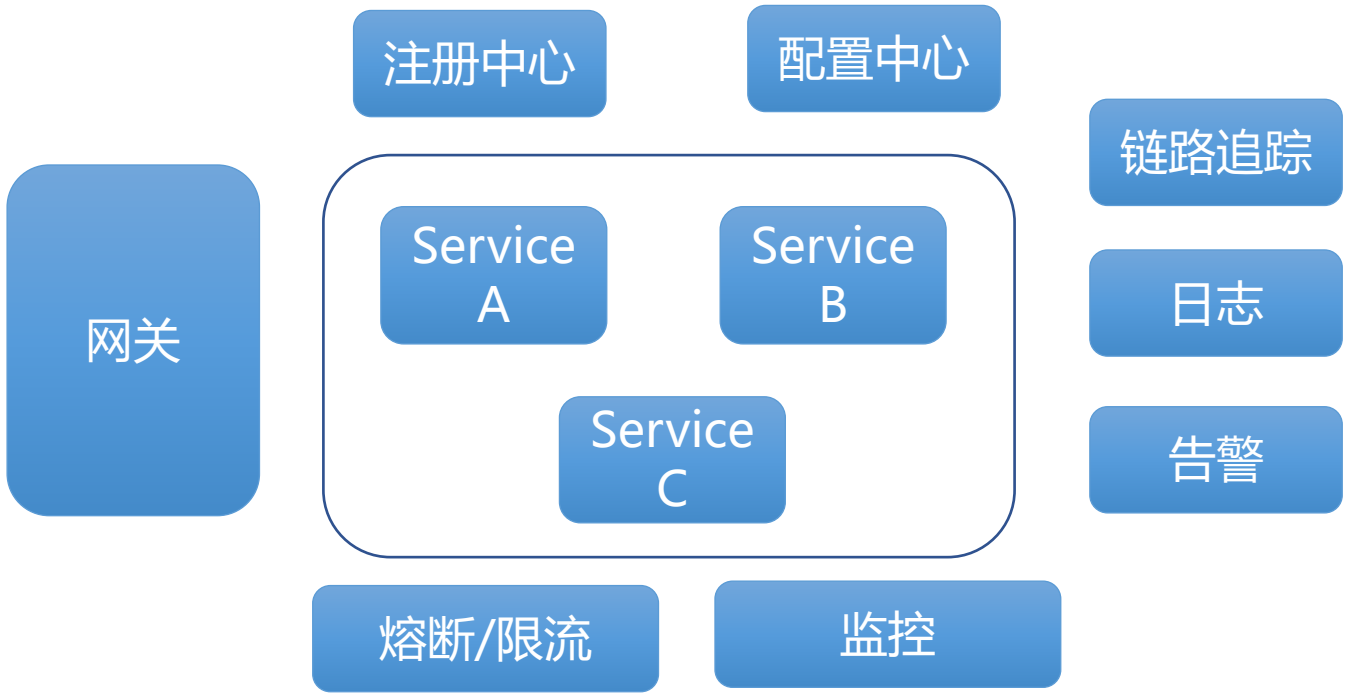

Service Mesh
Linkerd(2016), lstio(2017), Conduit(2017)
Serverless Computing
Amazon Web Serxices Lambda(2014), Google Clude Functions(2016), Spring Cloud Function(2017)
Sidecar
SmartStack(2013), Prana(2014), Envoy(2016)
Chaos Engineering
Chaos Monkey(2012), Simian Army(2014), Pumbar(2016), Chaos Toolkit(2017)
Continuous Delivery
Ansible(2012), Drone(2014), Spinnaker(2015), Amazon Web Services CodepipeLine(2015), Otter(2016)
Fault Tolerance
Finagle(2011), Hystrix(2012), Proxygen(2014), Reilience4j(2016)
Container Orchestration
Mesos(2009), Kubernetes(2014), Docker Swarm(2014), Amazon Elastic Container Service(2014), Nomad(2015)
Monitoring
Graphite(2008), InfluxDB(2013), Sensu(2013), Cavvisor(2014), Prometheus(2014)
Service Discovery
Zookeeper(2008), Eureka(2013), Etcd(2014), Synapse(2012), Consul(2014)
Container Engine
LXC(2008), Docker(2013), Rkt(2014)
Foundations' Service-Oriented Architecture, DomainDriven Design and so on
2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018
Year

