China · Beijing

构建大规模微服务架构应用

何李石 @ikbear

七牛云首席布道师

ArchSummit
全球架构师峰会 2016

[北京站]

主办方 Geekbang. 极客邦科技 InfoQ

About

《Go 语言程序设计》联合译者

5 年云服务，负责七牛对外技术布道

七牛云：数据存储、处理，直播云，容器云，大数据，机器学习云

Microservices

App: 七牛云富媒体处理平台应用

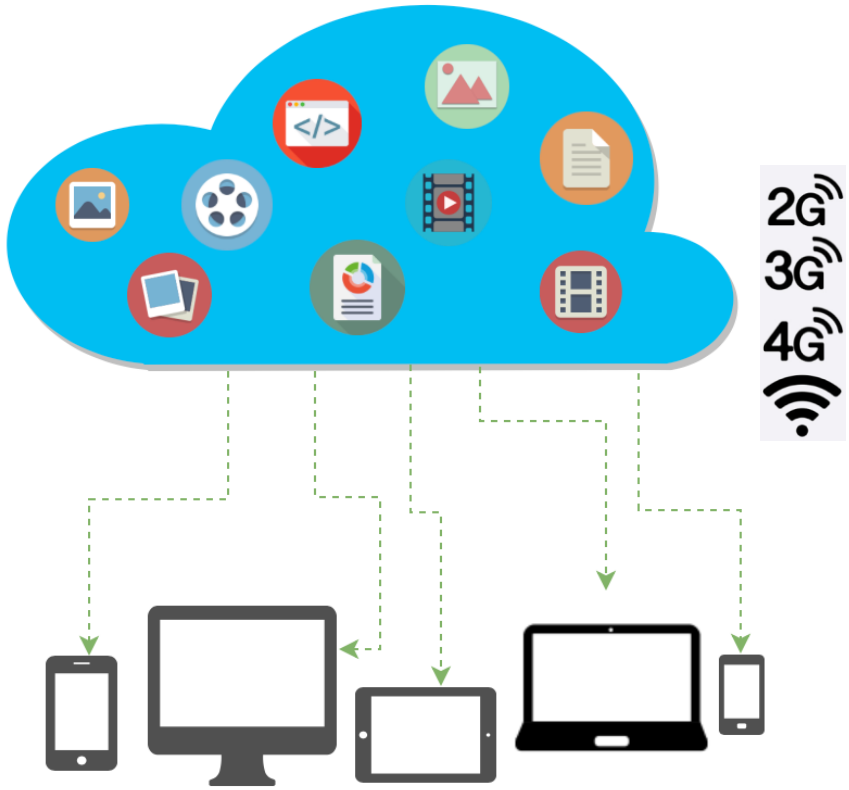
Infra: 面向大规模的服务架构

CI & CD & Ops: 构建自动化交付流程

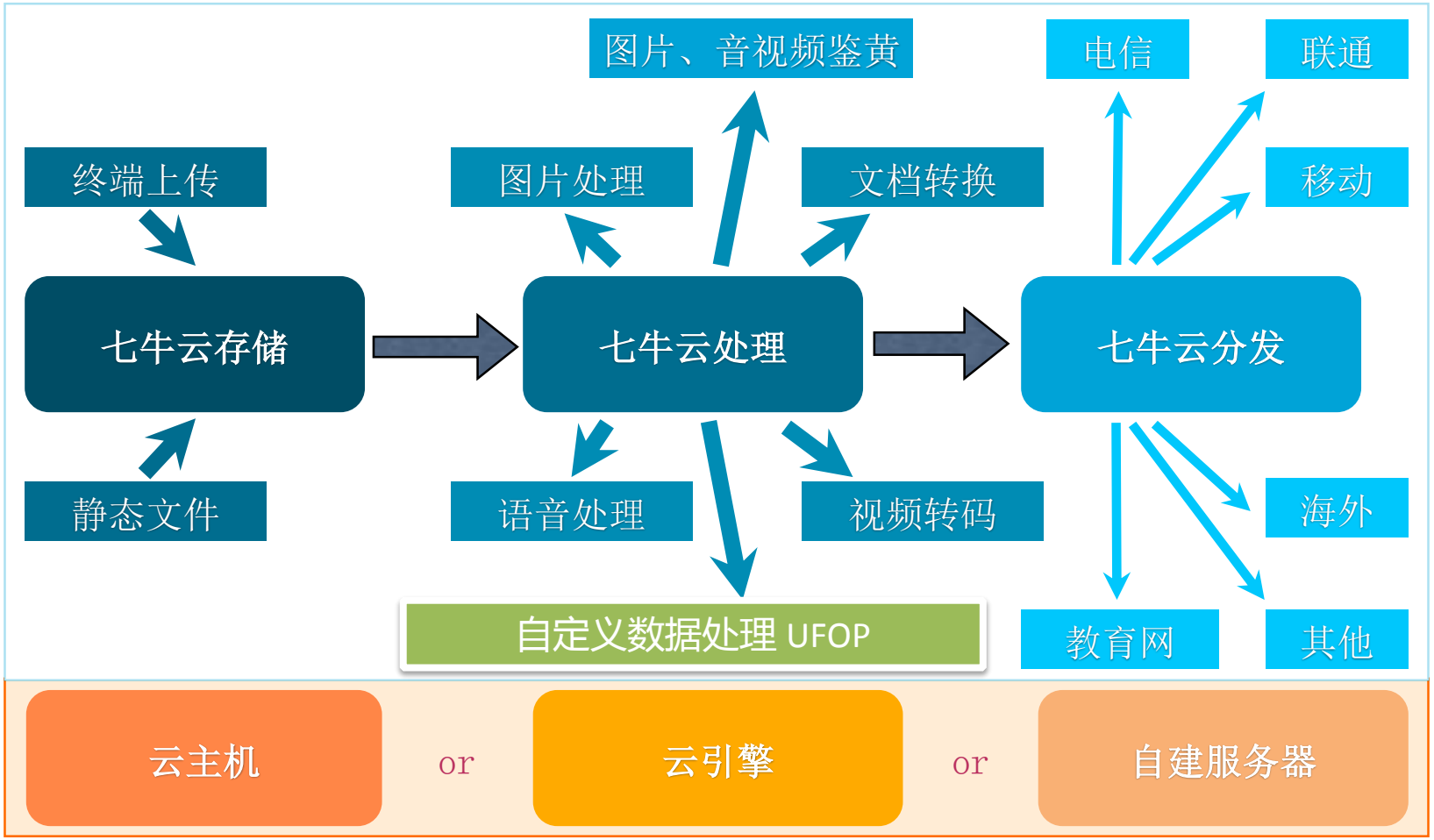
微服务架构：重新定义大规模应用设计

复杂世界的数据处理需求

- 多终端设备（硬件）
- 多 OS 平台（软件）
- 多尺寸观看、播放
- 多网络适配



基于云端的数据处理


图片、音视频鉴黄
电信
联通
终端上传
图片处理
文档转换
移动
七牛云存储
七牛云处理
七牛云分发
静态文件
语音处理
视频转码
海外
自定义数据处理 UFOP
教育网
其他
云主机
or
云引擎
or
自建服务器

