

微服务概述

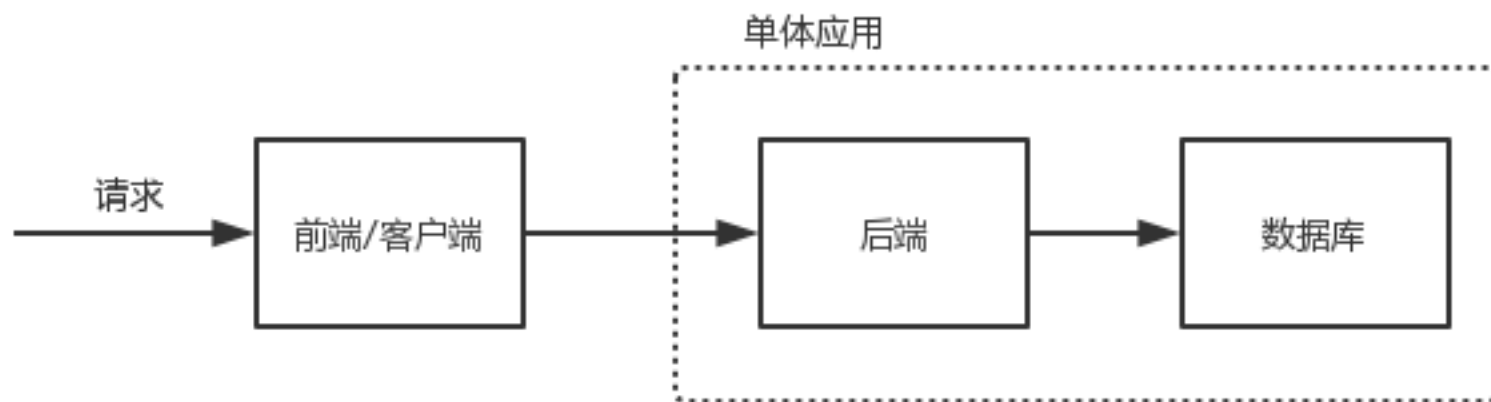
联系QQ: [2816010068](#), 加入会员群

目录

- 单体应用
- 微服务
- 微服务生态

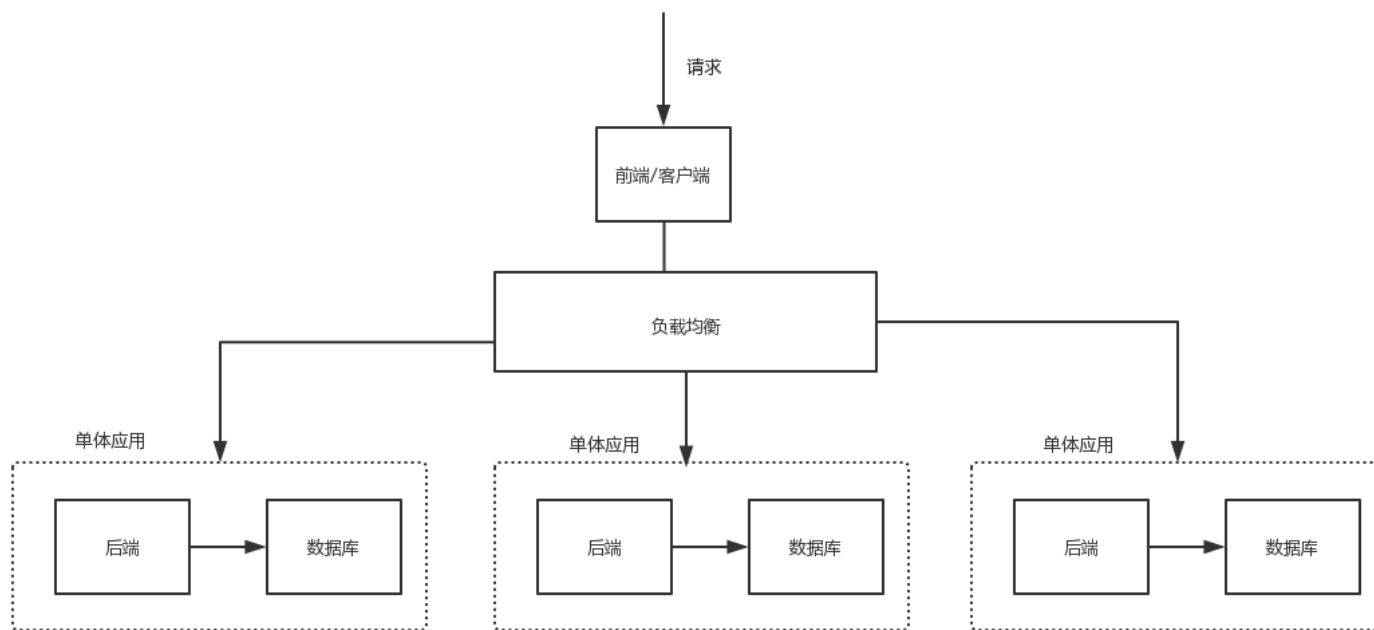
单体应用

- 概念
 - 所有业务功能都在一个应用程序里面
 - 研发人员开发并维护同一个代码库
 - 架构简单，典型的三层架构



