API Gateway 演进

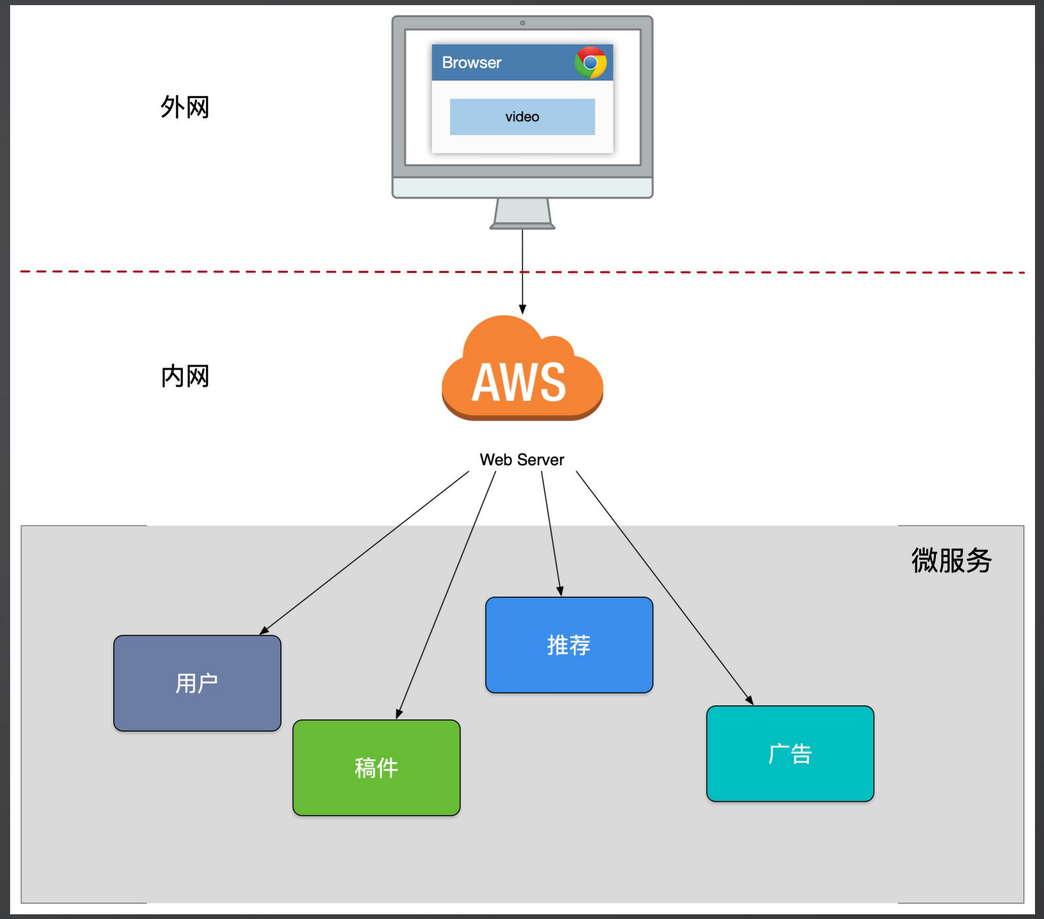
毛剑/bilibili 基础架构部

目录

API Gateway 演进

从 PC 时代向移动时代演进，移动端大量了沿用了过去的接口模型。没有明细的区分内外部服务，因为缺少统一出口我们逐渐发现了一些问题：

- *客户端到微服务直接通信，强耦合；*
- *需要多次请求，客户端聚合数据，工作量巨大，延迟高；*
- *协议不利于统一，各个部门间有差异，需要端来兼容；*
- *面向端的 API 适配，耦合到了内部服务；*
- *多终端兼容逻辑复杂，每个服务都需要处理；*
- *统一逻辑无法收敛，比如安全认证、限流；*