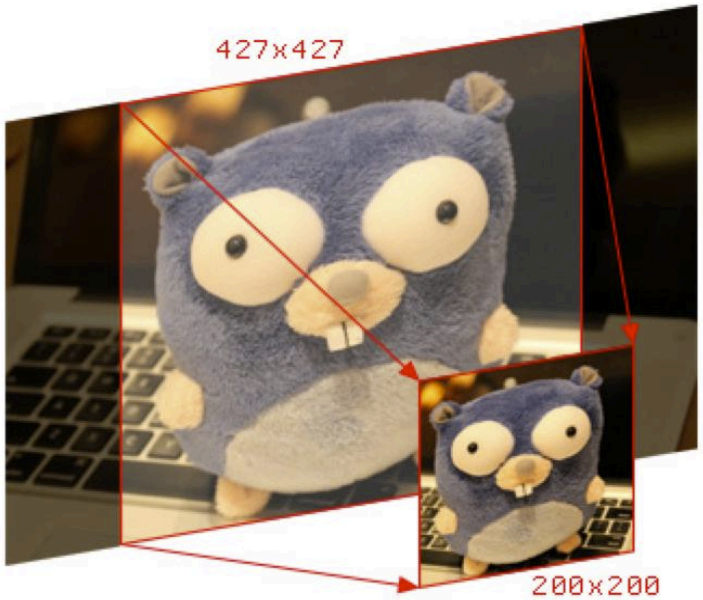
图片缩放裁剪

示例 1.3 裁剪正中部分，等比缩小生成200x200缩略图

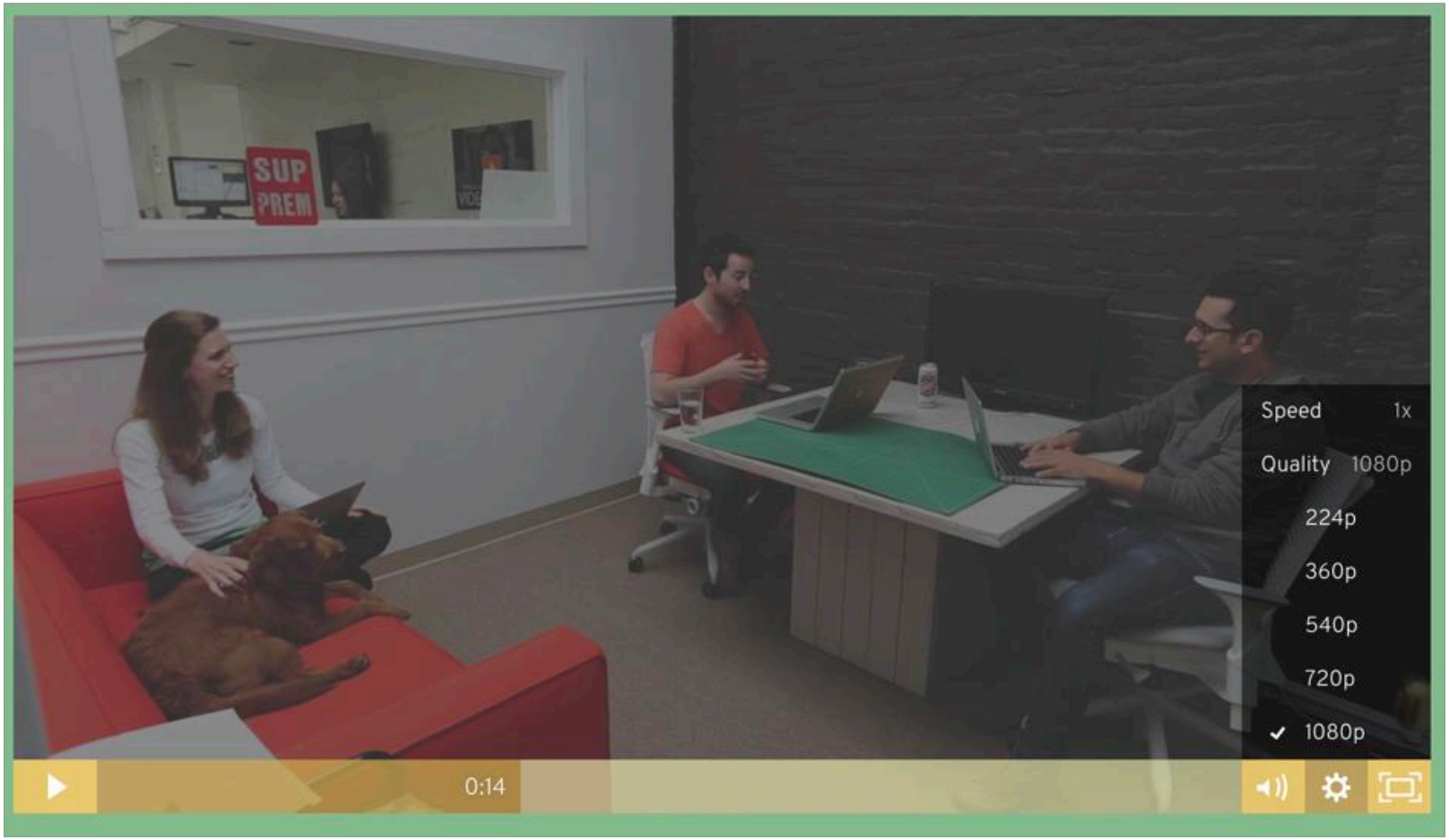
http://qiniuphotos.qiniudn.com/gogopher.jpg

⇒

http://qiniuphotos.qiniudn.com/gogopher.jpg?imageView/1/w/200/h/200

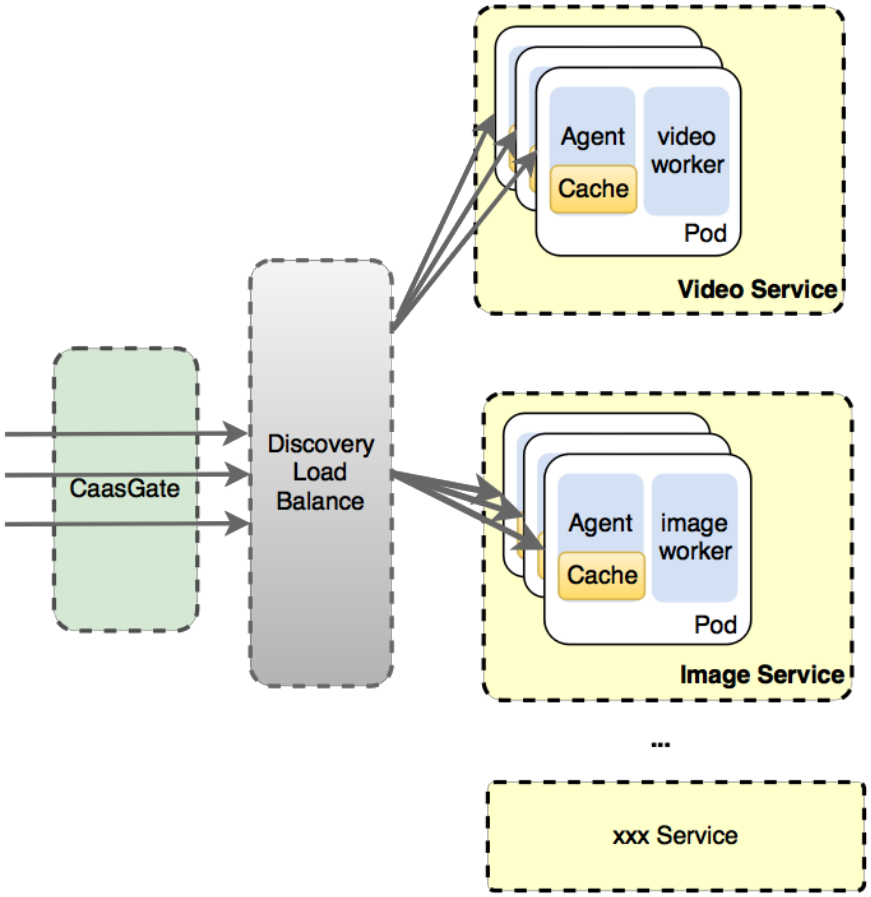


音视频自适应播放


SUP
PREM
Speed 1x
Quality 1080p
224p
360p
540p
720p
1080p
0:14

基本架构

- API Gateway: 访问控制
- Load Balance
- Image/Video Service 混合部署
- 客户自定义 Service 部署



Microservices

App: 七牛云富媒体处理平台应用

Infra: 面向大规模的服务架构

CI & CD & Ops: 构建自动化交付流程

微服务架构：重新定义大规模应用设计

大规模服务架构挑战

- 大规模：处理海量请求（300亿/天）
- 动态伸缩：突发请求、业务增长（早、中、晚高峰）
- 团队、技术架构异构（Go / Python / Java）
- ……

