云原生架构下的微服务选型和演进

原创

阿里巴巴云原生 于 2022-04-21 22:51:36 发布 240 收藏
文章标签： 云原生

本文链接：https://blog.csdn.net/alisystemsoftware/article/details/124333363

作者：彦林

本文整理自阿里云智能高级技术专家彦林的线上直播分享《云原生微服务最佳实践》。视频回放地址：
https://yqh.aliyun.com/live/detail/28454

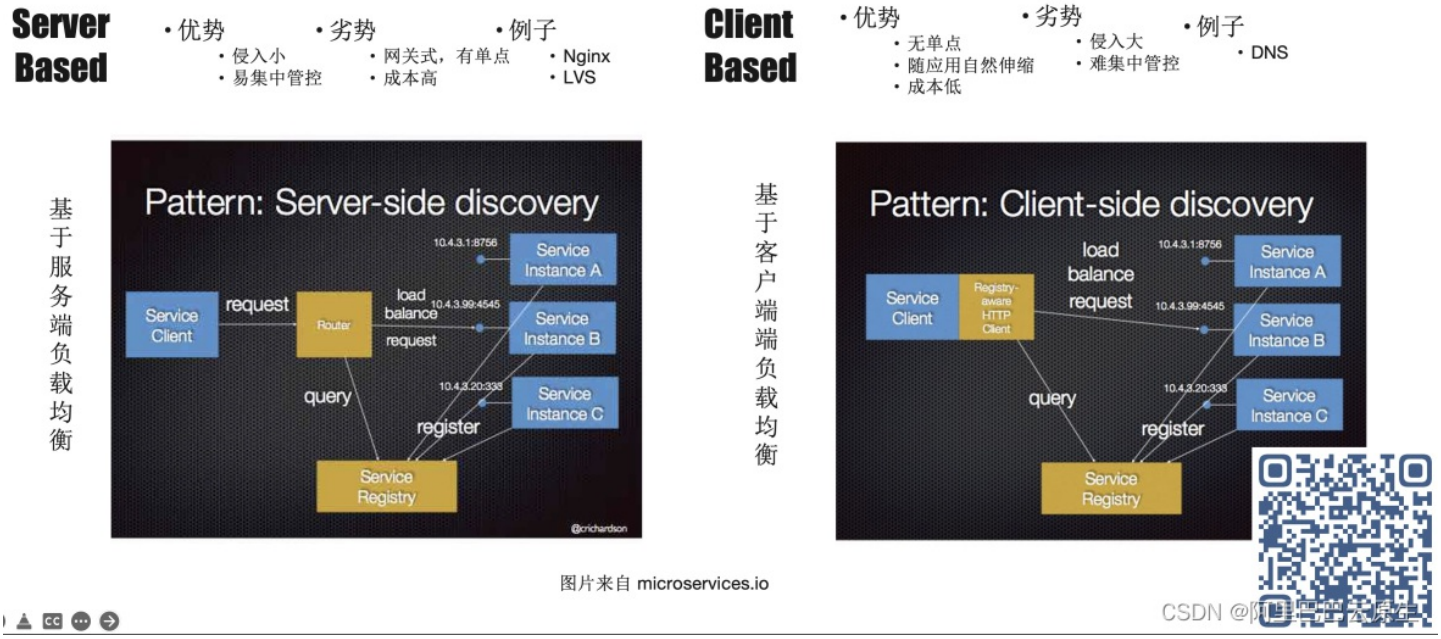
随着云原生的演进，微服务作为主流应用架构被广泛使用，其落地的难题逐步从如何建好延伸到如何用好。今天跟各位小伙伴分享一下我在微服务领域 10 余年的实践经验，如何以更高效的姿势把微服务这件事做扎实。

阿里微服务发展历程

微服务 1.0（1w 实例/微服务拆分/同城容灾）

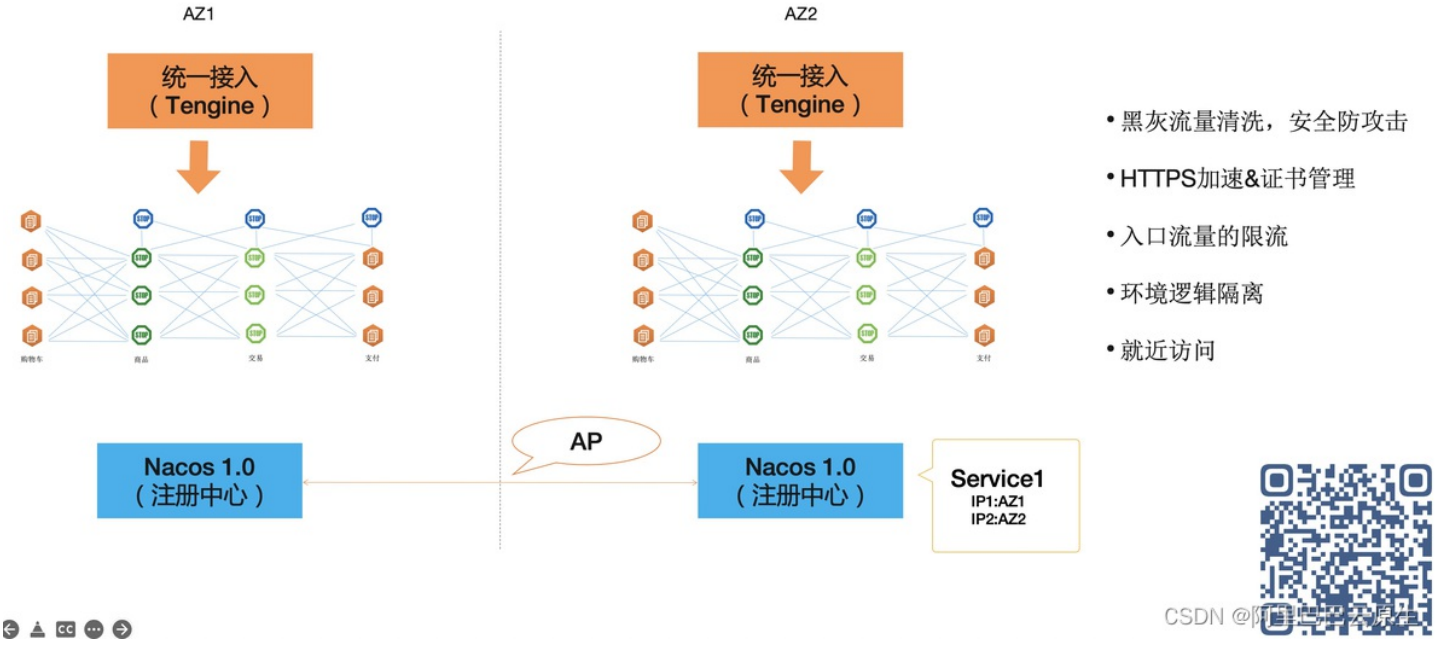
2008 年随着阿里业务规模不断增大，单体胖应用+硬负载的架构逐渐暴露性能瓶颈；随着研发人员逐步增多，协调效率也逐步下降，不能满足日益复杂的业务挑战，因此急需技术升级解决这些问题。

在当时 SOA 架构非常流行，也就成为我们技术演进的主要方向，当时有两种解决方案，一个是 Server Based 的解决方案，这种模式侵入小、方便集中管控，但是这种中心化方案会带来成本高、稳定性风险高、扩展性差；一个是 Client Based 的解决方案，这种模式去中心化，扩展性强，成本低，但是会带来一定侵入性，比较难以管理；当然很多人会问为什么不直接用 DNS 呢？主要是 DNS 不能满足 IDC 内部服务发现实时性，服务列表更新不能及时通知下有业务会导致业务流量损失。