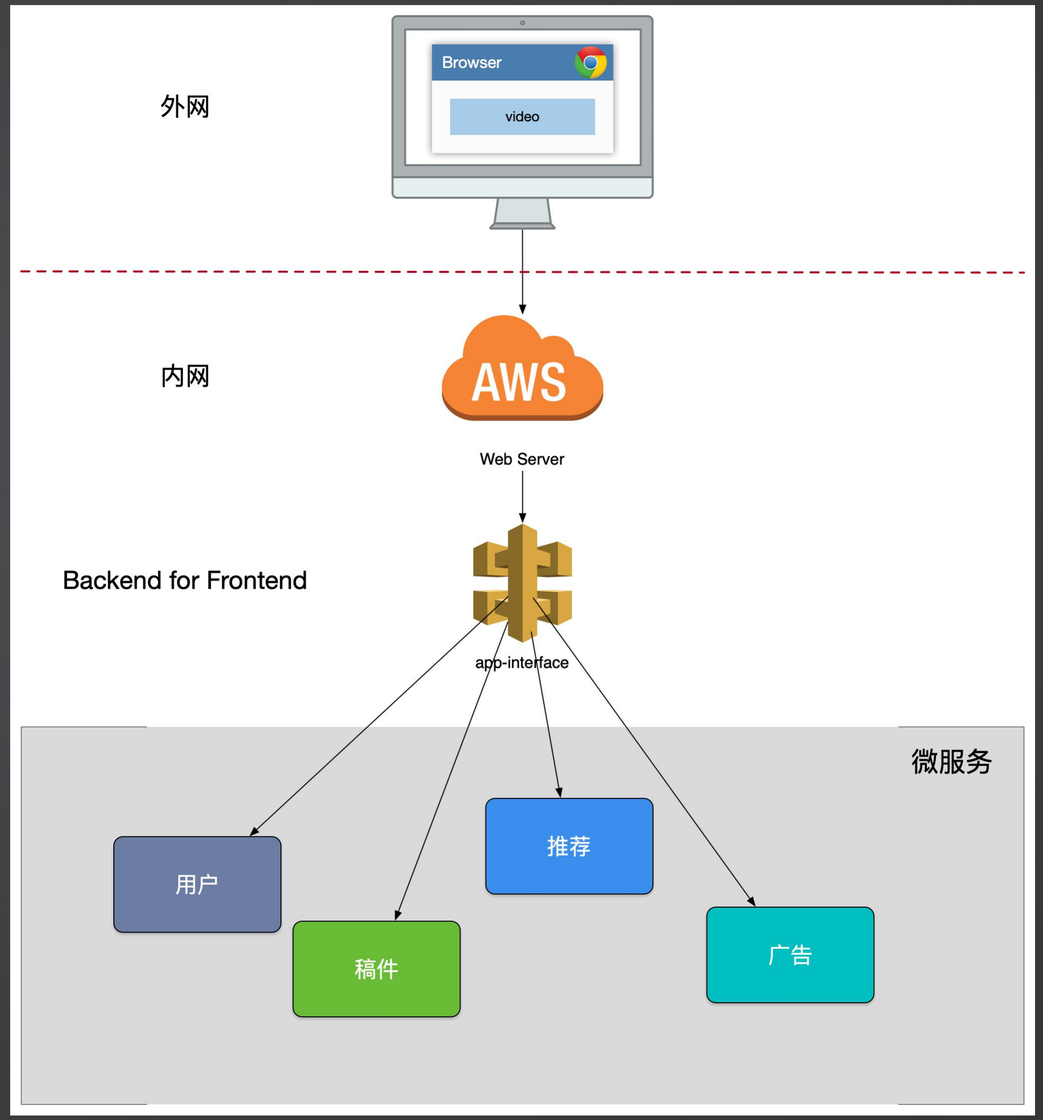


API Gateway 演进

新增了 app-interface 用于统一的协议出口，在服务内进行大量的数据组装，按照业务场景设计粗粒度的 API：

- ***轻量交互：协议精简、聚合。***
- ***差异服务：数据裁剪以及聚合、针对终端定制化 API。***
- ***动态升级：原有系统兼容升级，更新服务而非协议。***
- ***沟通效率提升，协作模式演进为移动业务+网关小组。***