单体应用

- 单体应用的横向扩展



单体应用

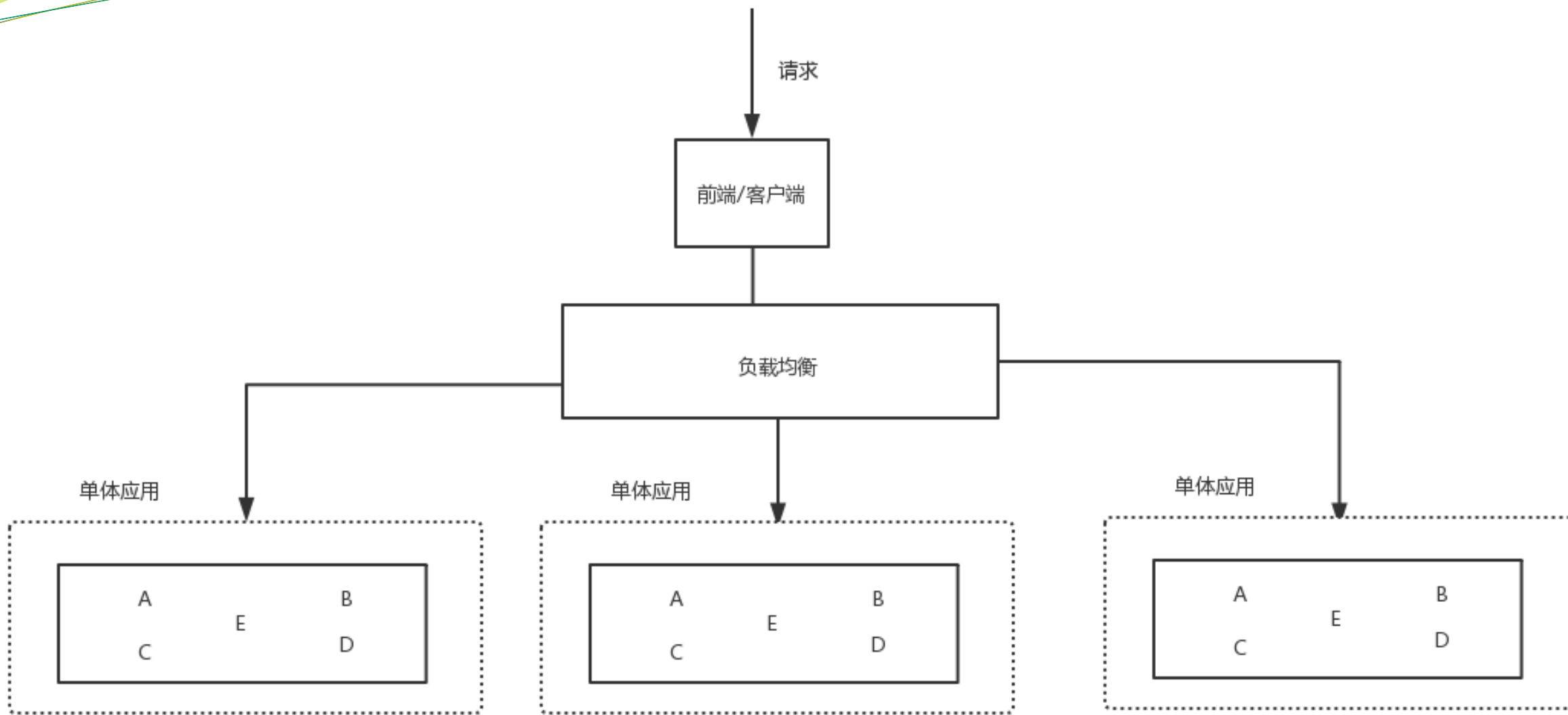
- 单体应用的优势（规模不大）
 - 架构简单，容易上手
 - 部署简单，没有复杂的依赖
 - 测试方便，一旦部署，所有功能就可以测了
- 单体应用的劣势（规模变大后）
 - 复杂度变高，代码越来越庞大
 - 开发效率低，开发协作越来越麻烦
 - 牵一发而动全身，任何一个功能出故障，全部完蛋