微服务化尝试

- 服务构建：基于最佳实践的微服务框架
- 服务治理：注册与发现
- 分布式系统设计：基于 Event Sourcing 的设计模式

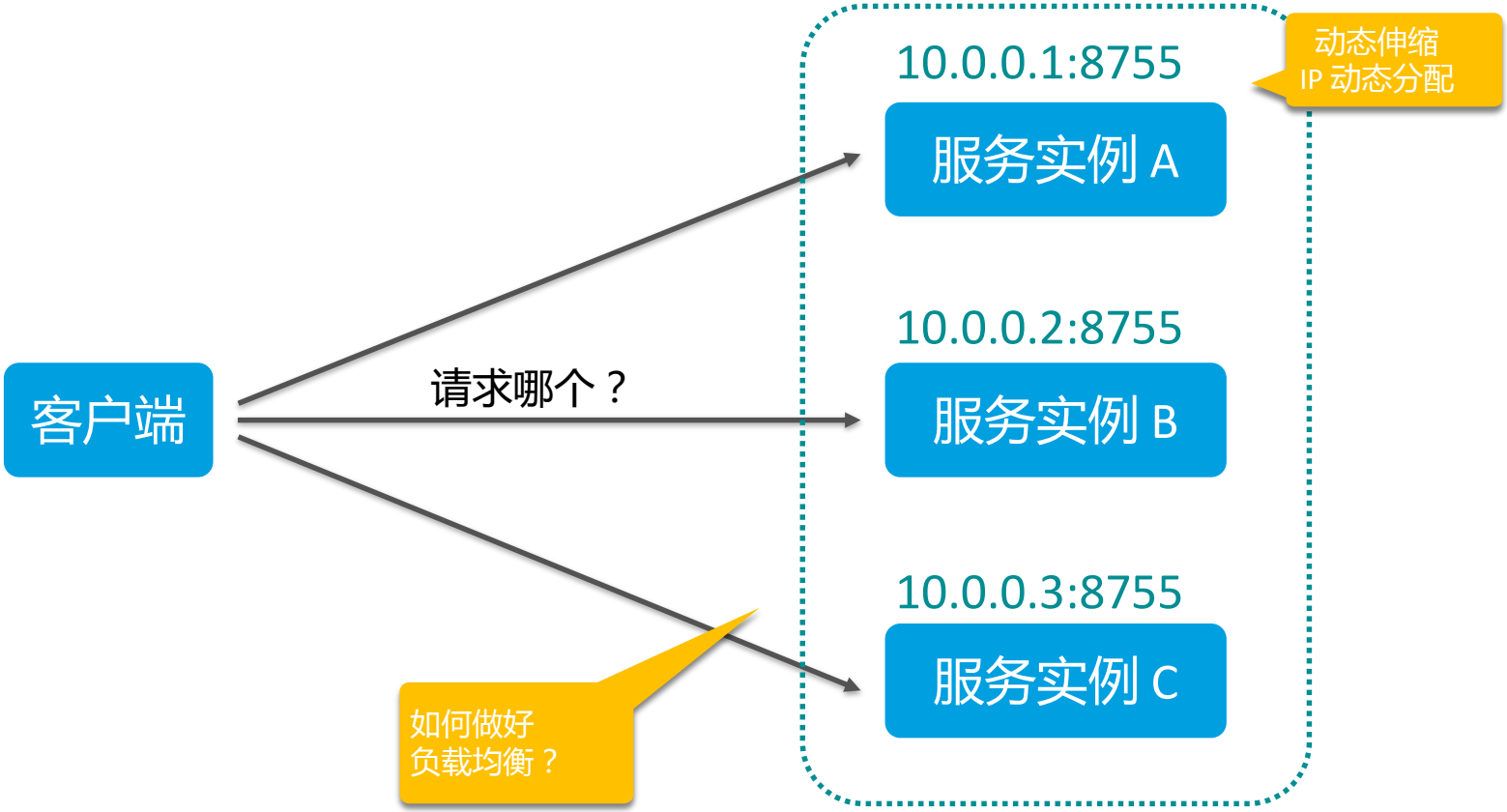
服务构建：Kite Micro-service Framework

microservice.go

```go
k := kite.New("math", "1.0.0")
k.Config.Port = 3636

// 创建一个 RPC 方法: square
k.HandleFunc("square", func(r *kite.Request) (interface{}, error) {
        a := r.Args.One().MustFloat64()
        result := a * a
        return result, nil
}).DisableAuthentication()
```

多实例部署的问题


客户端
请求哪个？
10.0.0.1:8755
服务实例 A
10.0.0.2:8755
服务实例 B
10.0.0.3:8755
服务实例 C
动态伸缩
IP 动态分配
如何做好
负载均衡？

服务注册与发现

服务注册

microservice-register.go

```
1  k := kite.New("first", "1.0.0")
2  k.Config.Port = 6000
3
4  k.HandleFunc("square", func(r *kite.Request) (interface{}, error) {
5          a := r.Args.One().MustFloat64()
6          return a * a, nil
7  })
8
9  // 注册微服务
10 k.Register(&url.URL{Scheme: "http", Host: "localhost:6000/kite"})
```

服务发现

microservice-discover.go

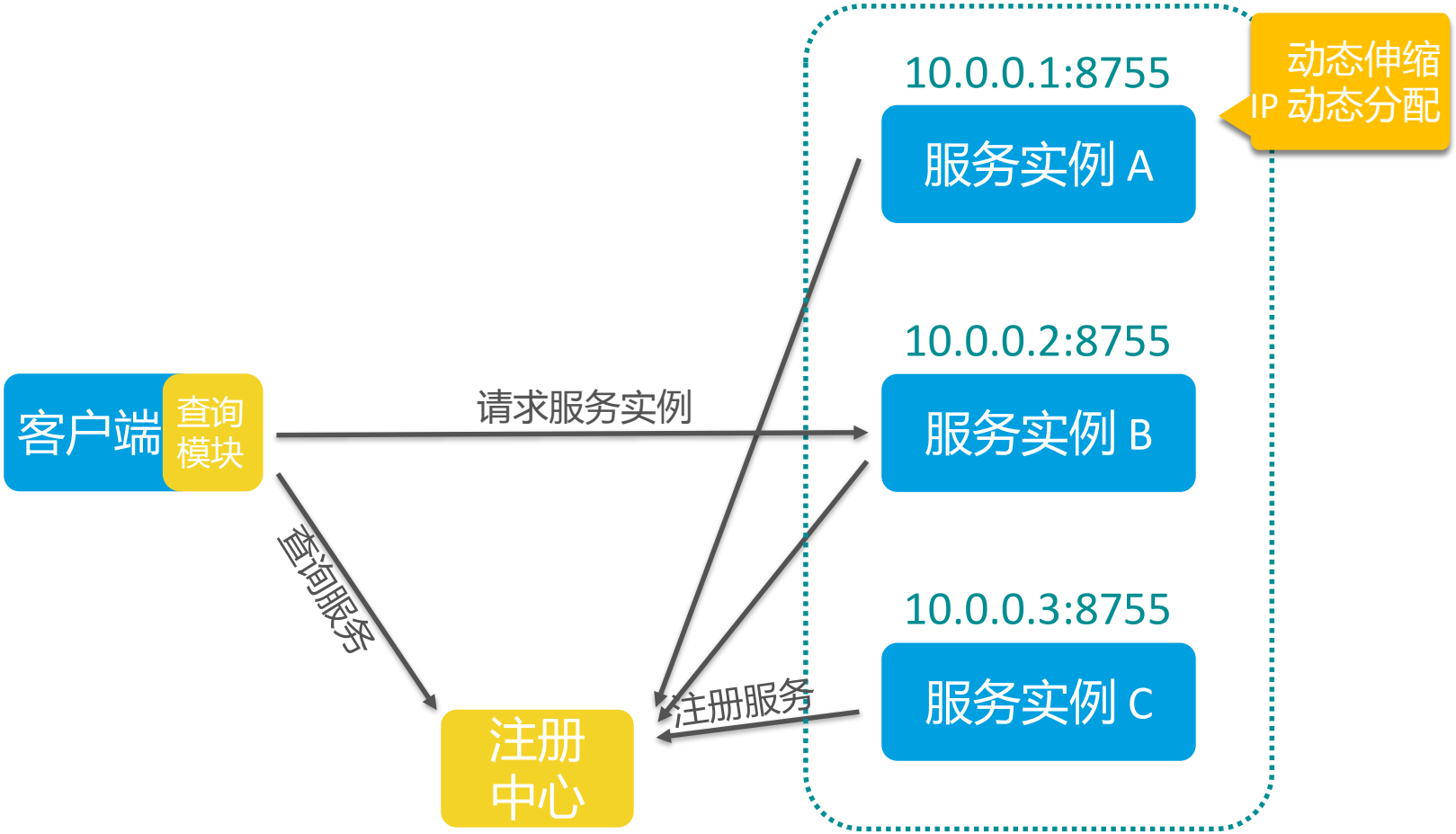
```go
k := kite.New("second", "1.0.0")

kites, _ := k.GetKites(&protocol.KontrolQuery{
        Username:    k.Config.Username,
        Environment: k.Config.Environment,
        Name:        "first",
})

// 查询的结果中可能返回多个微服务（名字都是 "first"）
client := kites[0]
client.Dial()

// 调用指令 "square" 求平方
response, _ := client.Tell("square", 4)
```