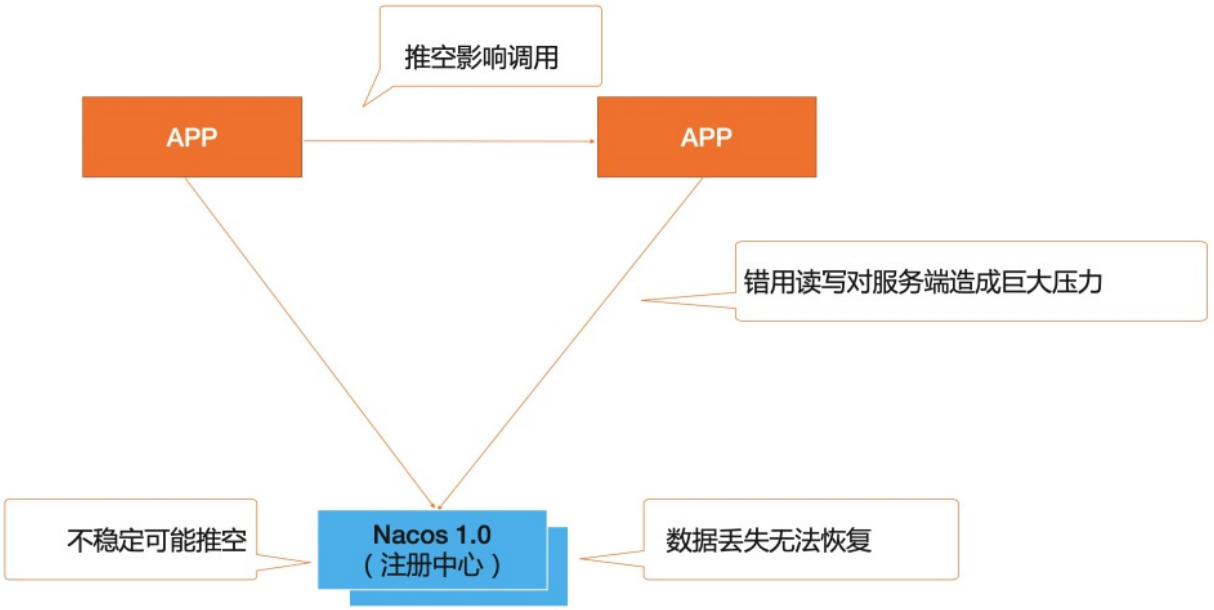


图片来自 microservices.io

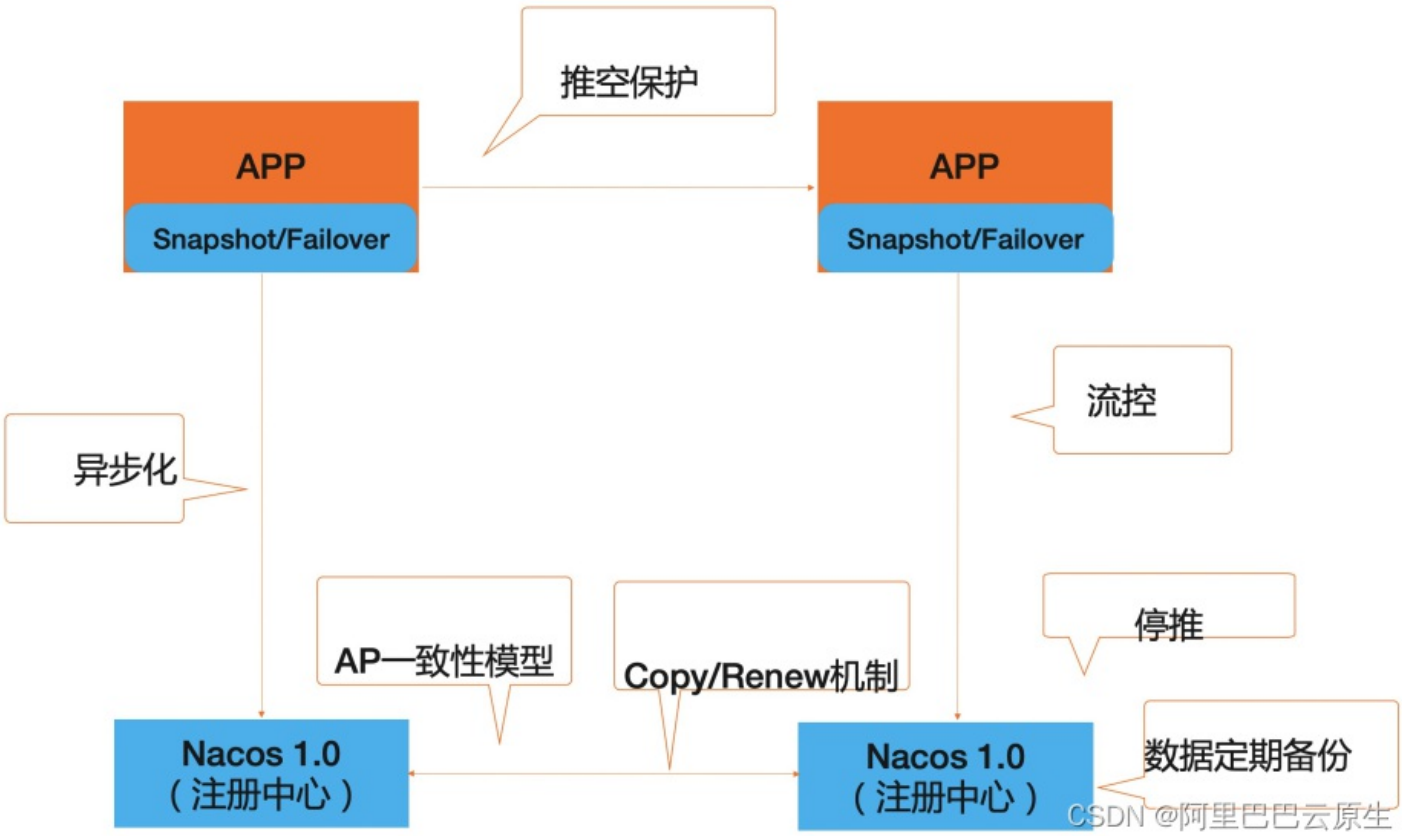
在评估两种方案利弊之后，我们在网关这种需要集中管理安全和简单路由场景采用了 Server Based 的方案，基于 Nginx 演进出了阿里 Tengine 网关技术体系，从入口处解决安全、高可用、简单路由能力；在 IDC 内部采用了 Client Base 模式，孵化出 HSF/Dubbo+Nacos 技术体系，支撑了业务微服务拆分。



随着第一代微服务架构落地，由于引入注册中心带来了稳定性风险，注册中心挂会导致调用链路全部中断；业务集中发布的时候注册中心压力会比较大。



针对可用性问题我们提供了推空保护能力，即使注册中心挂也不会影响业务正常运行；为了提供更好性能我们提供了全异步架构；为了支持同城容灾我们提供了 AP 一致性协议，具体协议可以参考《Nacos 架构与原理》电子书。



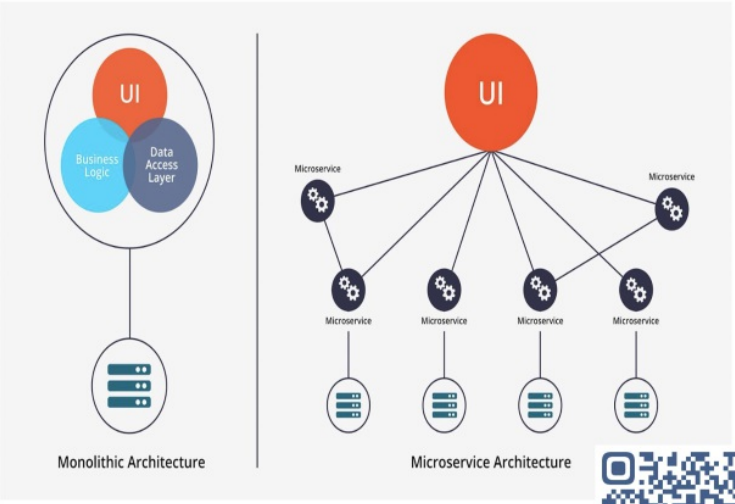
随着阿里微服务 1.0 架构落地，帮助业务完成微服务拆分，解决了扩展性和协同效率问题，同时支撑了阿里同城容灾能力。对于正在做微服务的小伙伴可能问阿里如何做微服务架构演进的：

前后端分离是第一步，因为前端变化多，变化快，后端相对变化小，演进慢，因此需要解耦发展，让前端更快的适应市场变化，以便在竞争中保持先机；

后端无状态改造是第二步，把内存状态外置到 Redis，把持久化状态外置到 Mysql，这样业务就可以随意进行切分；

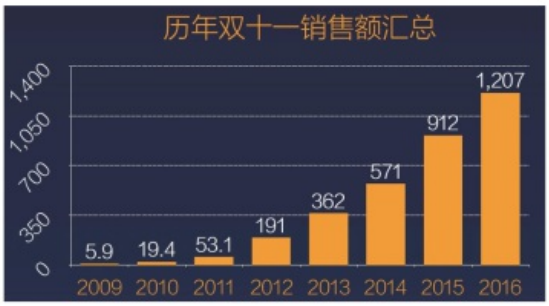
第三步是模块化拆分，这块是最考验架构师的，因为拆分一个是按照业务属性拆分，一个是按照应用复杂度进行拆分，这个是一个相对动态过程，建议拆分模块后 2-3 人负责一个模块，拆到太细会有比较高的运维成本，拆的太粗又会带来研发协同问题，阿里内部也经历过合久必分，分久必合的几波震荡，最终走到相对稳态。这里值得一提就是 HSF/Dubbo 的一个优势，因为早期采用 SOA 架构思想设计，一个接口就是一个服务，这样其实非常方便服务的拆分和合并，当然同时带来一个问题是对注册中心性能压力比较大，这是一个架构选择和平衡问题。

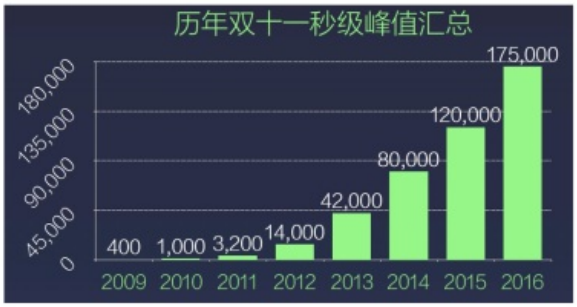
- 微服务1.0
  - 支持1w级实例规模
  - 具备99.9%高可用能力
  - 具备多级容灾体系
- 微服务拆分
  - 解决超过10人协同
  - 解决10+子系统复杂度
  - 前后端分离
  - 无状态改造（Redis/Mysql）
  - 模块化拆分（2-3个人1个模块）
- 同城容灾
  - 支撑同城双活能力
  - 支持就近访问能力
  - 具备隔离环境能力



微服务 2.0（10w 实例/业务中台/异地多活）

微服务 1.0 架构帮助阿里极大缓解性能和效率问题，但是由于阿里双十一的成功，技术上面临一个洪峰的技术挑战，我们必须在用户体验、资源成本、高可用之间做一个平衡。这个阶段我们最大的挑战是扩展性和稳定性，扩展性是要支撑业务 10w+实例扩容，但是单地资源有限，双十一商家投入的资金越来越大，导致我们双十一当天也不能出严重问题，不然损失非常大，因此对业务稳定性提出非常高的要求。