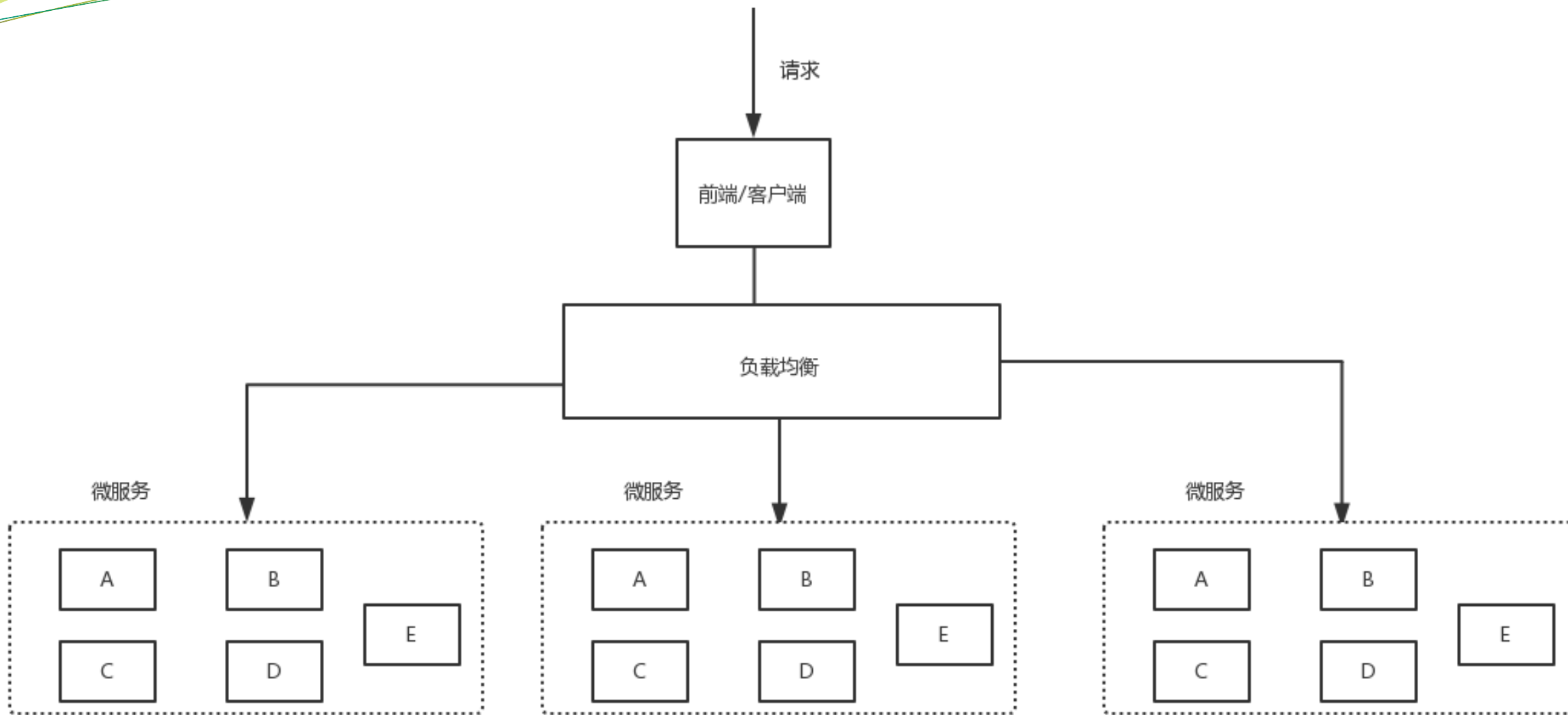
微服务

- 概念

- “微服务”就是微小的服务或应用，比如linux上各种工具：ls, cat, awk, wc, cp, rm等
- 基本原理：让每个服务专注的做好一件事情
- 每个服务单独开发和部署，服务之间是完全隔离的

- 架构





微服务

- 优势

- 迭代周期短，极大的提升研发效率
- 独立部署，独立开发
- 可伸缩性好，能够针对指定的服务进行伸缩
- 故障隔离，不会互相影响

- 缺点

- 复杂度增加，一个请求往往要经过多个服务，请求链路比较长
- 监控和定位问题困难
- 服务管理比较复杂

微服务

- 落地微服务关键因素
 - 配套设施
 - 微服务框架研发和维护
 - 打包、版本管理、上线平台支持
 - 硬件层支持，比如容器和容器调度
 - 服务治理平台支持，比如分布式链路追踪和监控
 - 测试自动化支持，比如上线前自动化case
 - 组织架构
 - 微服务框架研发团队
 - 私有云研发团队
 - 测试平台研发团队