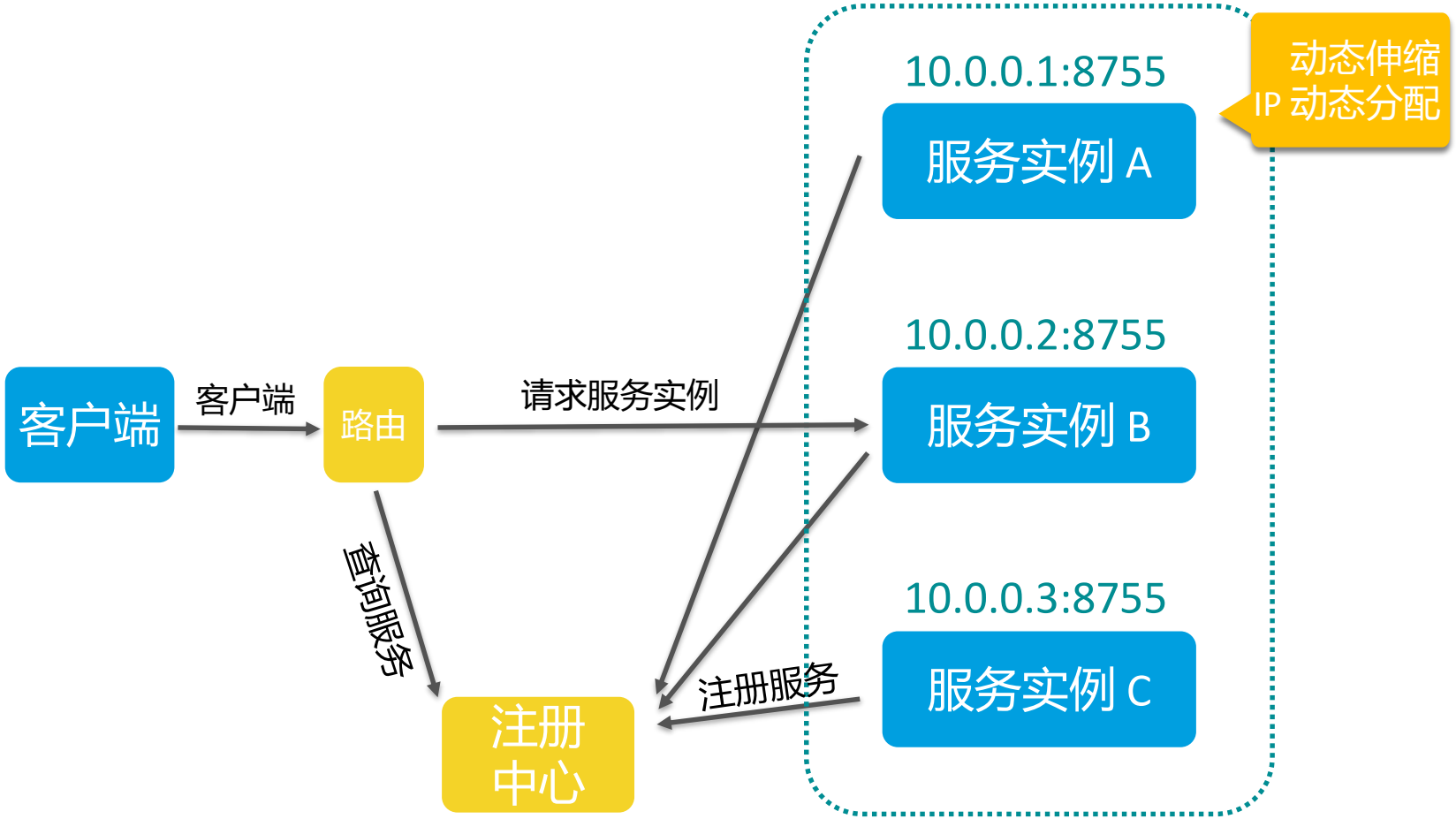
客户端查询模式


10.0.0.1:8755
服务实例 A
动态伸缩
IP 动态分配
10.0.0.2:8755
服务实例 B
10.0.0.3:8755
服务实例 C
客户端
查询模块
请求服务实例
查询服务
注册服务
注册中心

客户端查询模式

- **优点：**
  - 「注册中心」在服务之外维护，使用简单，对已有的微服务架构侵入小；
  - 客户端直接请求后端实例，查询完成后请求链路不需要经过其它中间环节；
- **缺点：**
  - 客户端和「注册中心」绑定；
  - 客户端的实现取决于具体语言或者框架，每个客户端都得自己去实现；

服务端查询模式


客户端
客户端
路由
请求服务实例
查询服务
注册
中心
注册服务
10.0.0.1:8755
服务实例 A
10.0.0.2:8755
服务实例 B
10.0.0.3:8755
服务实例 C
动态伸缩
IP 动态分配

服务端查询模式

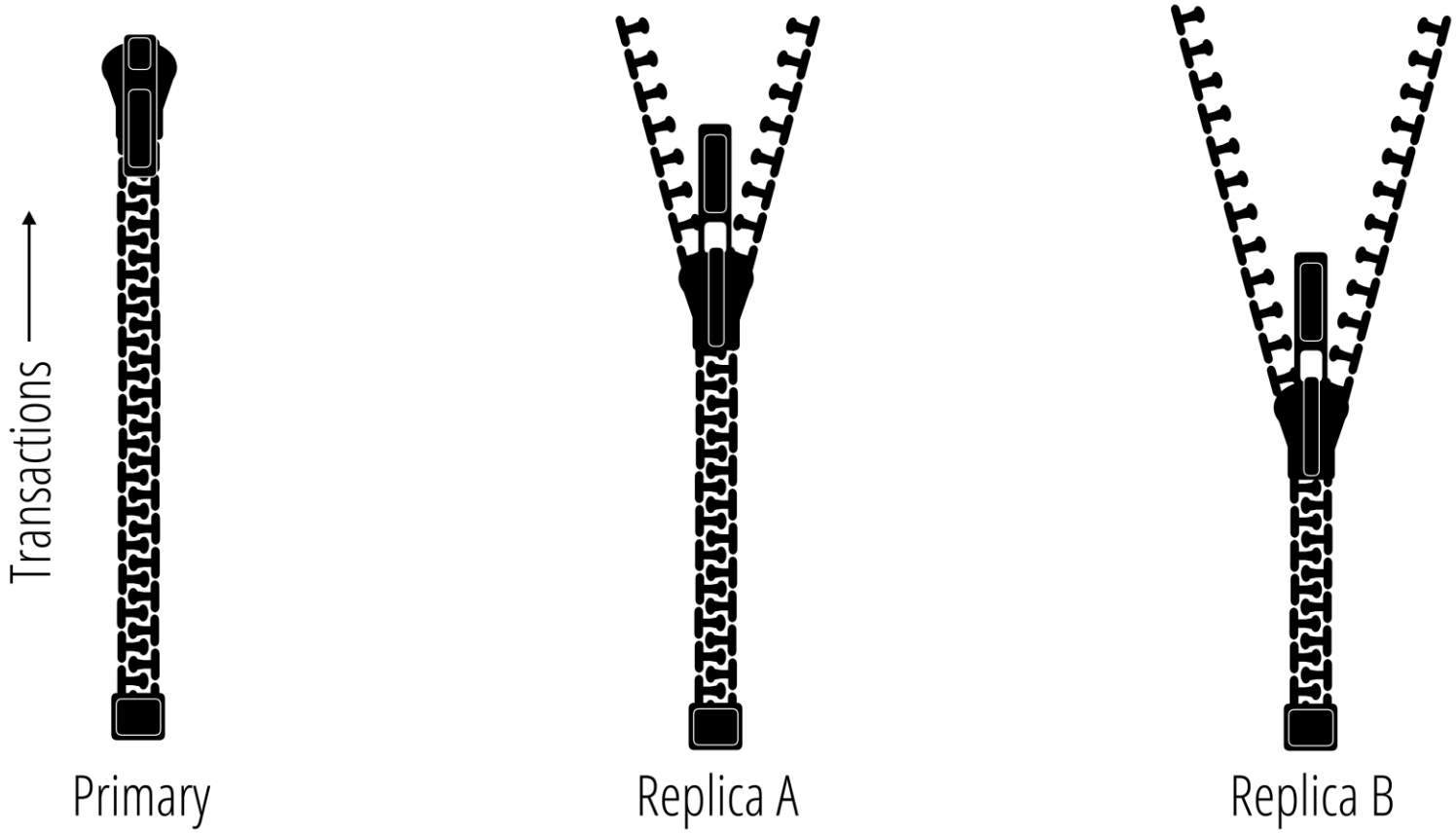
- **优点：**
  - 客户端不需要做额外的变更；
  - 有些云服务公司已经提供类似产品可以满足需求了，可以直接接入；
- **缺点：**
  - 除非是托管在云服务提供商那里，否则还需要额外的服务端来部署「路由」或者「负载均衡器」部分，这也就意味着需要保证这个部分的可靠性和可用性，需要额外的系统设计和运维工作；
  - 客户端请求多了一个路由代理的步骤，增加系统总体耗时；