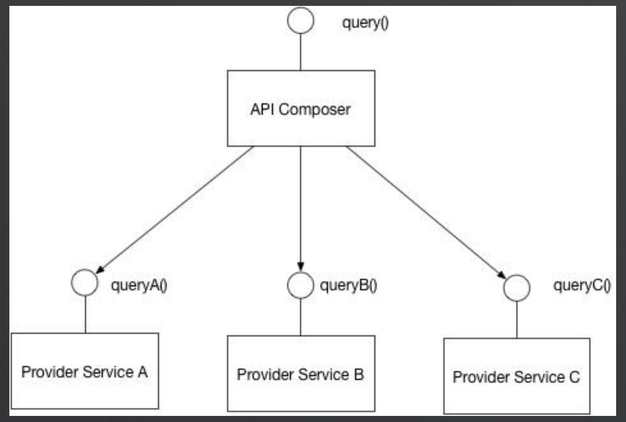
BFF *可以认为是一种适配服务，将后端的微服务进行适配（主要包括聚合裁剪和格式适配等逻辑），向无线端设备暴露友好和统一的* API，*方便无线设备接入访问后端服务。*

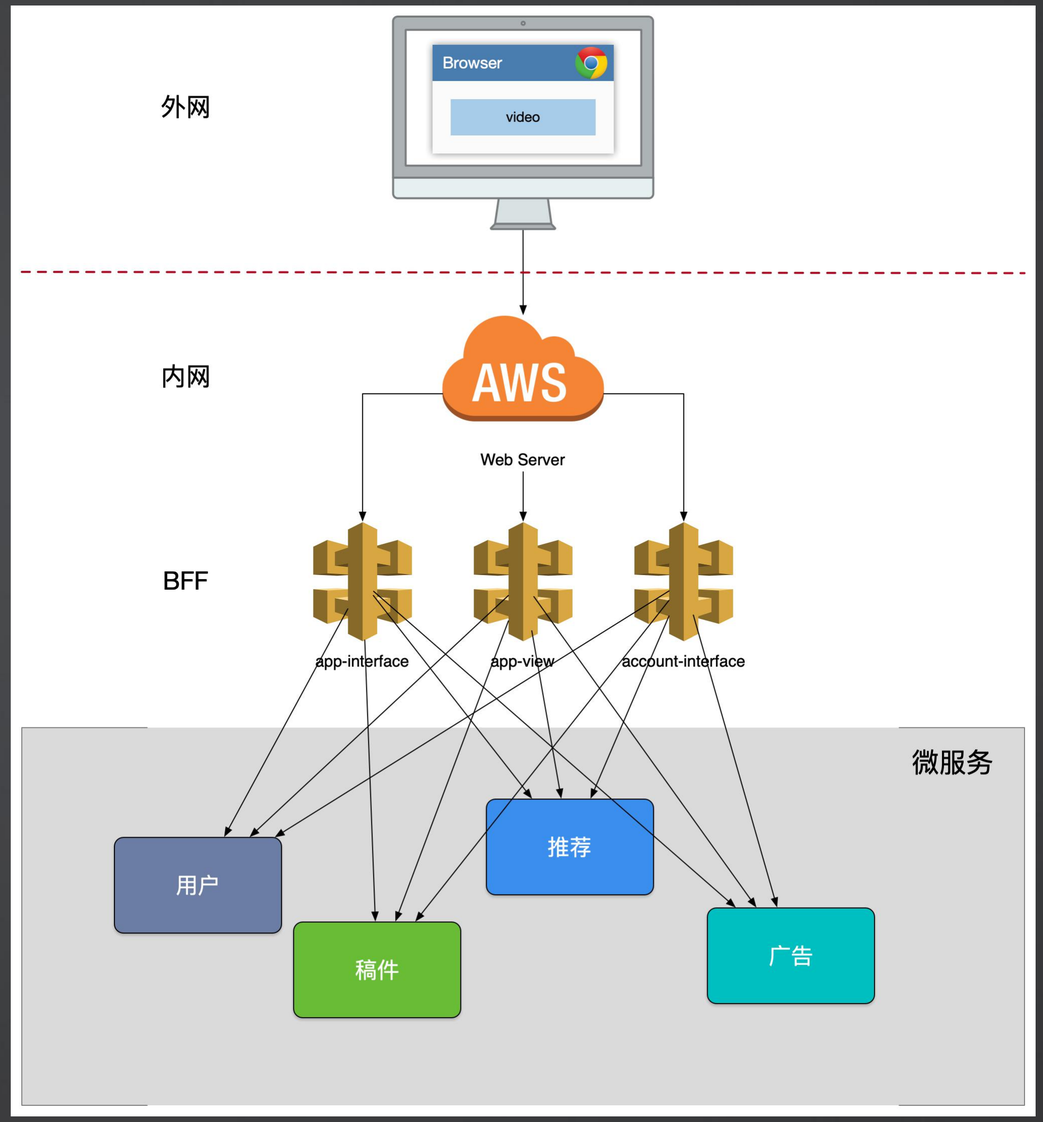


API Gateway 演进

最致命的一个问题是整个 app-interface 属于 single point of failure，严重代码缺陷或者流量洪峰可能引发集群宕机。

- *单个模块也会导致后续业务集成复杂度高，根据康威法则，单 BFF 和多团队之间就出现不匹配问题，团队之间沟通协调成本高，交付效率低下。*
- *很多跨横切面逻辑，比如安全认证，日志监控，限流熔断等。随着时间的推移，代码变得越来越复杂，技术债越堆越多。*