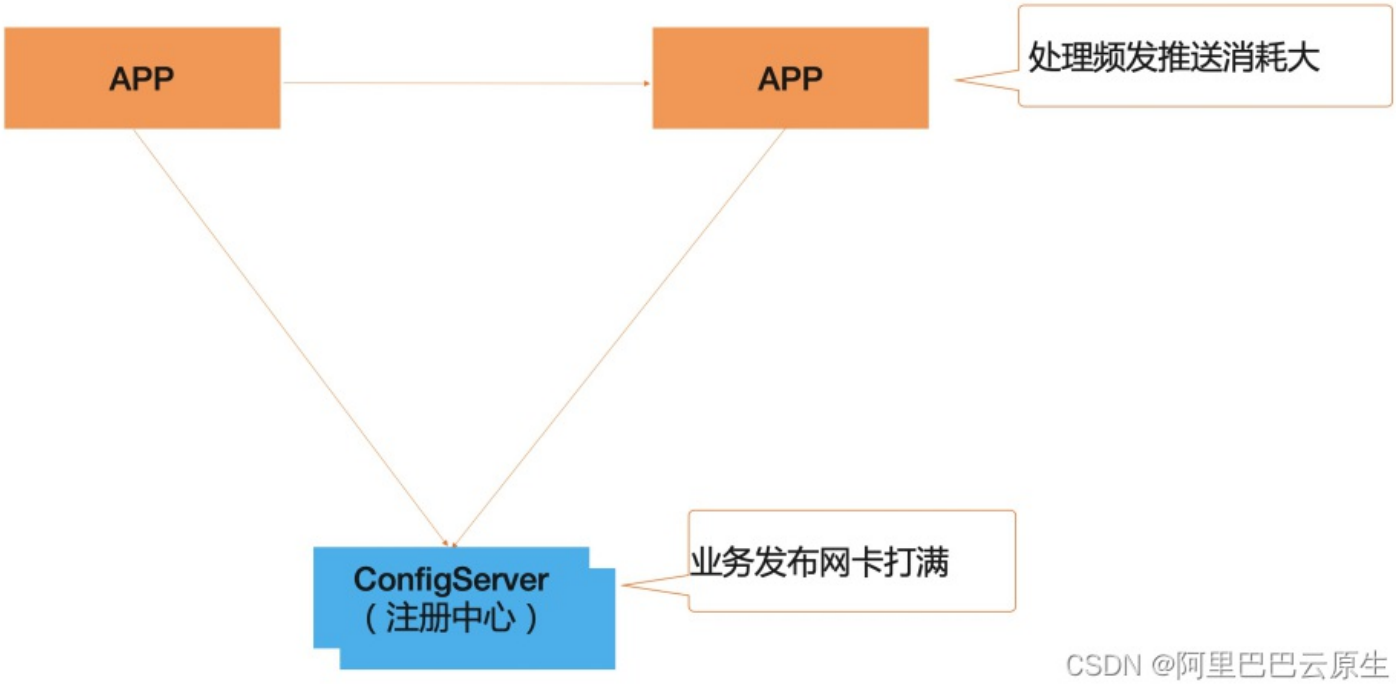




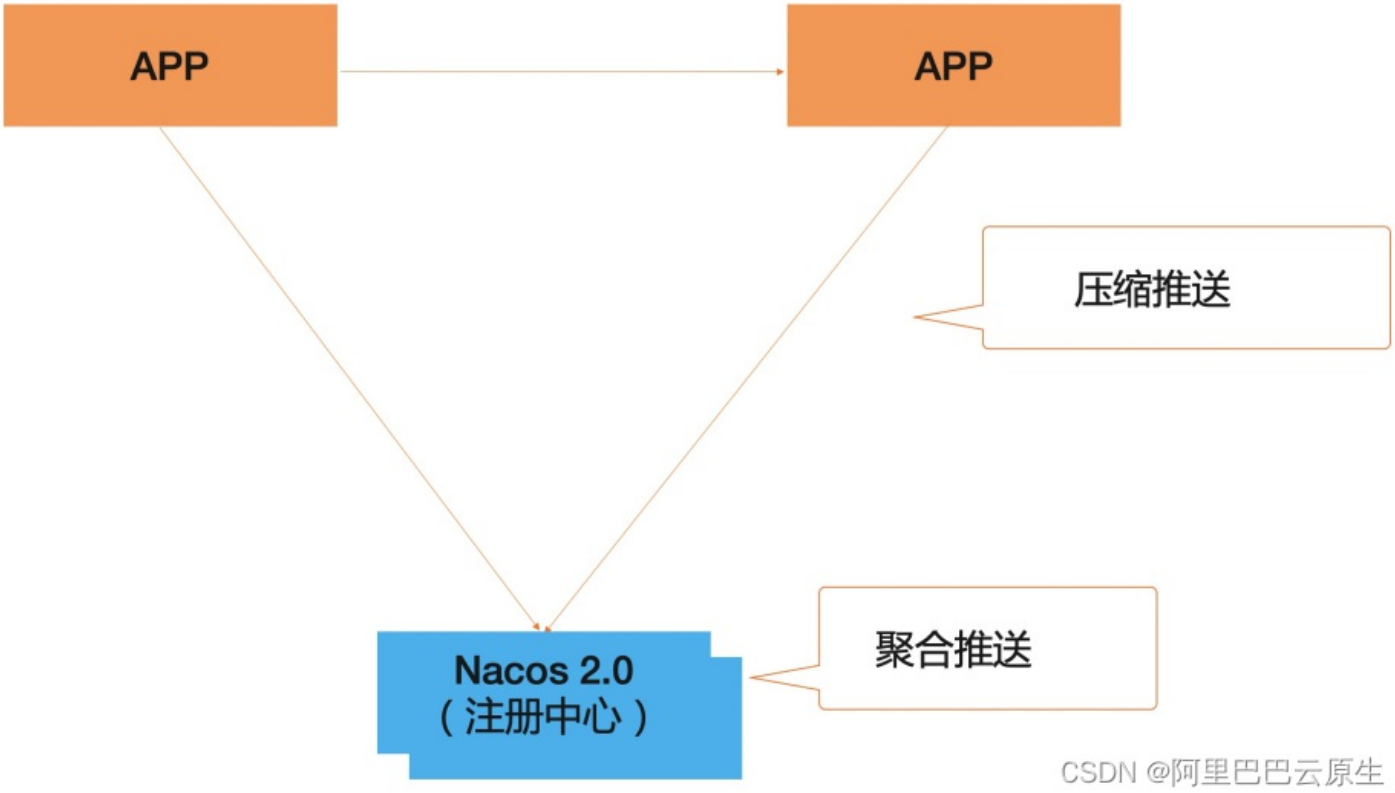
·资源：电商、阿里云、大数据等业务高速发展，单地资源容量受限（10w级）
·扩展：业务多元化对异地部署需求
·容灾：天灾、人祸都会影响业务的可用性

因此阿里演进到微服务 2.0 支撑了异地多活的高可用体系，让阿里业务可以按照 IDC 级别水平扩展，新的机房，新的技术体系都可以在单元中进行验证，也加速了阿里技术体系演进速度。

在此期间 Nacos Server 间水平通知压力巨大，业务发布窗口容易把网卡打满，频繁推送会消耗业务大量内存和 CPU，进而影响业务的稳定性。

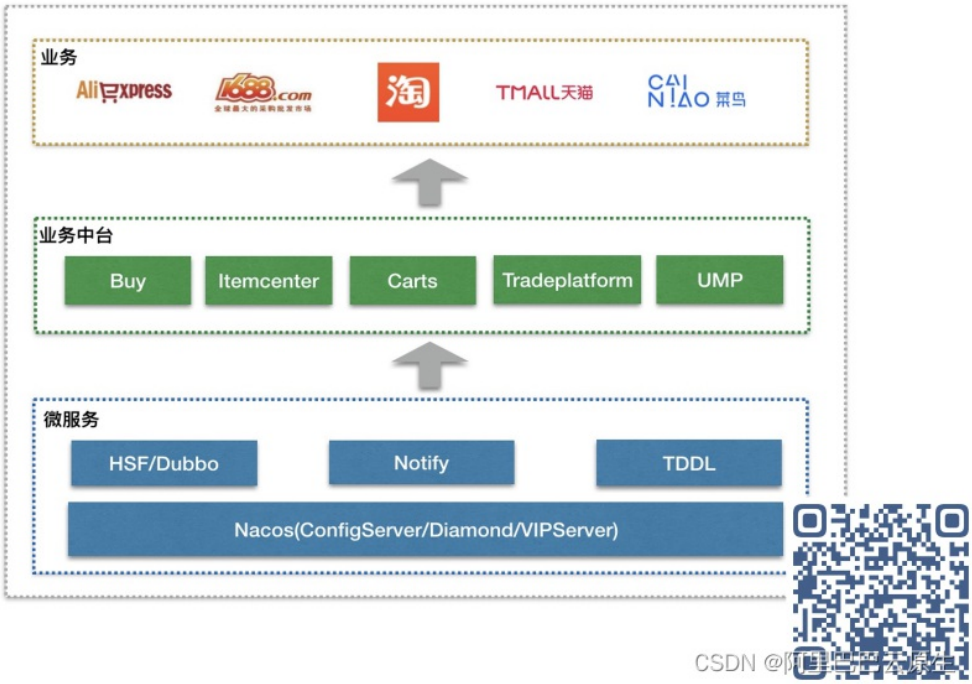


针对上述问题，我们在 Nacos Server 间做了聚合推送，将一定时间窗的变更合并聚合推送，推送过程中做了压缩推送，从而解决了上述问题。



在微服务解决扩展性和高可用的同时，业务系统变多，重复建设，业务孤岛也越来越多，协同效率也越来越低，因此阿里业务在这个时候推出了业务中台能力，将扁平的微服务抽象分层，将基础服务抽象为中台服务解决上述问题，业务分层后支撑了阿里业务高速增长，也加速了技术架构统一。

- 微服务2.0
  - 支持10w级实例规模
  - 具备聚合推送能力
  - 具备压缩推送能力
- 支撑业务中台
  - 解决超过100+人协同问题
  - 解决100+子系统复杂度
  - 抽象基础中台服务
- 全局高可用
  - 支撑异地双活能力

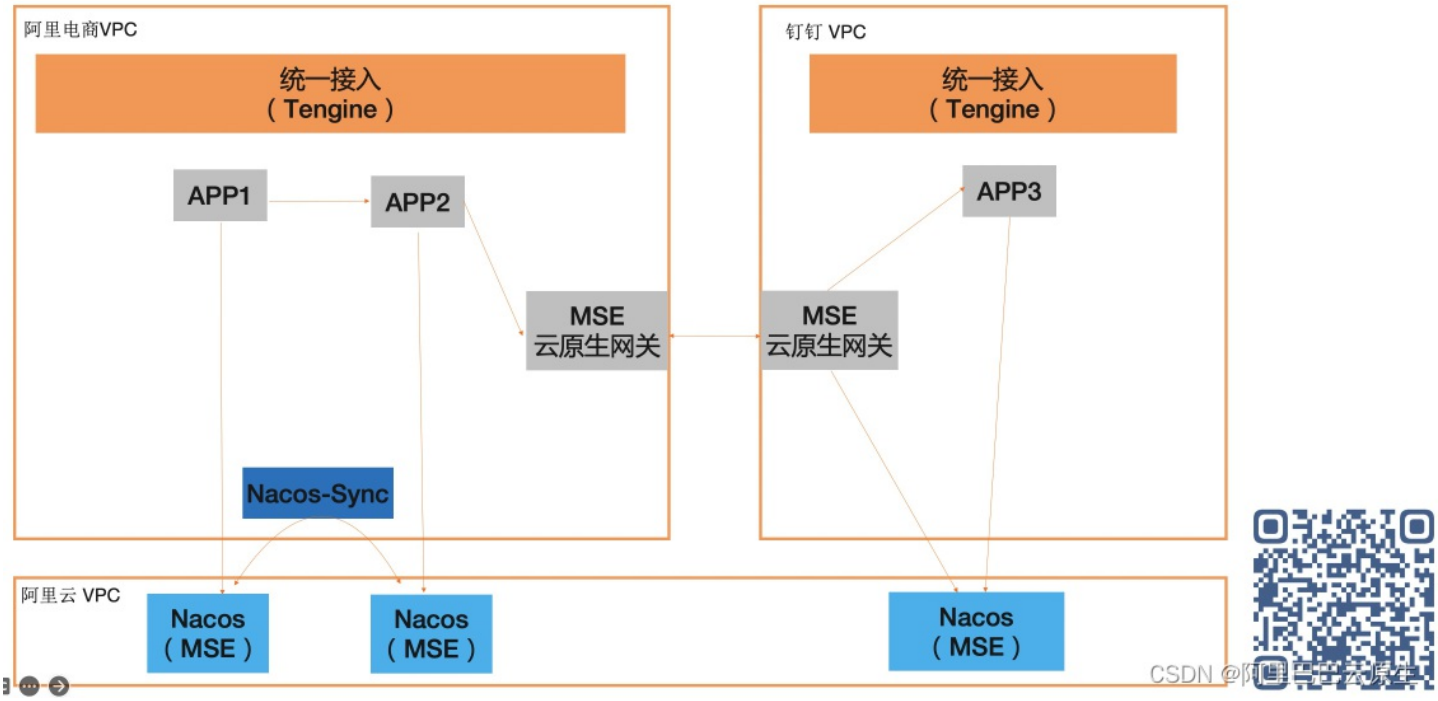


微服务 3.0（100w 实例/业务域拆分/云原生）

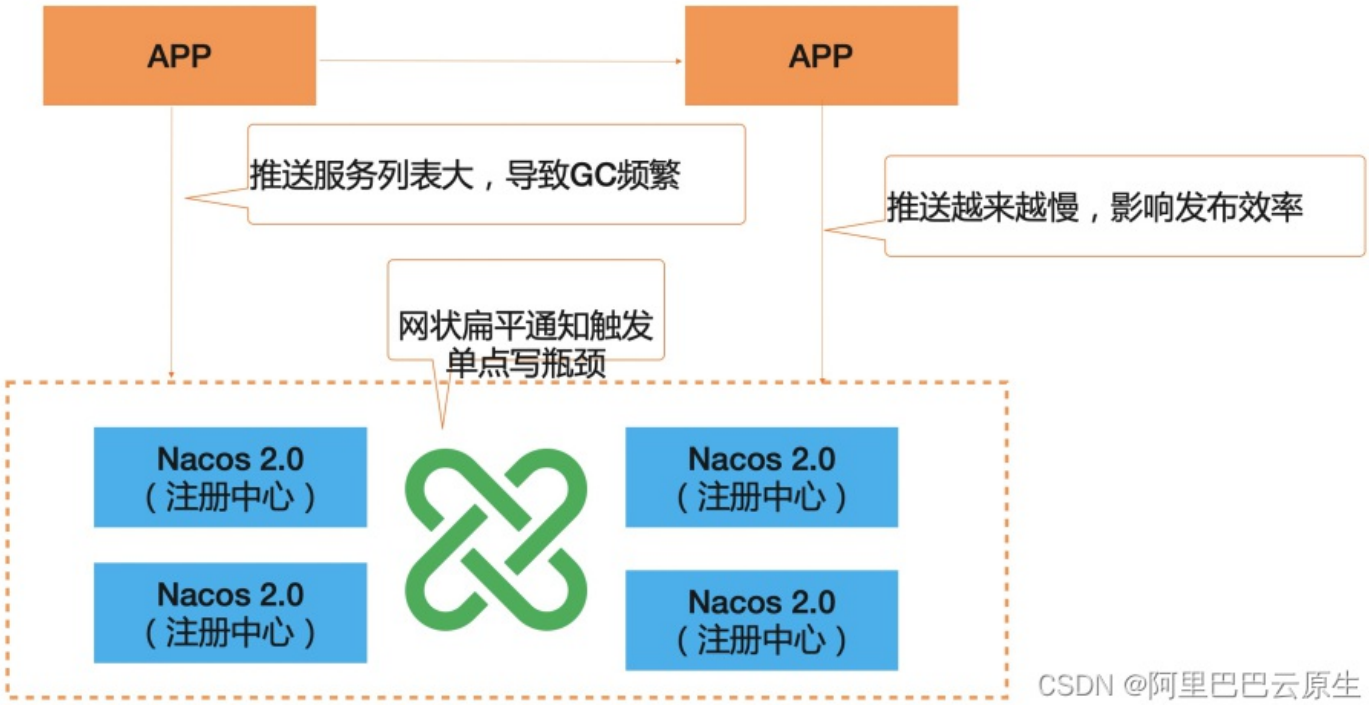
微服务 2.0 架构支撑了阿里双十一的技术奇迹，阿里也陆续开启业务扩张，构建更完整的互联网版图。在这个阶段阿里收购了比较多的公司，技术体系不统一如何形成合力；从线上走到线下后，线下系统对系统稳定性要求更高；云计算发展，如何利用好云的弹性做双十一，这个阶段我们也推出了微服务的云产品，期望通过云产品支撑阿里双十一。