开源工具

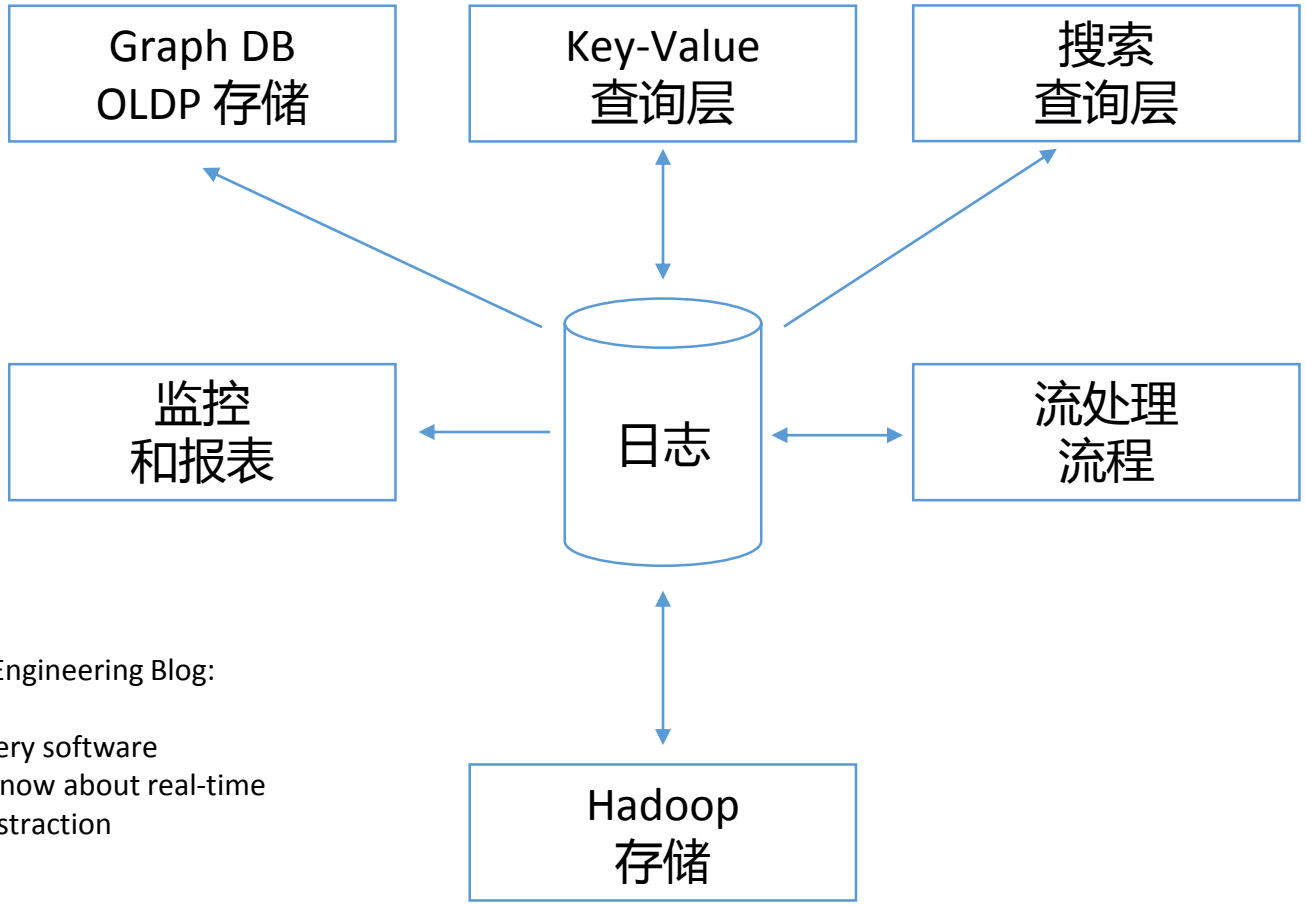
- etcd: 服务发现、全局分布式 key-value 存储；
- consul: 服务发现、全局分布式 key-value 存储；
- zookeeper: 服务发现、全局分布式 key-value 存储；

分布式系统设计

分布式数据库：基于日志的最终一致性



以日志为中心的基础架构



Source: LinkedIn Engineering Blog:

The Log: What every software engineer should know about real-time data's unifying abstraction

以 Log 为中心的分布式数据存储

- 通过顺序化节点的并发更新来处理数据一致性（最终和实时一致）；
- 为系统提供节点间的数据复制机制；
- 为数据的写提供「commit」语义；
- 为外部系统提供数据订阅服务；
- 为其它副本提供数据恢复的参考依据；
- 在节点之间提供数据读写平衡；

“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization’s communication structure.”

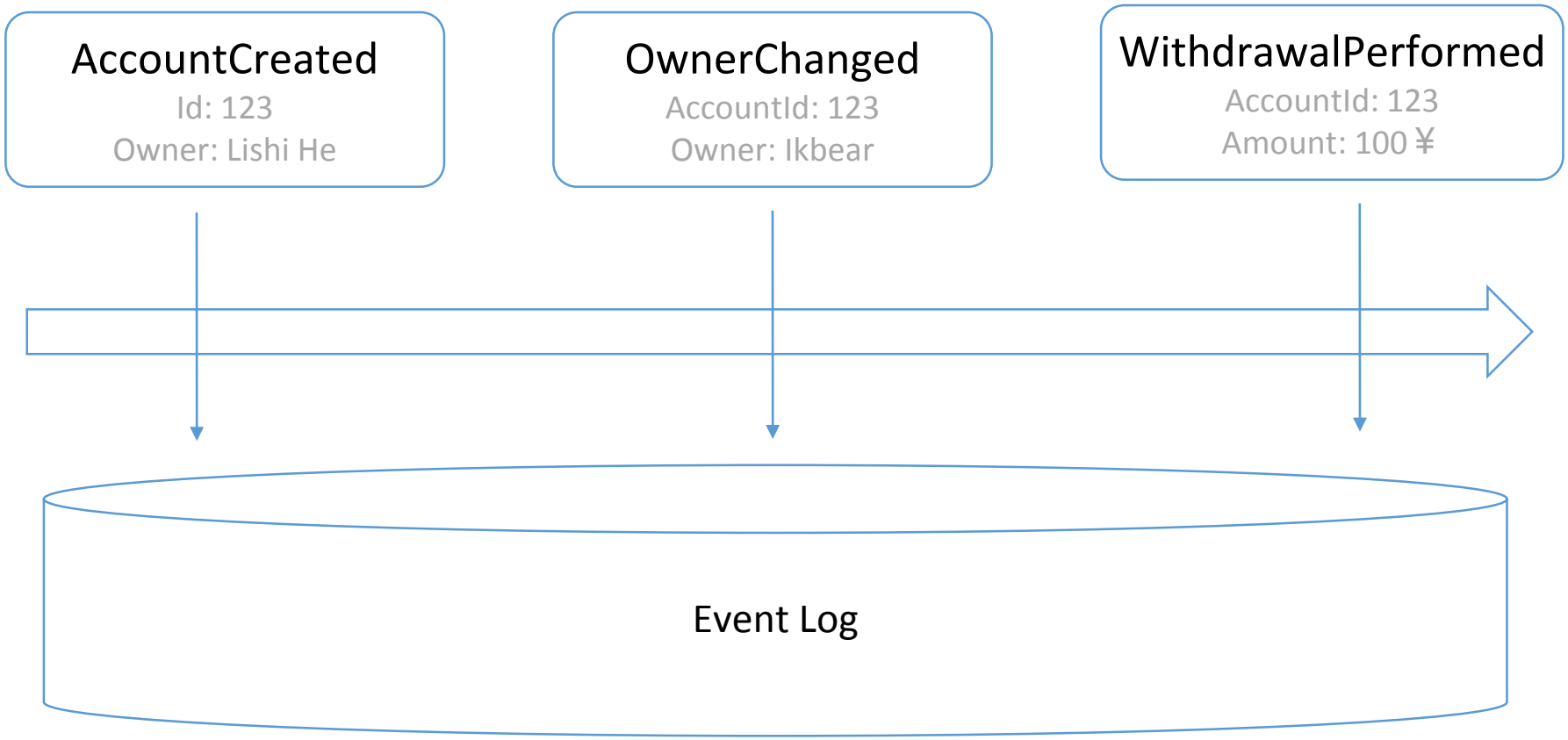
– Conway's Law

“一个组织的结构
是该组织设计系统时沟通结构的映射。”