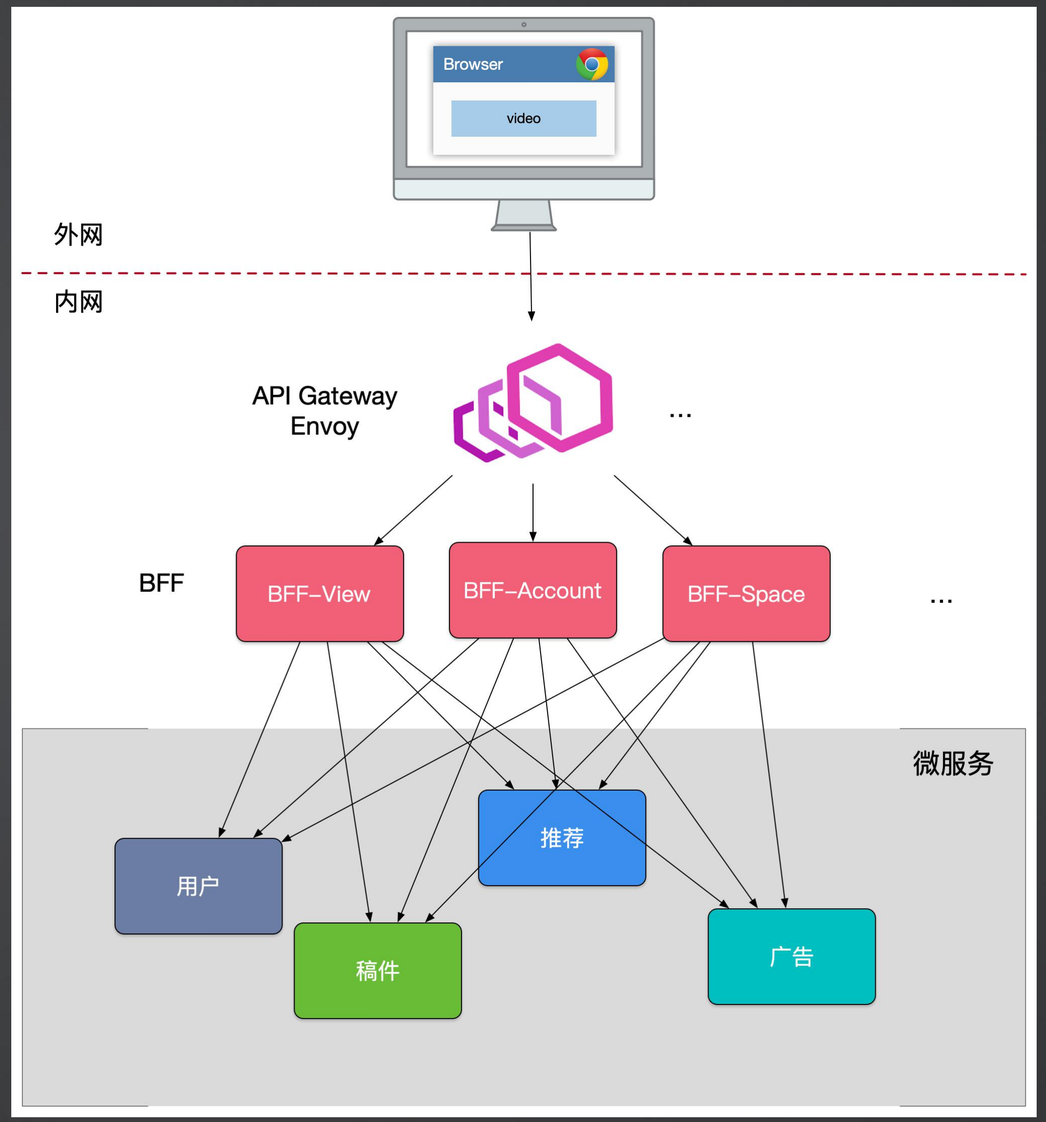
API Gateway 演进

跨横切面（Cross-Cutting Concerns）的功能，需要协调更新框架升级发版（路由、认证、限流、安全），因此全部上浮，引入了 **API Gateway**，把业务集成度高的 BFF 层和通用功能服务层 API Gateway 进行了分层处理。

在新的架构中，网关承担了重要的角色，它是解耦拆分和后续升级迁移的利器。在网关的配合下，单块 BFF 实现了解耦拆分，各业务线团队可以独立开发和交付各自的微服务，研发效率大大提升。另外，把跨横切面逻辑从 BFF 剥离到网关上去以后，BFF 的开发人员可以更加专注业务逻辑交付，实现了架构上的关注分离（Separation of Concerns）。

我们业务流量实际为：

移动端 -> API Gateway -> BFF -> Microservices，*在* FE Web*业务中，* BFF *可以是* Node.js *来做服务端渲染（*SSR，Server-Side Rendering*），注意这里忽略了上游的* CDN、4/7*层负载均衡（*SLB*）。*



API Gateway 演进

使用 Go 语言全新开发了 https://github.com/go-kratos/gateway，核心考虑可控性、稳定性。控制面目前自定义实现，后续会对齐社区 XDS 协议，可以使用 Istio 或者 kratos gateway UI 进行治理。