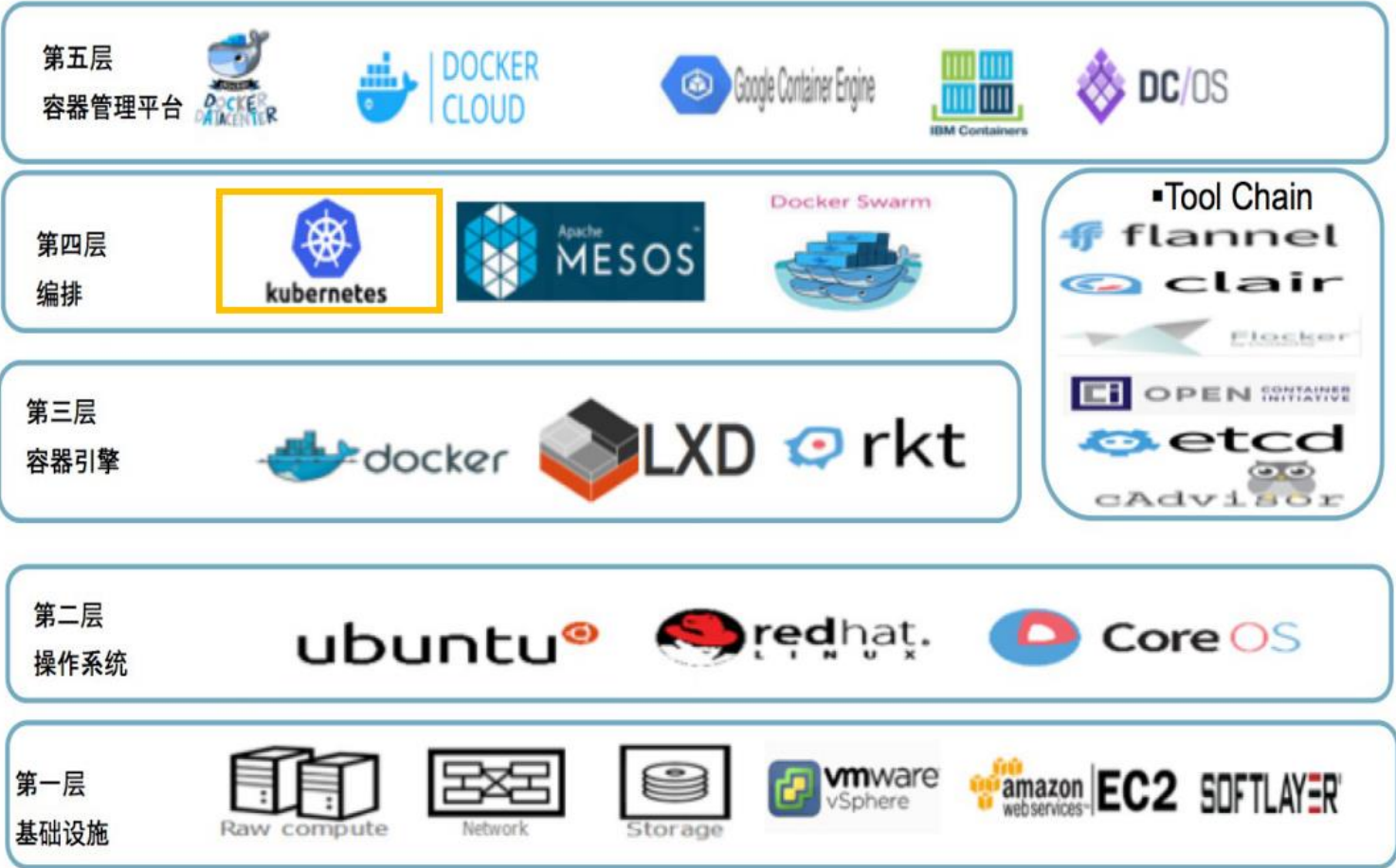
微服务生态



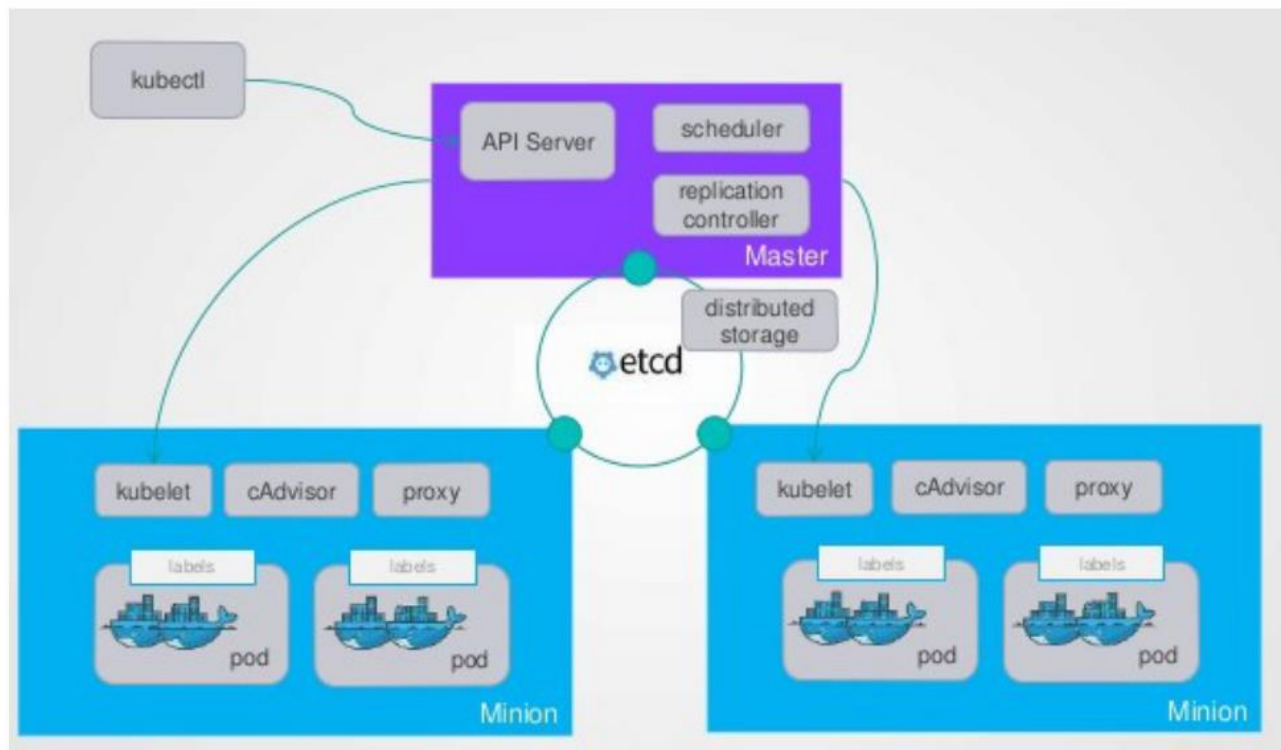
硬件层

- 物理服务器管理
- 操作系统管理
- 配置管理
- 资源隔离和抽象
- 主机监控和日志

硬件层架构



Kubernetes容器分配



- ❑ kube-apiserver 提供统一接口

- ❑ kube-scheduler 负责资源与Pod的匹配

- ❑ Kube-controller-manager 负责“资源”管理同步

- ❑ Kube-proxy 负责k8s 中的网络配置

- ❑ Kubelet 管理Pod 的生命周期

通信层

- 网络传输
- RPC（rpc客户端和rpc服务端）
- 服务发现