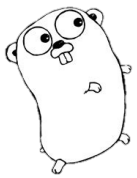


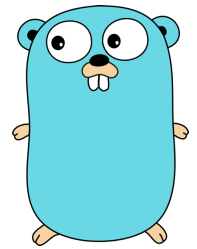


微服务发展历史

第一代：基于RPC的传统服务架构

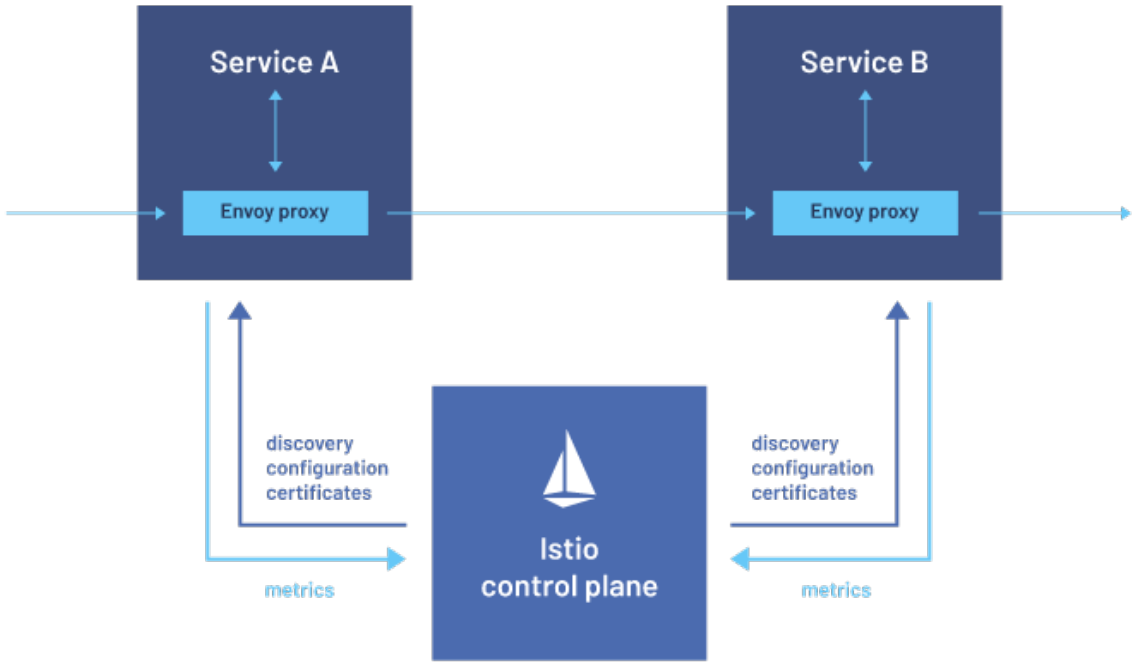


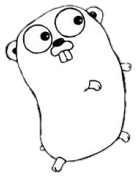


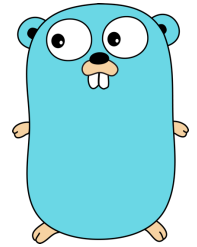


微服务发展历史

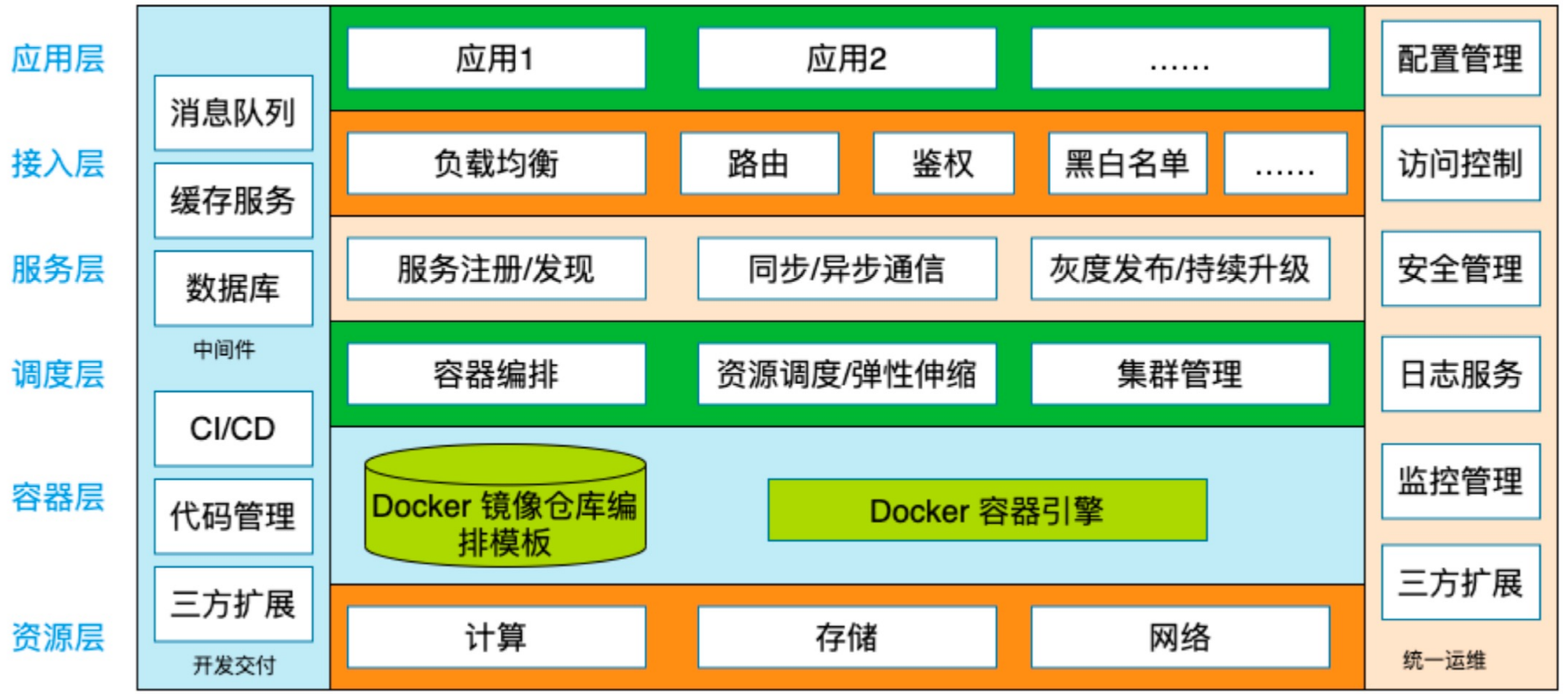
第二代：Service Mesh（istio）

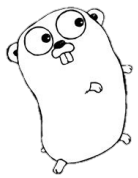


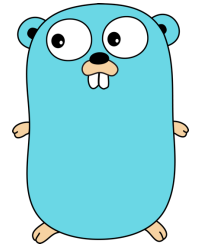




微服务架构分层


应用层
接入层
服务层
调度层
容器层
资源层
消息队列
缓存服务
数据库
中间件
CI/CD
代码管理
三方扩展
开发交付
应用1
应用2
.....
负载均衡
路由
鉴权
黑白名单
.....
服务注册/发现
同步/异步通信
灰度发布/持续升级
容器编排
资源调度/弹性伸缩
集群管理
Docker 镜像仓库编排模板
Docker 容器引擎
计算
存储
网络
配置管理
访问控制
安全管理
日志服务
监控管理
三方扩展
统一运维





微服务核心组件

API网关

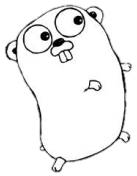
服务注册中心

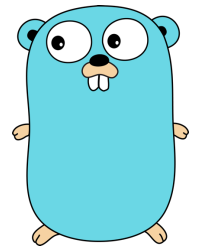
配置中心

服务通信

服务治理

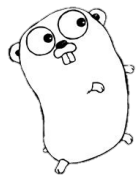
服务监控

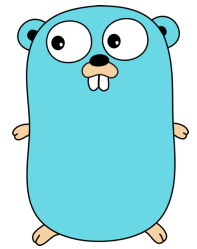




总结

- 了解微服务架构的优势与挑战
- 熟悉微服务架构的核心组件
- 掌握进行微服务划分的依据





推荐阅读：《微服务架构》