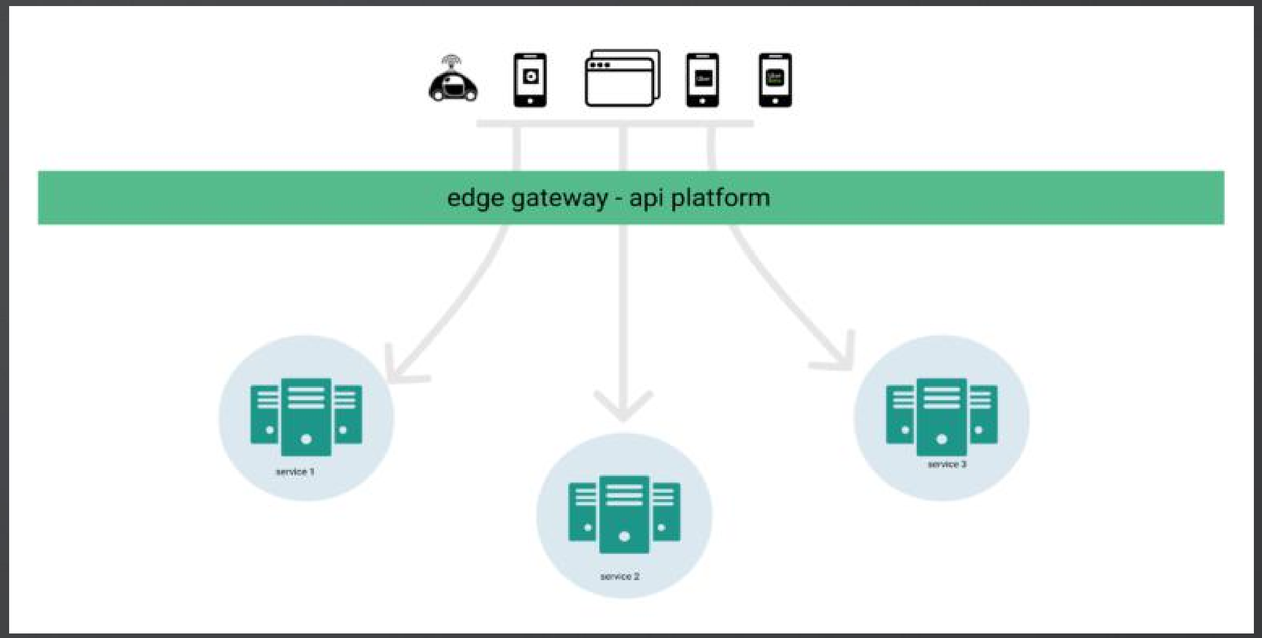
BFF *目前使用* DSL *编排* View *逻辑，当复杂场景无法描述时候可能想一个高级点的语言* Typescript，*更进一步我们想使用* Control Flow Framework + Serverless/FaaS *易于交付的形式解决* BFF *的问题。*