- 服务注册
- 负载均衡
- 消息传递

网络传输

- HTTP+RESTFUL
 - GET、POST、PUT、DELETE
- TCP RPC调用
 - Thrift
 - Dubbox
 - Grpc
- 消息传递
 - JSON
 - Thrift
 - protobuf

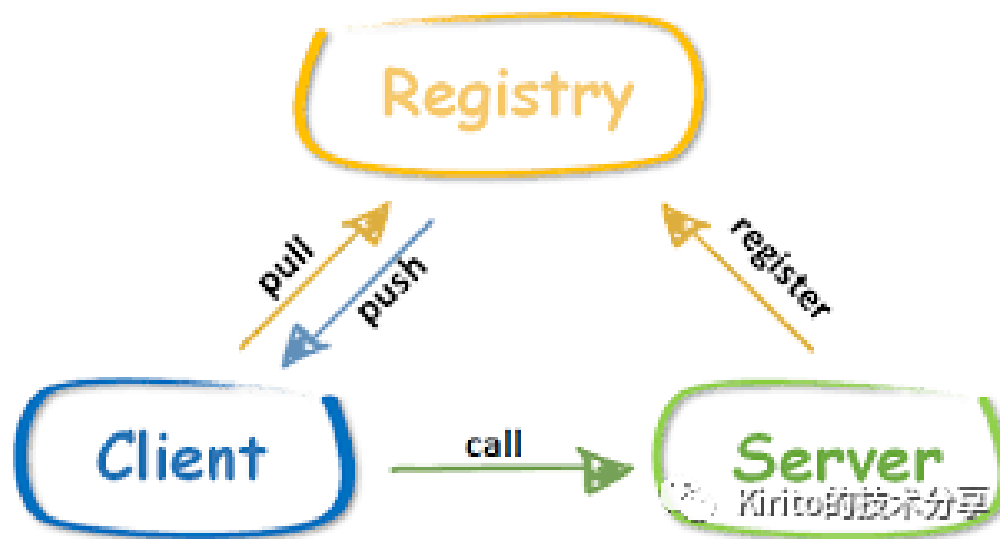
服务注册&服务发现

- 分布式数据存储
 - Consul
 - Etcd
 - Zookeeper

CAP原理

- CAP原理
 - C: consistency, 每次总是能够读到最近写入的数据或者失败
 - A: available, 每次请求都能够读到数据
 - P: partition tolerance, 系统能够继续工作, 不管任意个消息由于网络原因失败

服务注册和发现



消息传递

- 序列化和反序列化
 - Json
 - Protobuf
 - Thrift
 - Msgpack
- 数据对比
 - <https://tech.meituan.com/2015/02/26/serialization-vs-deserialization.html>
 - <https://github.com/eishay/jvm-serializers/wiki>
 - <http://blog.mirthlab.com/2009/06/01/thrift-vs-protocol-buffers-vs-json/>

支撑平台

- 私有云平台
- 服务管理平台
- 监控报警平台
- 测试&构建&发布平台
- 日志检索平台
- 服务治理平台
- Metrics平台