- 收购扩张业务，技术融合挑战大
- 从线上走到线下，稳定性要求更高
- 云计算迅猛发展，迁移云底座技术难度高

业务域切分比较容易，切完之后如何更好的互联互通是一个关键，因此我们内部推出了 Nacos-sync 和云原生网关两个产品。Nacos-sync 适合业务流量超大，协议一致场景。云原生网关适合网络不通，协议不同，跨 Region 等场景。



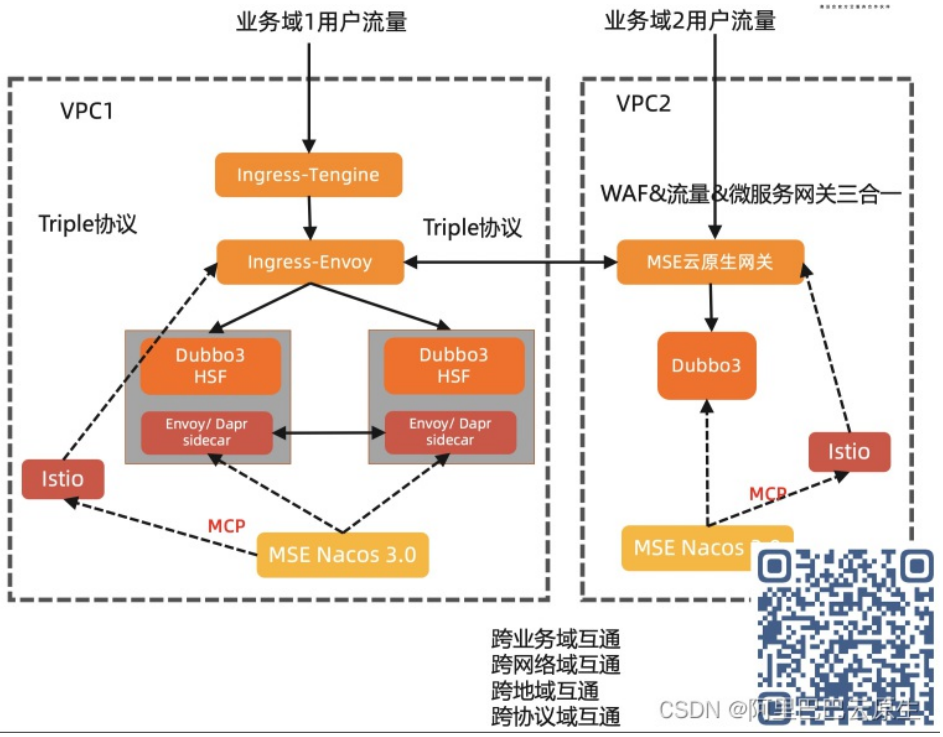
即使从顶层做了业务域拆分，但是最大的电商集群往百万实例演进过程中对注册中心的压力越来越大，我们把聚合窗口时间不断拉长，推送慢了会导致业务发布时间变长，推送快了会对业务消耗较大，因此陷入了两难境地。



这个阶段我们进行问题的分解，首先根据服务列表大小做了一个切分，服务列表多的可以推送慢一些问题也不大，服务列表小的需要及时推送，因此我们优化了聚合推送逻辑，根据服务列表大小做了分级推送。还有一个优化思路是变更只有几个列表变化，因此我们提供了增量推送能力，大幅降低服务变更推送数据量。

通过微服务 3.0 架构演进很好的解决了跨域互通和平滑上云的问题，新业务可以先上云，或者部分业务上云，通过网关做云上云下互通等问题，同时支撑了百万实例微服务架构演进。

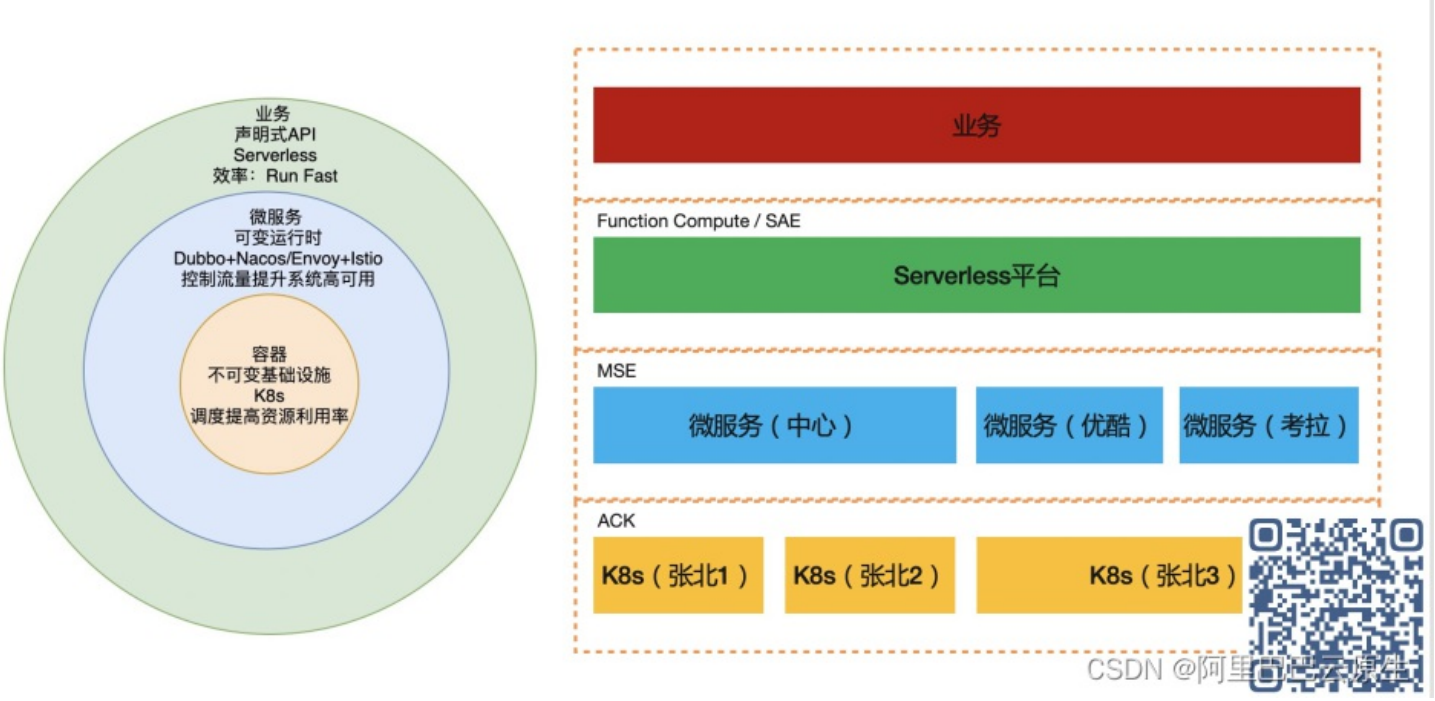
- 微服务3.0
  - 支持100w级实例规模
  - 具备增量推送能力
  - 具备分级聚合能力
- 业务域拆分
  - 解决1000+协同问题
  - 解决1000+子系统复杂度
  - 支持跨域互通
- 云原生
  - 支撑平滑上云
  - 后端BaaS化
  - 业务Serverless化



期望通过我分享阿里微服务发展历程给大家做微服务架构演进提供一些思路和启发。

云原生微服务趋势

随着云原生技术演进，容器以不可变基础设施为理念，解决运维标准和资源利用率问题；微服务以可变运行时为理念，解决研发效率问题，提升系统整体扩展性和高可用。经常有人问我，为什么有了容器的服务发现机制，还需要微服务的注册中心呢？从架构上首先是分层的，小的时候确实也看不到明显区别，大一些就会发现问题，如阿里中心最大微服务集群，底层是多个 Kubernetes 集群，防止一个 Kubernetes 出问题影响全局，底层 Kubernetes 也可以水平扩展，如果依赖了 Kubernetes 的服务发现机制，跨 Kubernetes 服务发现就成了第一个问题。当然底层是一个 Kubernetes 上面也可以是多个微服务环境，微服务可以按照业务域切分。两层可以做解耦，自由环境组合。还有就是阿里微服务体系积累了推空保护、服务治理完整体系，而 Kubernetes 的 CoreDNS 将服务发现强制拉到业务调用链路，每次调用都会做域名解析，因此 CoreDNS 挂的时候业务全部中断。
对于阿里整体正在从百万实例往千万实例的规模演进，这部分也是阿里微服务 4.0 的内容，这部分给大部分公司的借鉴意义有限，因此不做展开。