目录

API Gateway 可用性设计

API Gateway 从架构层面来说仍然是单点，类似之前我们 SLB 的故障。可用性最大的一环是涉及核心组件的变更流程管理，其他的可用性这里基本类似微服务：

- *隔离*
- *超时控制*
- *重试*
- *限流*
- *过载保护*
- *熔断*
- *降级*
- *负载均衡*

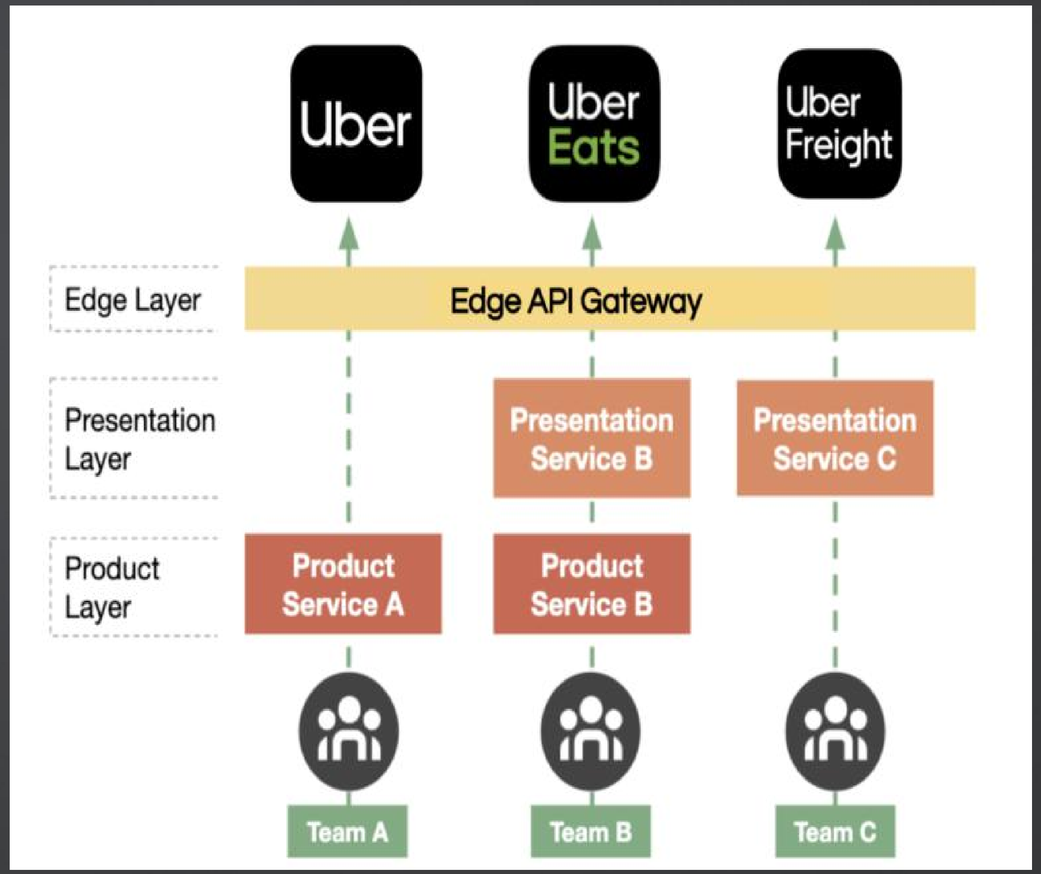


目录

API Gateway 边缘计算

Separation of concerns，引入 API Gateway 其本质是关注分离，引导开发人员更专注本层的产品开发工作：

- API Gateway *层：解耦、协议转换、横切面功能（监控、数据验证、安全审计、CORS、缓存、限流、过载保护等），但不包括* BFF *的职责；*
- *展现层：专门为产品前端提供后端的微服务。管理自己的视图和编排服务，以满足消费应用程序所需的*API*。这些服务中的代码旨在生成视图并聚合来自许多下游服务的数据，并且有单独的*API*来修改针对特定消费者的响应。*
- *产品层：平台型微服务，具备特定的产品功能，提供可复用的* API*，可以直接对外提供服务，也可以被其他系统所使用；*
- *领域层：微服务的叶节点，为产品团队提供单一的功能；*



References

Uber 1: https://eng.uber.com/architecture-api-gateway/

Uber 2: https://eng.uber.com/scaling-api-gateway/

Uber 3: https://eng.uber.com/gatewayuberapi/

Zuul: https://xie.infoq.cn/article/2636d77e2fe433e8144680eac

阿里 1 : https://mp.weixin.qq.com/s/RNpGF4tP2mj7aS3UfMqzAA

阿里 2: https://mp.weixin.qq.com/s/uoJIIIMX3Sj7I-VepwSC9A

爱奇艺: https://mp.weixin.qq.com/s/joaYcdmeelGZmpMcEo-mpw

京东: https://mp.weixin.qq.com/s/WjZqLCCxc8oCP42qC6_o1Q

淘宝：https://tool.lu/deck/gp/detail?slide=12

美团：https://tech.meituan.com/2021/05/20/shepherd-api-gateway.html

eolinker：https://help.eolinker.com/private_cloud/api_gateway/html/quick/terms.html

访问控制LBAC：https://www.ibm.com/docs/zh/db2/11.5?topic=security-label-based-access-control-lbac

THANKS