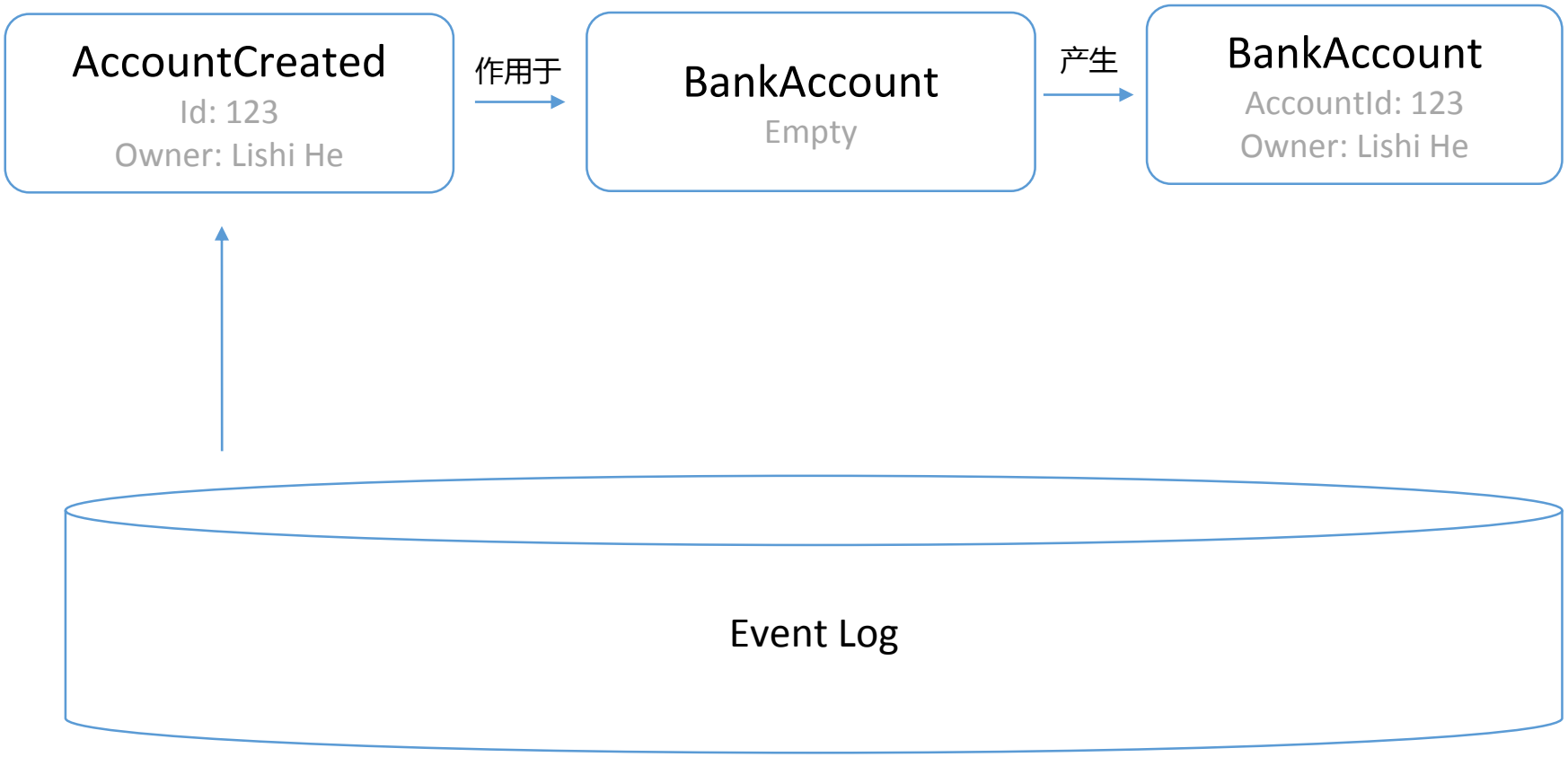
软件即沟通

- API as a Service: 基于 API 的沟通；
- 数据一致性：基于业务需求的沟通；

基于 Event Sourcing 的设计模式



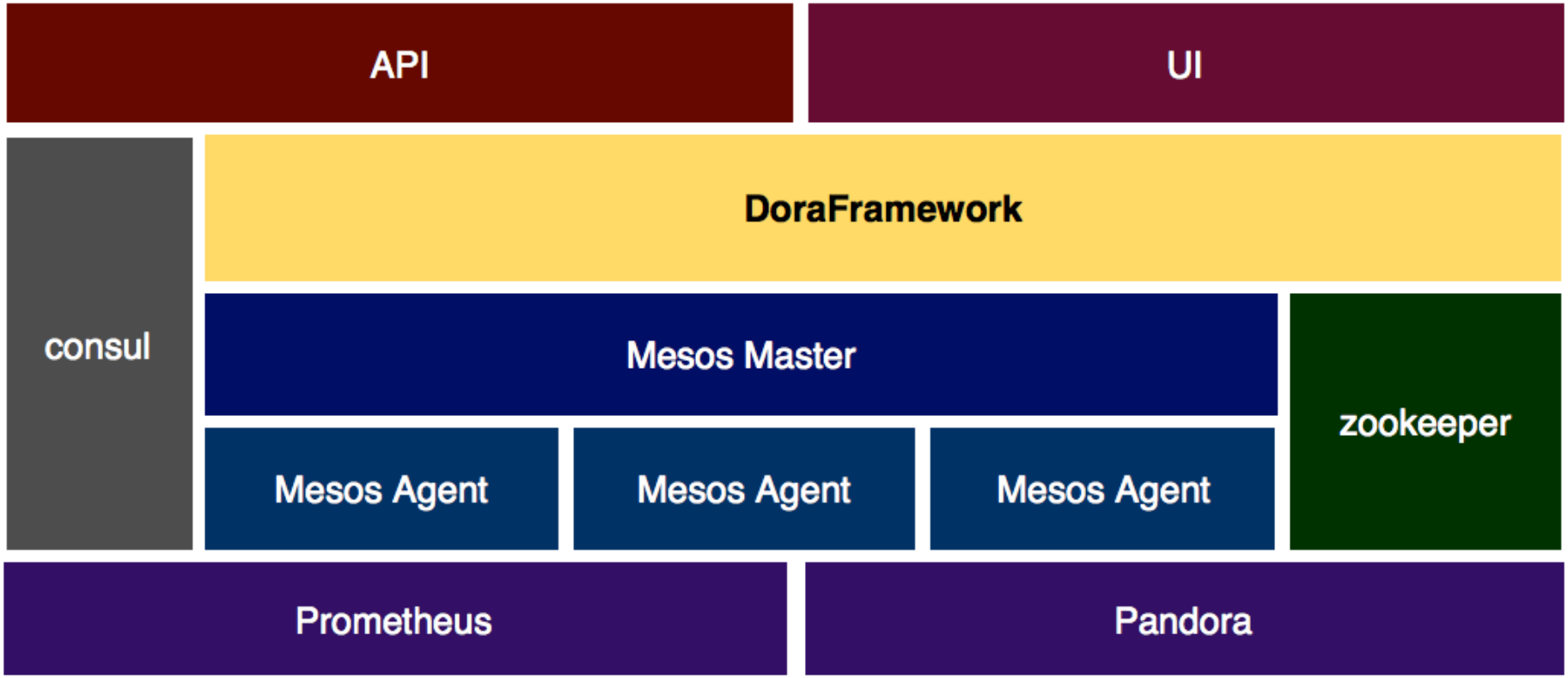
基于 Event log 恢复对象状态



数据一致性：基于 Event Sourcing 的设计模式

- 基于数据状态变更进行记录的分布式 Log；
- 可追踪和可调试的分布式微服务调用链路；
- 异步模式，在大多数场景下能够保证较好的性能；
- ORM 不能准确反映状态变更：保存变更事件，而非对象的结果值；
- 微服务之间可以更加彻底解耦；