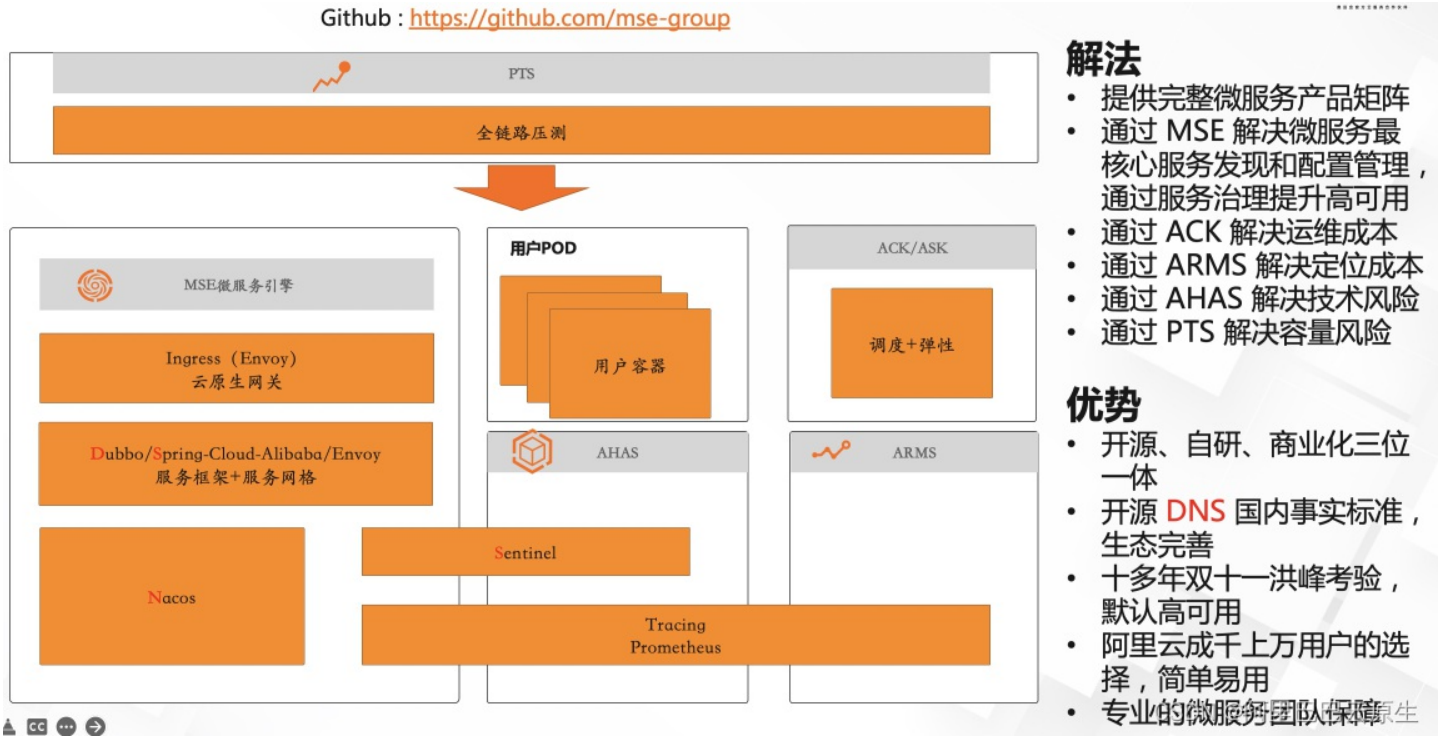


微服务最佳实践

阿里微服务体系经过 10 余年的发展，目前已经通过开源被广泛使用，通过阿里云支撑了成千上万家企业做数字化升级。借此机会把我们的最佳实践总结分享给大家，期望都对大家用好微服务有所帮助。

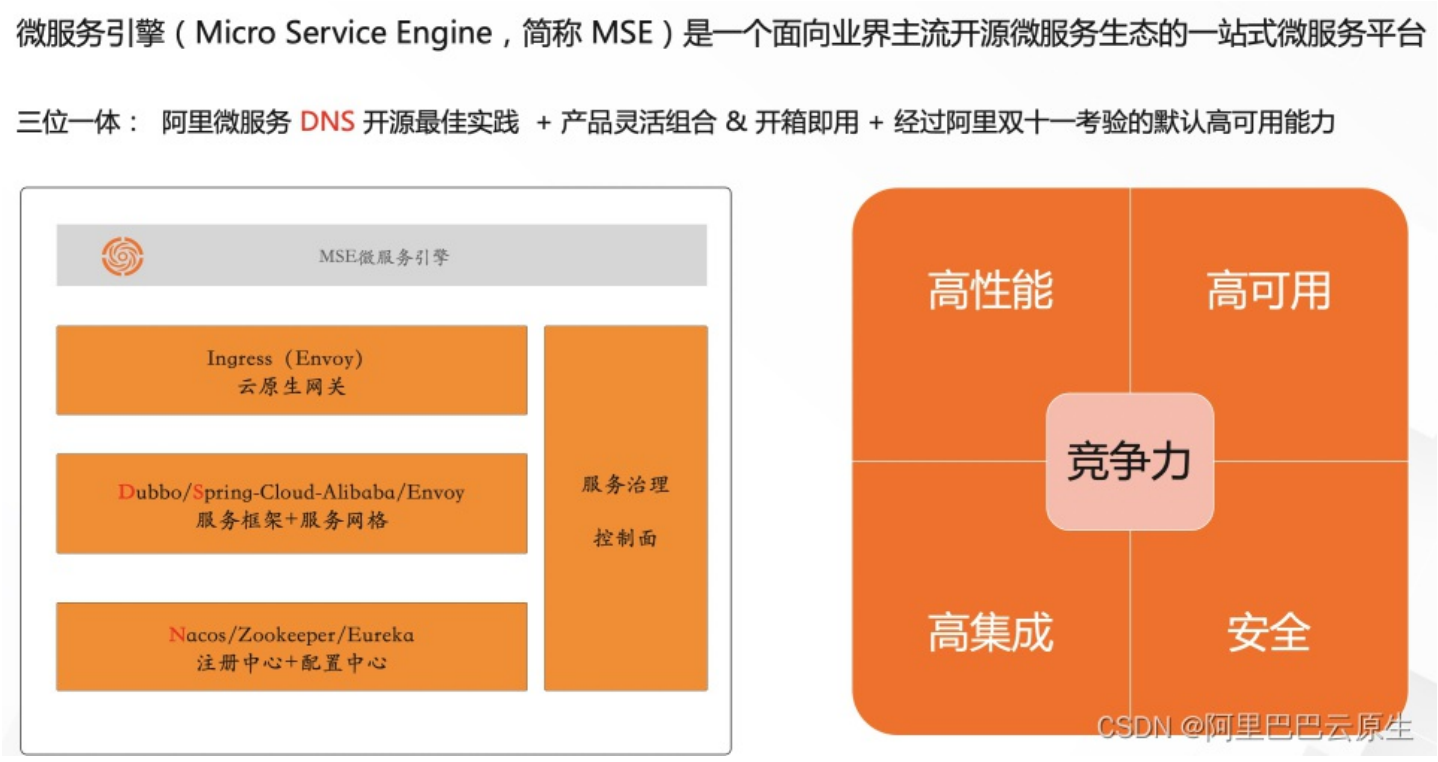
阿里微服务体系简介

通过 MSE + ACK 能够完成第一步云原生技术升级，释放云弹性红利，释放研发效率红利，可以通过可观测和高可用进一步用好微服务体系。



微服务最佳实践

通过注册&配置中心完成微服务拆分；通过网关统一入口，从入口处解决安全和高可用问题；最后通过服务治理提升用户微服务的问题。



网关最佳实践

云原生网关作为下一代网关，提供高集成、高可用、高性能、安全的一站式网关解决方案。

• 统一接入：将流量网关、 微服务网关、 WAF 三合一大幅降低资源和运维成本，需要强调的是云原生网关集成 WAF 的方案有非常好的性能优势，WAF 做为控制面下发防护规则到云原生网关，流量直接在云原生网关清洗完毕直接路由到后端机器，RT 短，运维成本低。

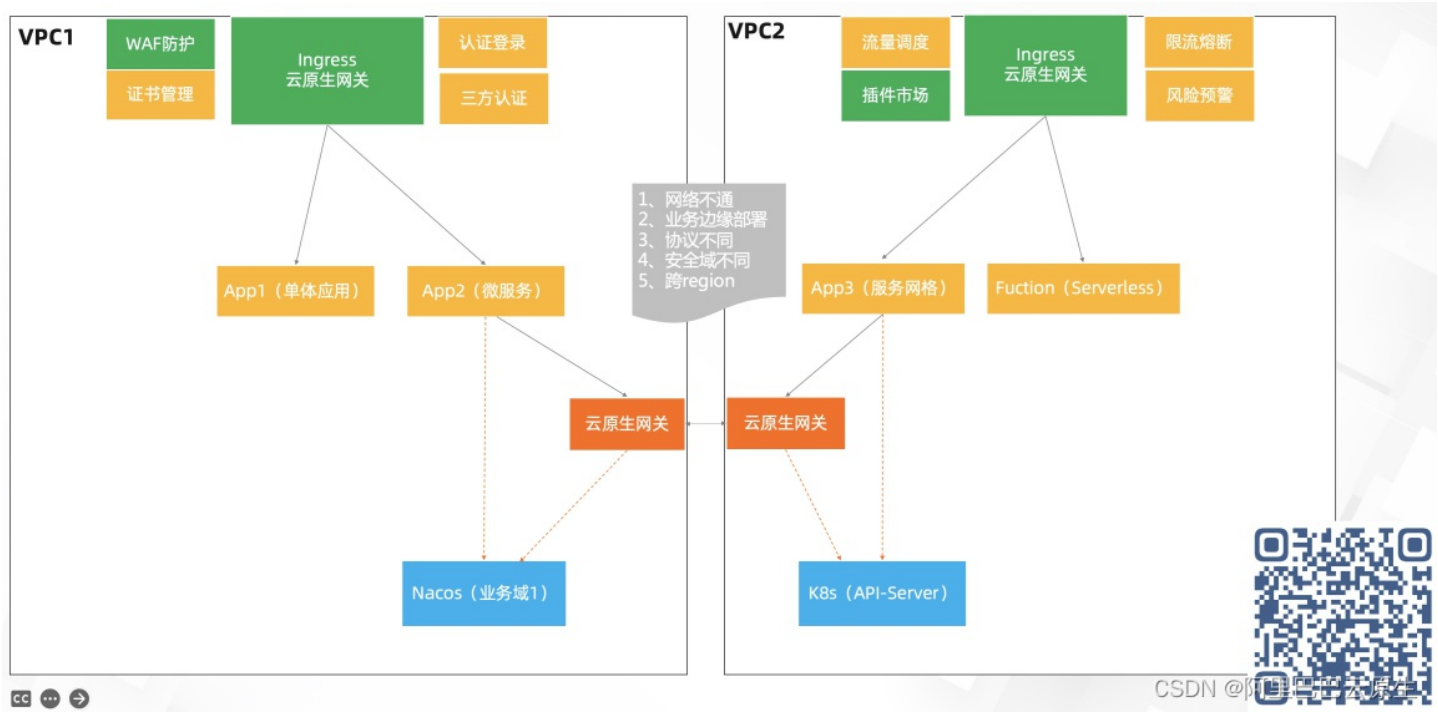
• 统一入口安全防线：自动更新证书防过期，支持 JWT/OAuth2/OIDC/IDaaS 认证机制，支持黑白名单机制。

• 统一东西南北流量：统一解决跨域互通问题，包括跨网络域，跨业务域，跨地域，跨安全域等。

• 统一服务发现机制：支持 Nacos/Kubernetes/DNS/ 固定 IP 多种服务发现方式。

• 统一观测平台：从入口做好 tracing 埋点全链路诊断，丰富业务大盘和告警模板大幅降低网关运维成本。

• 统一服务治理：从入口做限流、降级、熔断等高可用能力，提供全链路灰度方案控制变更风险。统一性能优化：采用硬件加速性能提升 80%，Ingress 场景比 Nginx 性能高 90%，参数调优+模块优化提升 40%。



云原生网关支持 WASM 扩展网关自定义功能，并且通过插件市场提供丰富的插件能力。

服务治理最佳实践

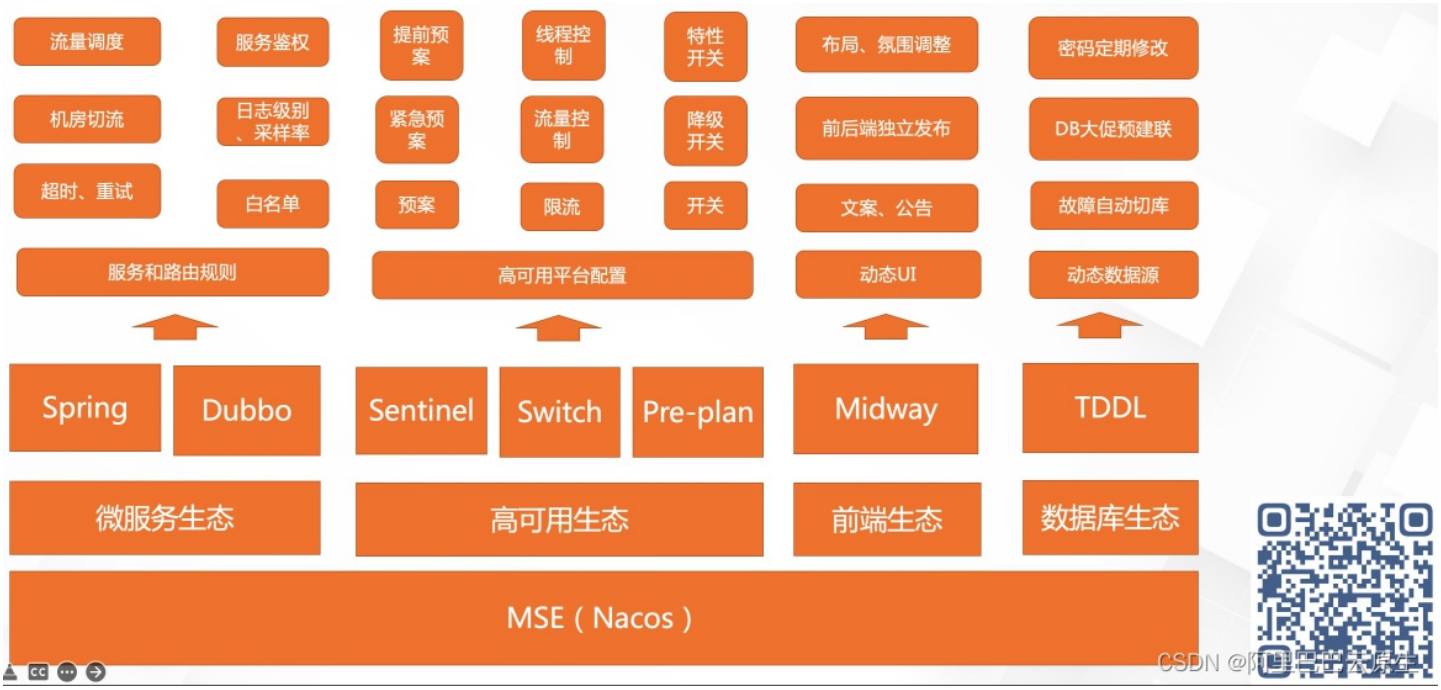
提供零业务侵入，开发，测试，运维全覆盖服务治理能力，提升系统高可用。如发布阶段即使注册中心是毫秒级推送也会有延迟，这个期间就会导致流量损失，因此我们提供了无损上下线能力解决这个痛点。本月我们将服务治理能力通过 OpenSergo 开源，欢迎各位小伙伴参与共建！

日常环境隔离最佳实践

共享一套环境联调开发相互影响，所有环境都独立联调机器成本太高，这个是一个矛盾，我们通过全链路打标能力将流量隔离，让大家可以在一套环境隔离多个逻辑联调环境，巧妙的解决这个问题。