基于容器云的大规模数据处理平台（逻辑架构）



Microservices

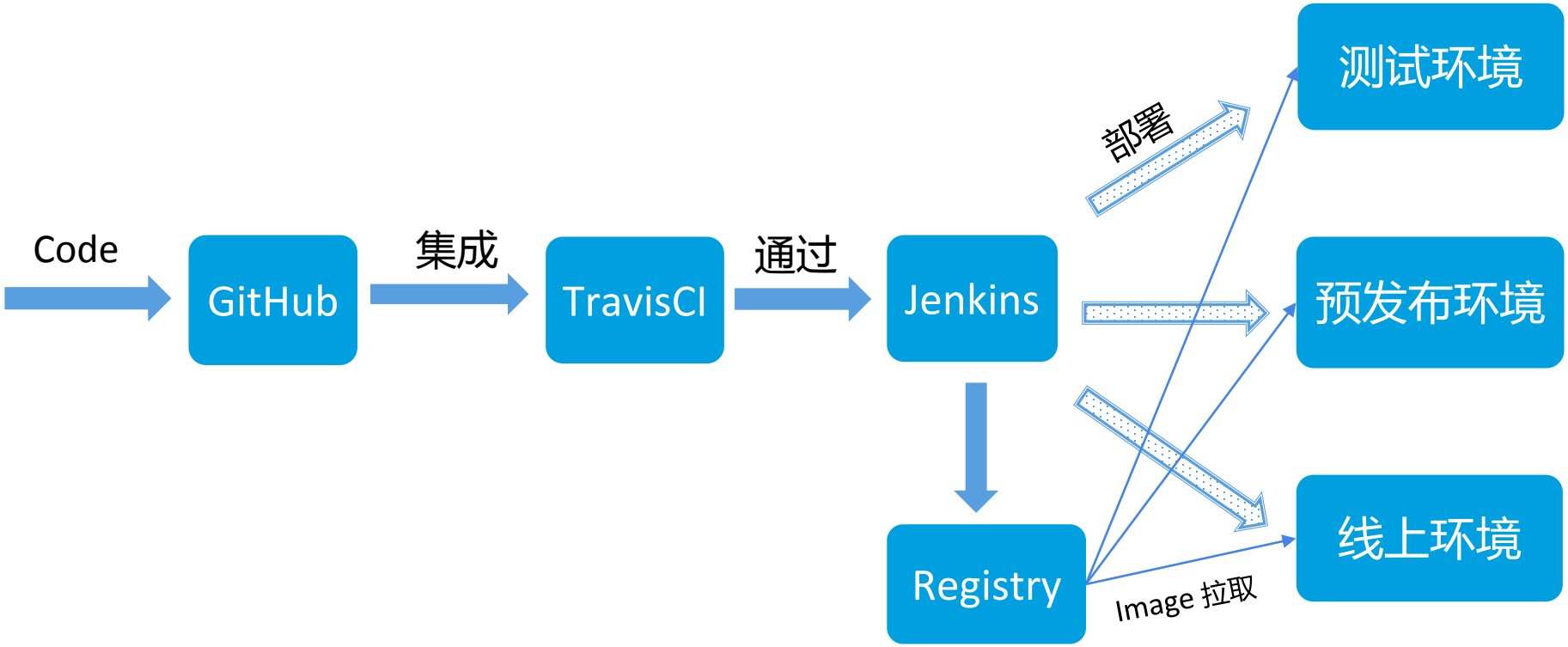
App: 七牛云富媒体处理平台应用

Infra: 面向大规模的服务架构

CI & CD & Ops: 构建自动化交付流程

微服务架构：重新定义大规模应用设计

构建基于容器云的持续交付流程

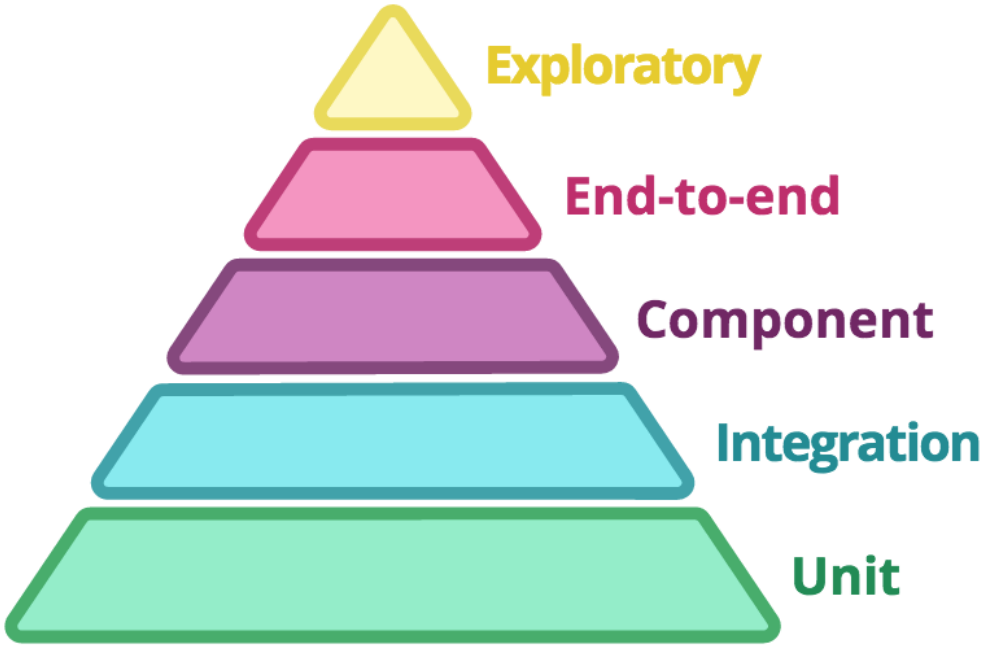


基于容器云的构建环境

- **1. 并行构建：自动任务调度；**
- **2. 混合构建：容器隔离构建环境**
  - 1) 多版本、多语言构建环境：Java / Node / Go；
  - 2) 构建环境瘦身；
- **3. 环境一致性：**
  - 1) 本地 Dockerfile 入库
  - 2) 构建环境：基于 Dockerfile 构建测试、线上镜像
  - 3) 线上：基于构建的镜像进行部署、升级

自动化测试

测试金字塔

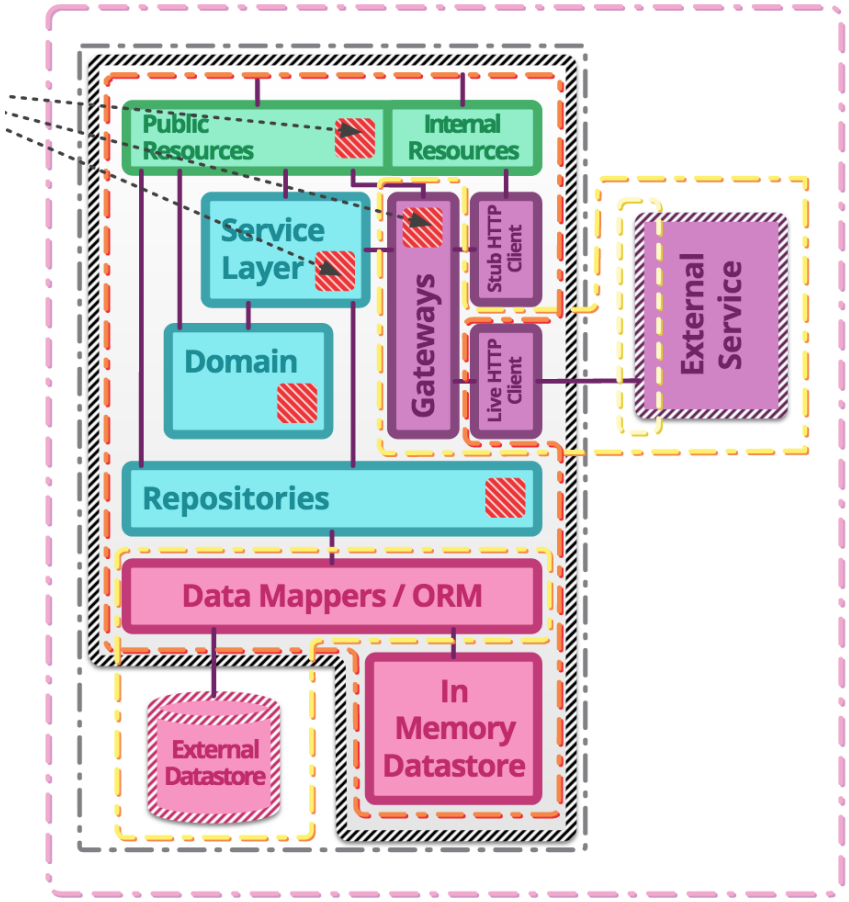


Source: Toby Clemson - Testing Strategies in a Microservice Architecture: http://martinfowler.com/articles/microservice-testing/

微服务架构测试

Unit tests : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.



Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

Source: Toby Clemson - Testing Strategies in a Microservice Architecture: http://martinfowler.com/articles/microservice-testing/