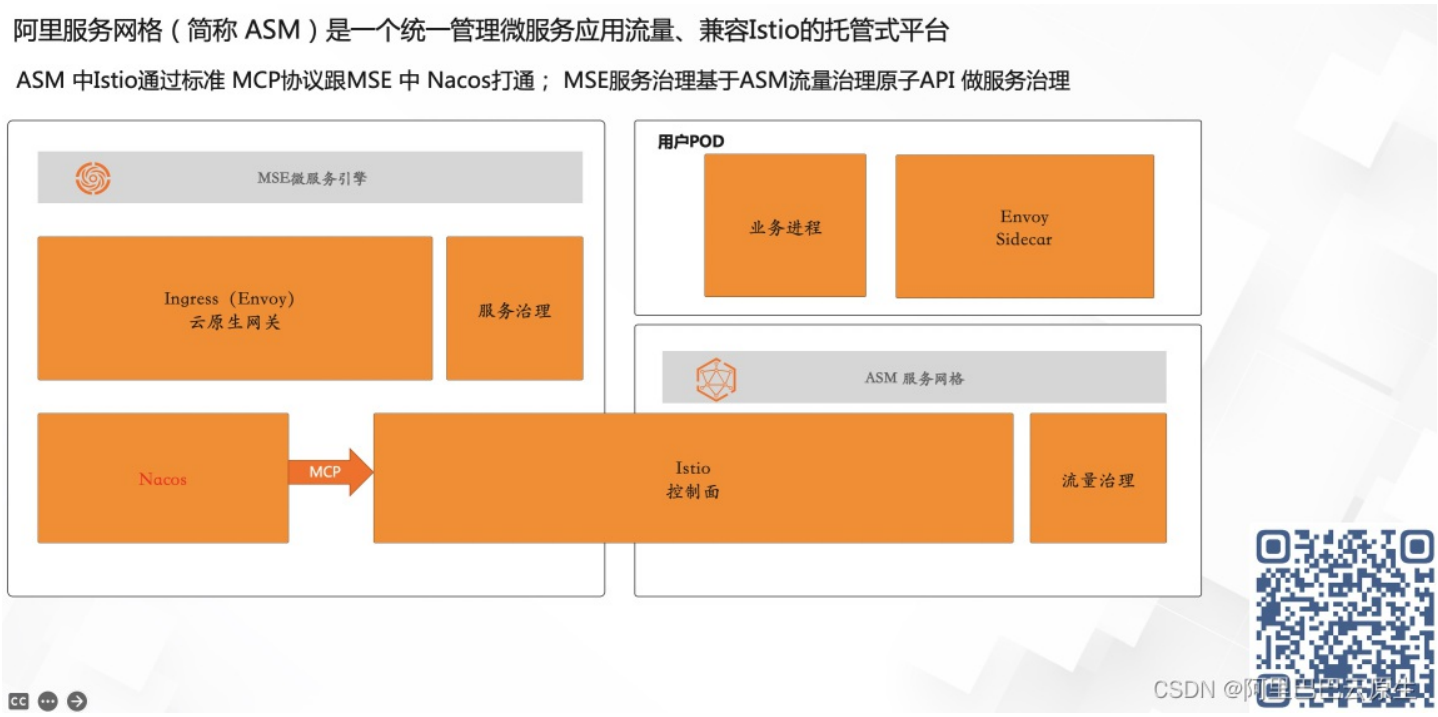
配置管理最佳实践

随着应用规模变大，到每个机器去修改配置运维成本太高，因此需要配置中心统一维护应用配置，将静态业务动态化，动态修改业务运行时行为，提升应用运行时灵活性。



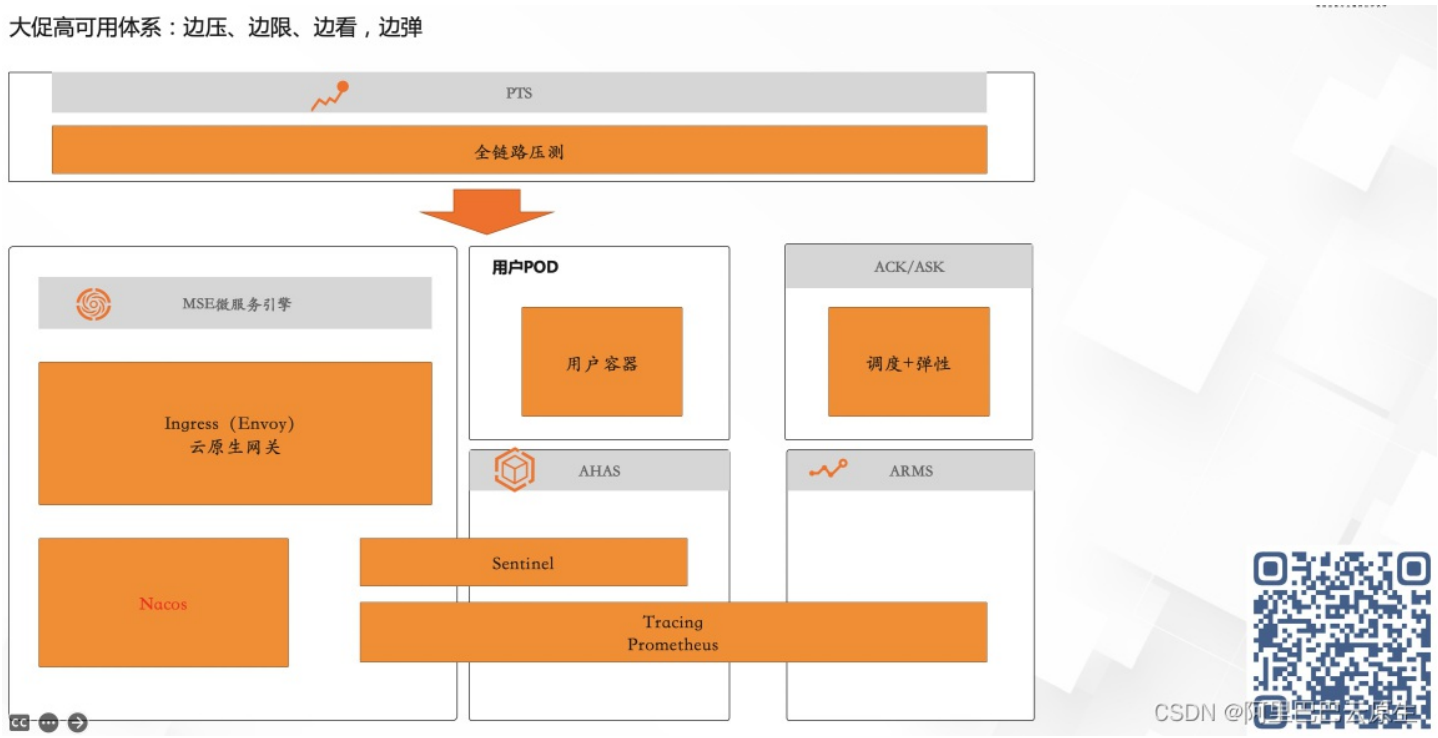
服务网格最佳实践

对于多语言开发有诉求和对服务网关感兴趣的小伙伴可以通过 MSE+ASM 快速构建服务网格解决方案，完成服务互通，快速体验新的技术。



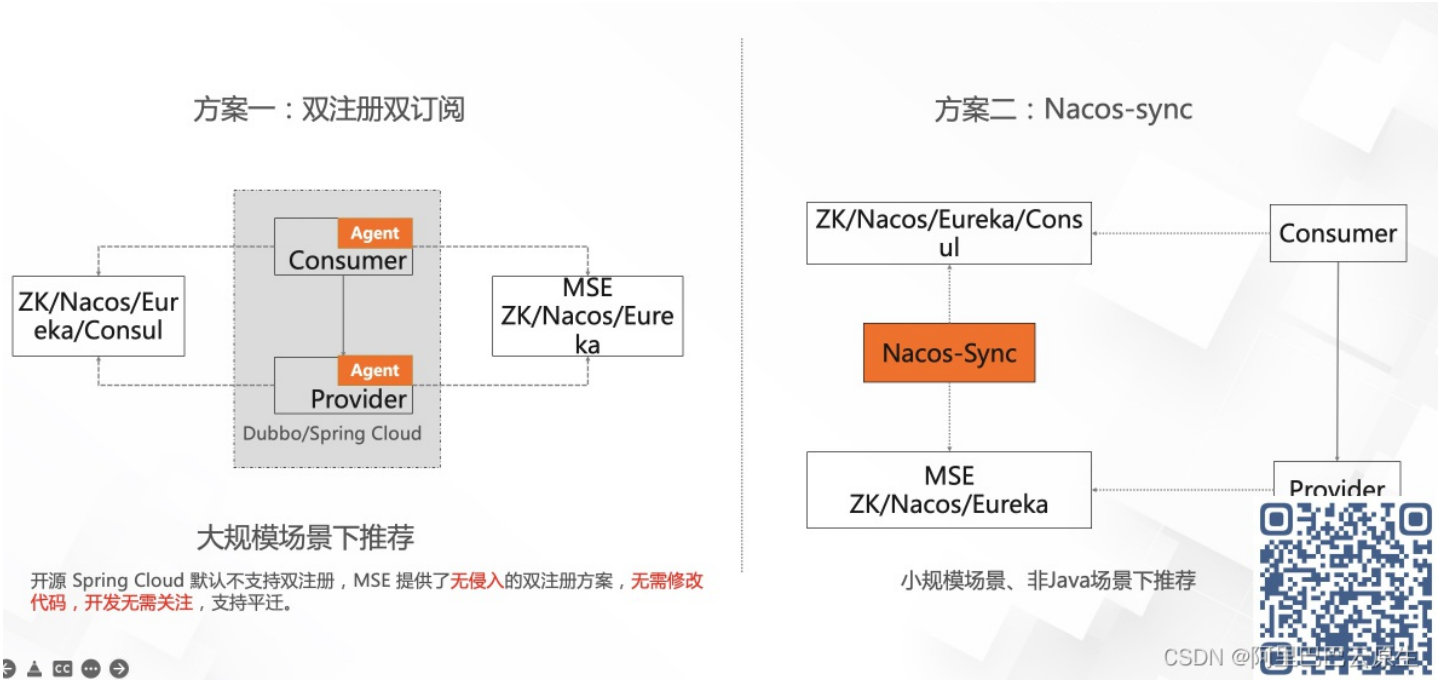
微服务高可用最佳实践

随着业务复杂度变高，业务峰值不可测，面对失败的设计和微服务高可用工具使用就非常重要，可以通过 Sentinel 完成限流、降级、熔断的保护，可以通过 PTS 完成压测，可以通过混沌工程完成破坏性测试，从体整体提升系统高可用。



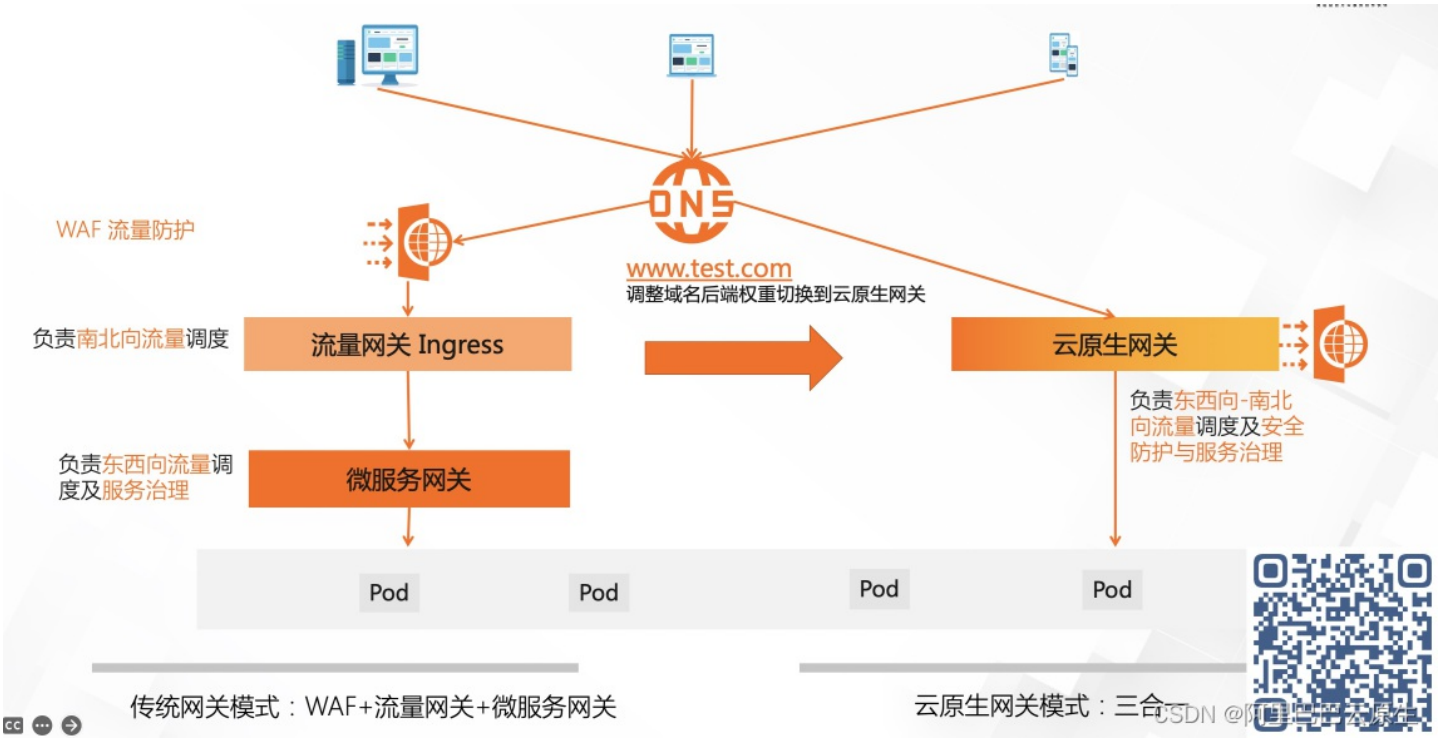
注册中心平滑迁移实践

目前大规模场景推荐双注册，如 1w 实例以上，这样发布周期长，稳定性更高一些。如果不到 1w 实例可以通过 Nacos-sync 同步完成注册中心平滑前一，这样通用型强一些。



网关平衡迁移实践

由于前面云原生网关三合一和性能优势，大家可以通过入口 DNS 灰度切换到云原生网关。

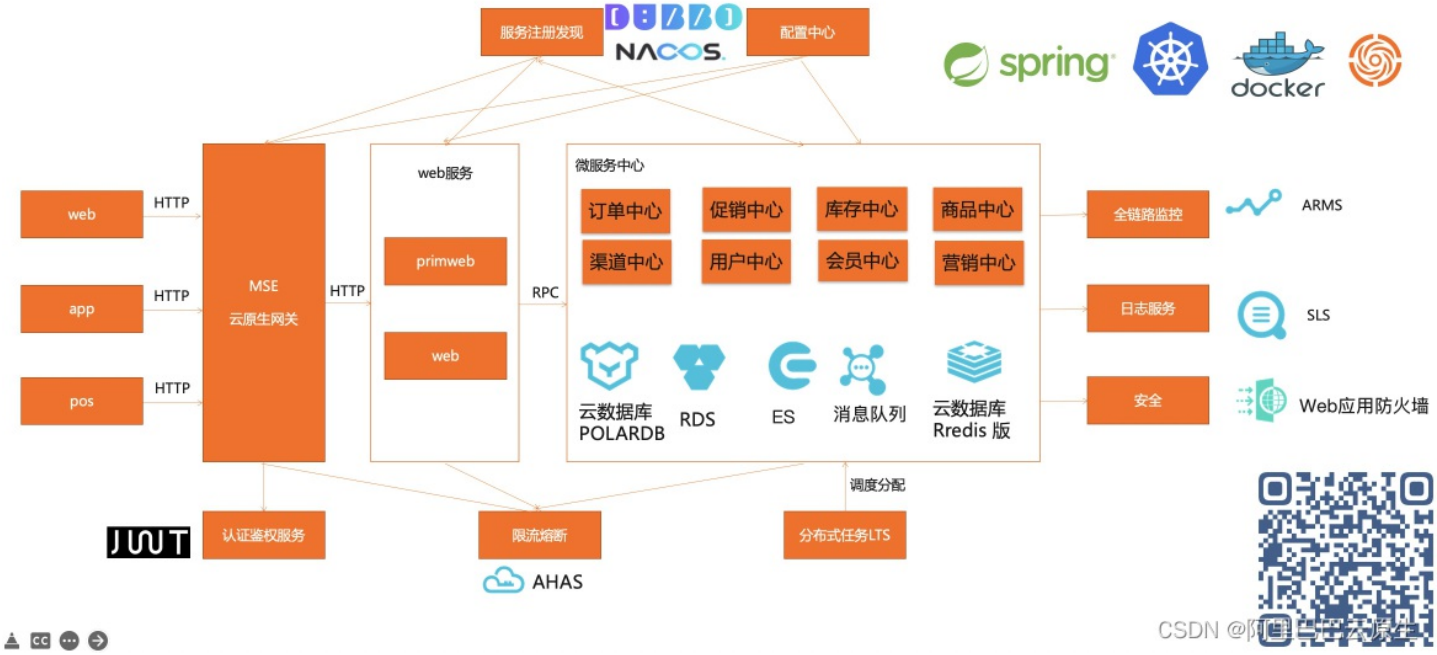


微服务标杆客户

用户上云中有两类典型客户，一类是传统的单体胖应用客户，一类是已经采用了微服务需要用好微服务的用户，我们通过两个标杆客户分享一下。

斯凯奇微服务＋业务中台实践

斯凯奇 2021 年找到我们做数字化升级时间非常紧急，需要双十一前 3 个月左右要完成数字化升级，采用 MSE 微服务+中台解决方案，斯凯奇借助云原生网关完成了东西南北流量的统一控制，借助南北向云原生网关完成安全认证和入口限流，从入口做好流量防护；借助东西向网关完成了多个业务域的互通，新老系统的互通，1 个月左右完成了整个系统的搭建，1 个月左右完成了整个系统压测和高可用验证，并且最终大促业务非常成功，助力斯凯奇双十一 12 亿营收规模。



来电微服务全链路灰度最佳实践

来电的技术挑战

来电科技的业务场景丰富且系统众多，在技术架构上已完成容器化以及微服务化改造，微服务框架使用的是 Spring Cloud 与 Dubbo。随着近年来的高速发展，充电宝设备节点以及业务量都在快速增加，系统的稳定性面临几点挑战:

1.在系统服务的发布过程中如何避免业务流量的损失；
2.系统缺少简单有效的灰度能力，每次系统发布都存在一定的稳定性风险。MSE 微服务治理提供了开箱即用且无侵入的线上发布稳定性解决方案以及全链路灰度解决方案，帮助来电科技消除发布风险、提升线上稳定性。

来电全链路灰度最佳实践

1.来电科技选用 MSE 微服务治理专业版来实现无侵入微服务治理能力，无缝支持市面上近 5 年所有的 Spring Cloud 和 Dubbo 的版本，不用改一行代码，不需要改变业务的现有架构就可以使用，没有绑定。

2.MSE 微服务治理专业版提供了全链路灰度解决方案帮助来电科技快速落地可灰度、可观测、可回滚的安全生产三板斧能力，满足业务高速发展情况下快速迭代和小心验证的诉求；

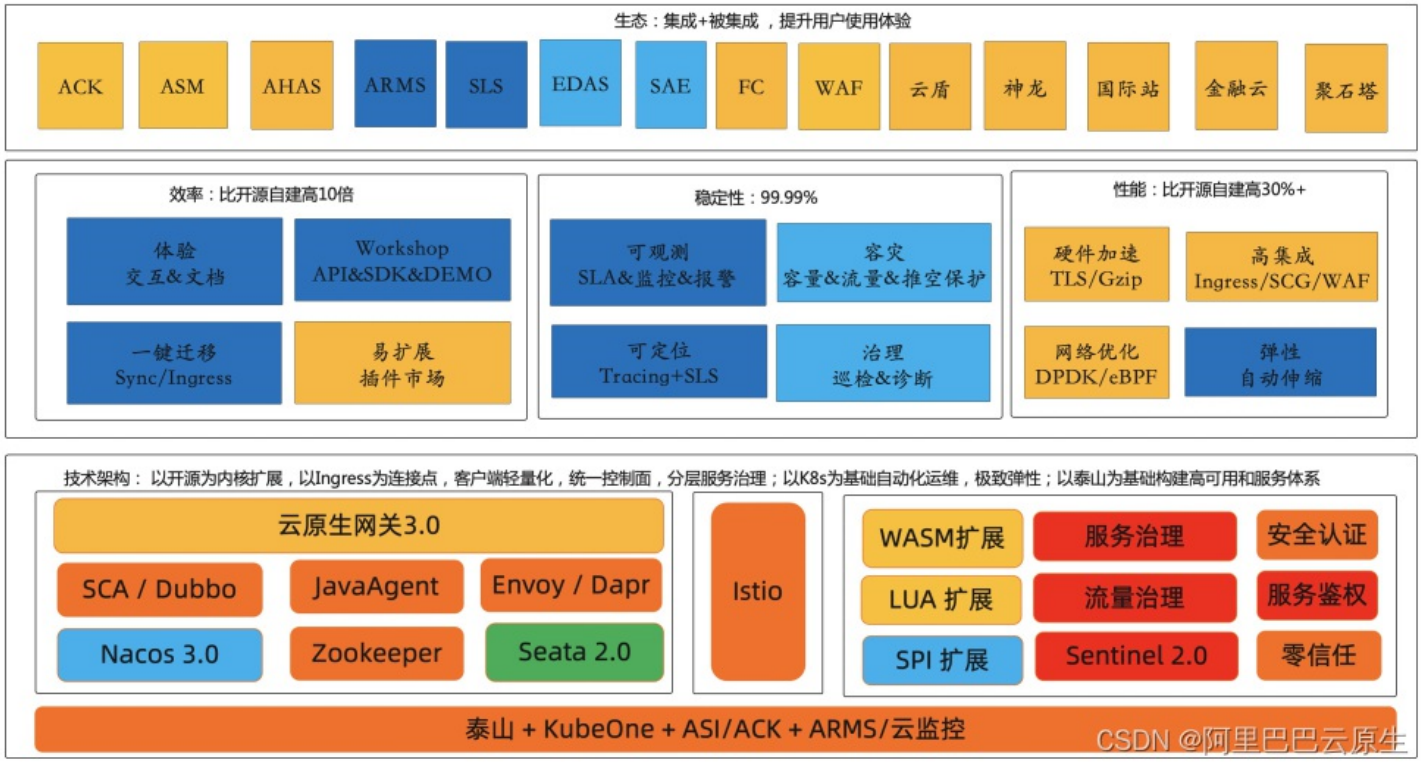
3.MSE 微服务治理的无损上下线能力，对系统服务的全流程进行防护，通过服务预热、无损下线、与 Kubernetes 微服务生命周期对齐、延迟发布等一系列能力，保证在服务冷启动或销毁过程中，业务连续无损。

4.MSE 微服务治理的离群实例摘除能力，可以做到让服务消费者自动检测其所调用提供者实例的可用性并进行实时的权重动态调整，以保证服务调用的成功率，从而提升业务稳定性和服务质量。

阿里云微服务生态与规划

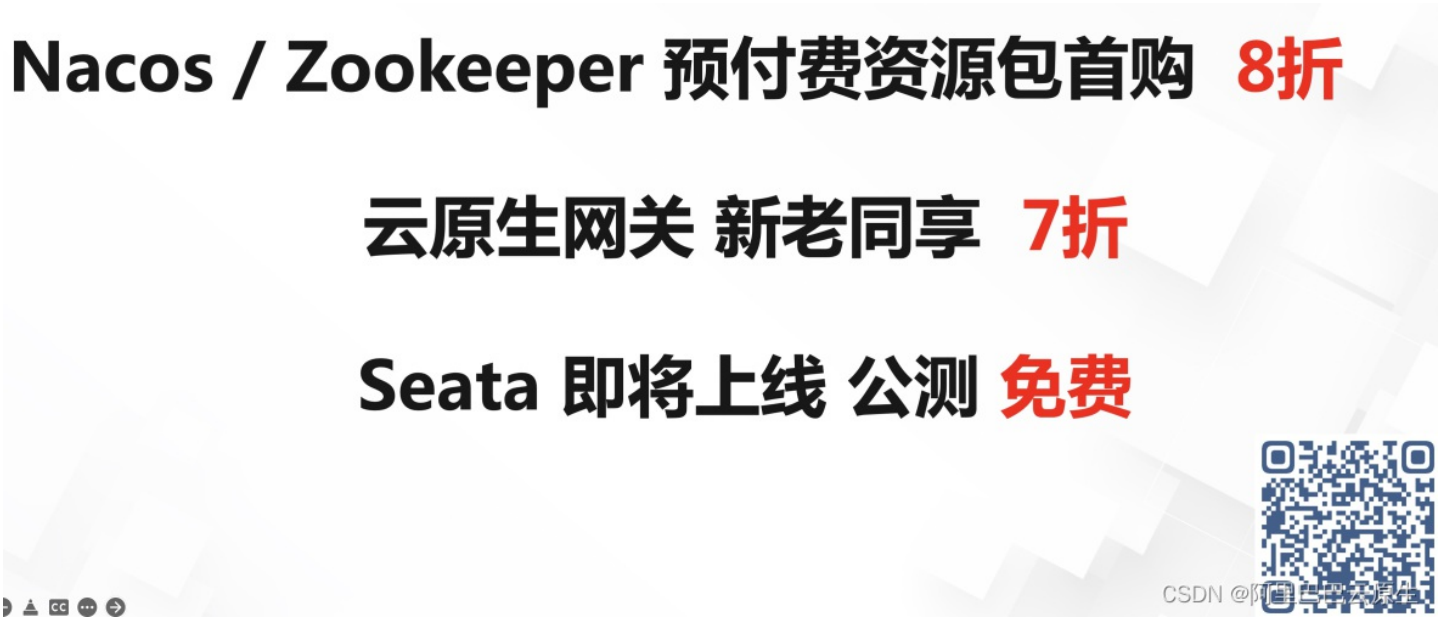
阿里开源微服务会贴着服务治理帮助开发者用户微服务，云产品做好产品集成提升大家的使用体验。

ACK+MSE = 云原生架构升级解决方案
ASM+MSE = 服务网格解决方案
AHAS + MSE = 微服务高可用解决方案
ARMS + MSE = 微服务可观测解决方案
EDAS + MSE = APaaS解决方案
SAE + MSE = 微服务 Serverless 解决方案
WAF + 云盾 + IDaaS + MSE = 微服务安全解决方案

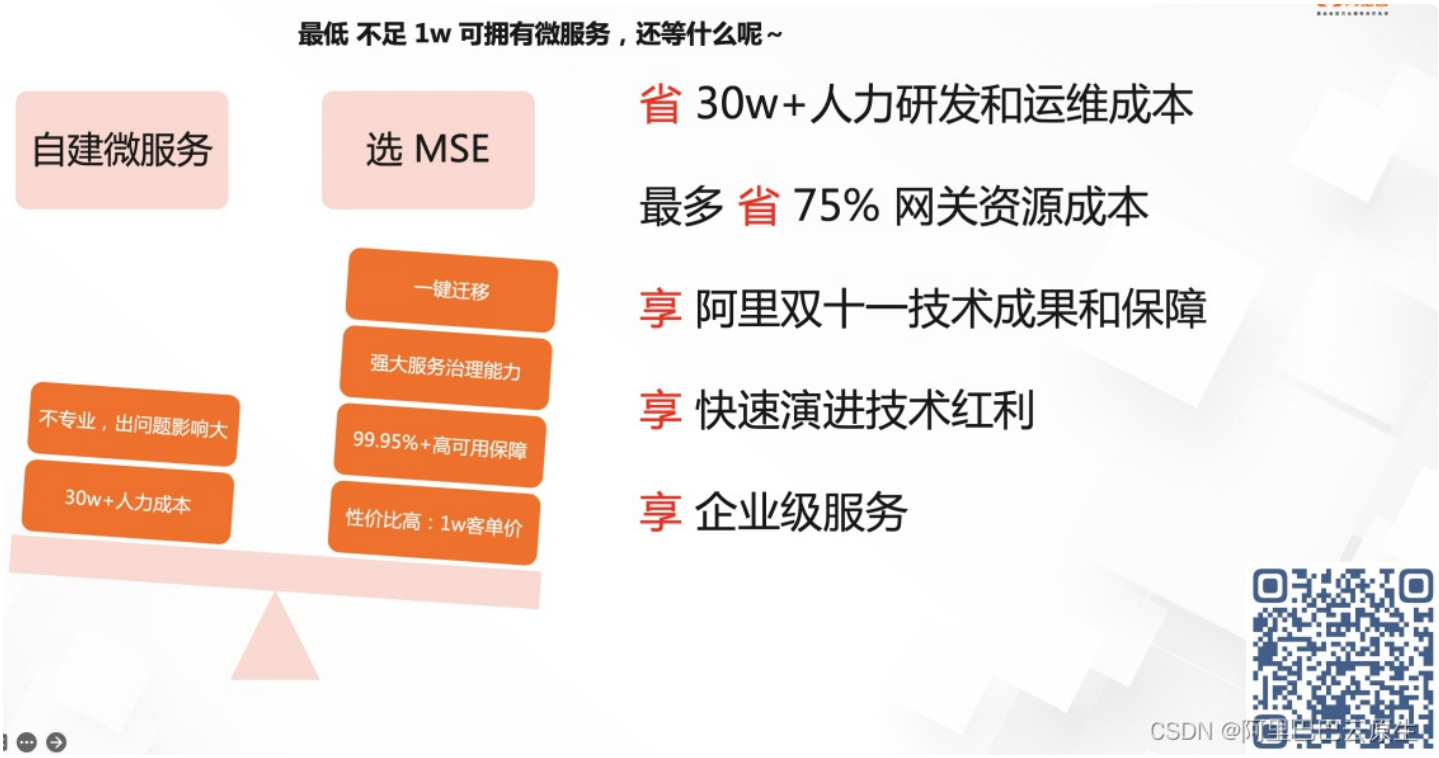


运营活动

限时折扣（4.21-4.30）



微服务全家桶，省、省、省～



下期预告 - Kubernetes Ingress 最佳实践

随着 Kubernetes 普及，Ingress 成为云原生架构的流量入口，云原生网关作为 Ingress 的最佳实践如何助力业务降本提效，如何从入口处建立安全、高可用的防线，如何从 Nginx Ingress 实现平滑切到云原生网关，4.28 将为大家揭晓!

阿里云 MSE 抢购入口：
https://www.aliyun.com/product/aliware/mse
MSE 国际站购买入口：

https://www.alibabacloud.com/product/microservices-engine

点击此处即可观看微服务最佳实践相关视频~

创作打卡挑战赛

赢取流量/现金/CSDN周边激励大奖