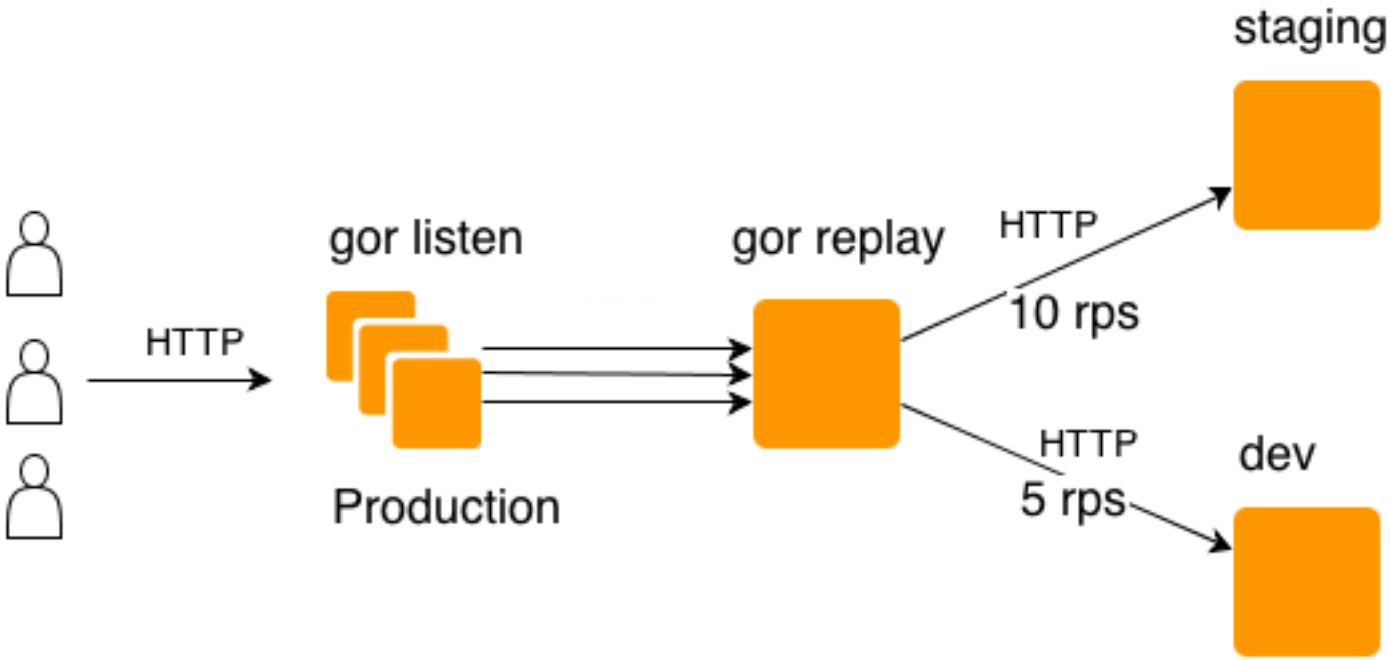
Netflix 猴子测试

Chaos Monkey: 随机杀死实例

Latency Monkey: 人为引入延迟

Chaos Gorilla: 模拟整个可用区突然断电

流量重现（测试）



https://goreplay.org/

Microservices

App: 七牛云富媒体处理平台应用

Infra: 面向大规模的服务架构

CI & CD & Ops: 构建自动化交付流程

微服务架构：重新定义大规模应用设计

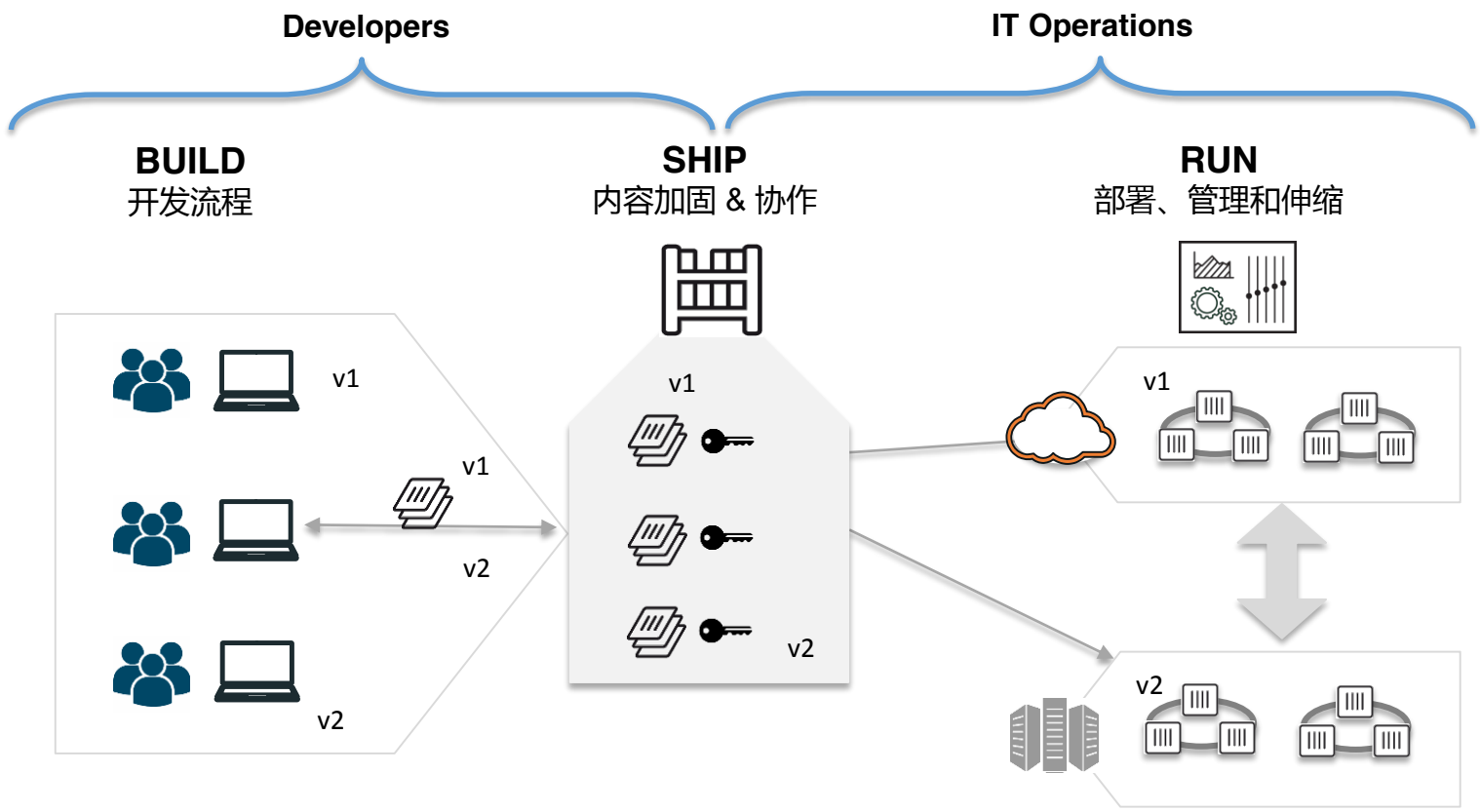
微服务架构：重新定义大规模应用设计（1）

- **组件化设计：**
  - 独立性： 每个组件都可以进行独立的「Build => Ship => Run」流程；
  - 耦合性：组件之间的关系通过 API 进行定义，如果不能抽象出这个交互关系，则不能让每个组件进行独立的演化；
- **不可变基础设施：基于镜像的部署**
  - 一处构建，到处运行（Docker）；
  - 自动化流程管理构建和安全升级过程 ；

微服务架构：重新定义大规模应用设计（2）

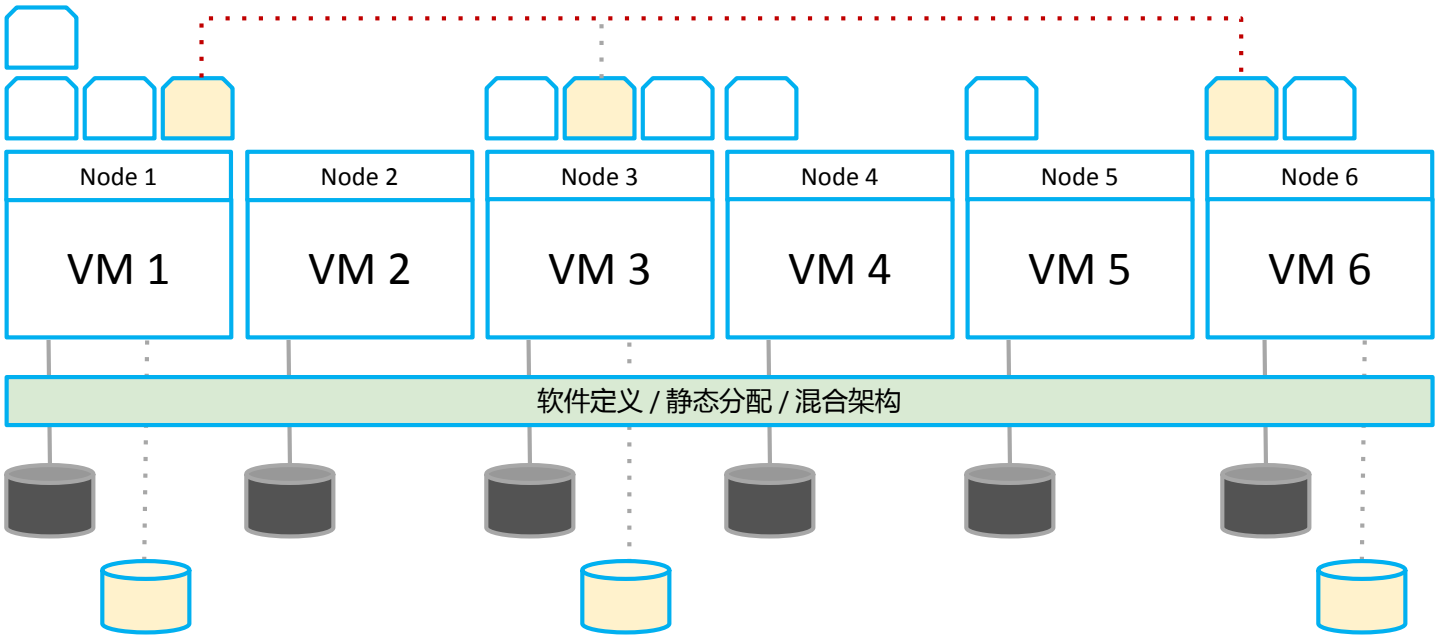
- **解耦研发和运维工作：**
  - 让两个团队共享技术栈、基础设施：基于容器镜像的研发、测试和上线流程；
  - 将一切模板化：配置信息和流程模板化，变更流程模板化；
  - 快速构建新部署、测试环境；
  - 基于容器云平台的基础架构，快速应对应用的扩容、缩容；

版本化的构建、持续灰度发布



微服务架构：重新定义大规模应用设计（3）

- **简化运维复杂度：**



容器云

节点（物理 or VM）

网络

本地存储

持久化网络存储

THANKS

ArchSummit
全 球 架 构 师 峰 会 2016

[北京站]

主办方 Geekbang. 极客邦科技 InfoQ