2020 年，从架构谈起，到 Mesh 结束 (https://www.kubernetes.org.cn/8037.html)

2020-07-24 12:01 alicloudnative (https://www.kubernetes.org.cn/author/alicloudnative) 分类：Kubernetes大咖交流 (https://www.kubernetes.org.cn/exchanges) 阅读(213) 评论(0)

作者 | 张羽辰（同昭）阿里云交付专家

> 导读：如今，几乎所有的事情都离不开软件，当你开车时，脚踩上油门，实际上是车载计算机通过力度感应等计算输出功率，最终来控制油门，你从未想过这会是某个工程师的代码。

当我们谈论架构时，我们到底在谈论什么？

面向对象编程？函数式？模块化设计？微服务？这些词汇貌似都和架构这个 buzzword 有点关系，的确我们这个领域充满了很多难以理解的词汇，这些词汇从英语翻译到中文已经丧失了部分上下文，再随着上下文的改变使得意义彻底扭曲，比如：引擎、框架、架构、应用、系统……诚然大家都或多或少对这些词语达成共识，在工作中使用这些词汇进行沟通，某时就是指“我们都懂的那个东西”，但是在我深入的想聊聊架构或者说软件架构时，的确不得不问自己这个问题，我们到底是谈论什么？

Kubernetes 1.18 版本

我们很高兴宣布Kubernetes 1.18的交付，这是我们2020年的第一版！Kubernetes 1.18包含38个增强功能：其中15个功能已趋于稳定，beta版本中有11个，alpha版本中有12个。(https://www.kubernetes.org.cn/tags/kubernetes1-18)

关注「K8S中文社区」微信公众号
回复“文档”
获取K8S文档下载链接
回复“加群”
加入K8S微信技术交流群

最新文章

Porter 进入 CNCF 云原生全景图，新版本即将发布！
2020-07-27 评论()
(https://www.kubernetes.org.cn/8118.

掌门教育微服务体系 Solar | 阿里巴巴 Nacos 企业级落地上篇
2020-07-27 评论()
(https://www.kubernetes.org.cn/8117.

JFrog助力Google Anthos混合云Devops实践，实现安全高质量的容器镜像管理
2020-07-27 评论()
(https://www.kubernetes.org.cn/8114.

灵雀云入选Gartner 2020中国ICT技术成熟度曲线报告，容器技术处于顶峰
2020-07-27 评论()
(https://www.kubernetes.org.cn/8113.

热门推荐

你的微服务还差个容错机制
2020-03-26
(https://www.kubernetes.org.cn/7060.

Kubernetes 1.18GA，15个稳定 11个beta，引入kubectl debug命

事实上，架构这个词根据上下文所确定的范围较为固定，建筑学上的架构指代房屋结构、整体设计、组合构成等，而这些 high-level 设计往往并不需要全面了解底层，就像使用 RestTemplate 进行 WebService 调用时，我们也不关心 socket 是在四层连接的一样，**因为细节被隐藏了**。

但是，建筑学上的架构与软件架构却又极大的不同之处，问题出现在“软件”这个词上，按照 software 的词解，ware 是指产品一样的东西，而 soft 则强调易变，这是与 hardware 所对应的。我们希望“软件”能够进行快速的修改，应该能够快速响应甲方或者客户的需求，所以软件架构必然不像建筑架构一样，建筑一经建成，修改的成本极高，**而软件应该走对应的方向，发挥易于修改的特点**。

> “现在的大多数软件非常像埃及金字塔，在彼此之间堆建了成千上万的砖块，缺乏结构完整性，只是靠蛮力和成千上万的奴隶完成。” —— Alan Kay。
>
> 笔者认为，虽然这句话表达的意思我很赞同，但实际上，金字塔作为帝王的陵墓，是有着完整的设计逻辑，并且随着好几座金字塔的迭代的，以及逐渐完备的施工管理，后期金字塔是非常杰出的建筑代表，并作为地球上最高的人造建筑持续了好几千年。关于金字塔是否由奴隶建造还是存有争议。（图片来自 Isabella Jusková @ Unsplash）。

作为工程师，我们一方面关注软件产品的能力和行为，这往往是一个项目的起点，另一方面我们需要关注软件的架构设计，因为我们希望设计有着弹性、易于维护、高性能、高可用的系统，更希望系统能够不断演进，而不是在未来被推倒重做。所以，回正我们的视野，当我们决心要设计一个好的架构时，我们需要明确，**架构往往决定的是软件的非功能性需求**。这些非功能性需求有：

1. **易于开发**：我们希望工程师一进入团队就可以立刻开始进行研发工作，我们希望代码易于阅读与理解，同时开发环境足够简单统一，说到这里大家可以回想下当你进入项目时，学习上下文的痛苦。当我们开始采用 docker 辅助开发时，时任架构师提出了一个要求，只要一行命令就可以使用 docker 启动本地测试环境，而且必须所有的微服务都要做到这一点。痛苦的改造完成后，三年后进入项目的同学只需要安装好 docker，再在 ternimal 中运行一句 ./run-dev.sh 就能够获取一个具有完整依赖的本地环境，提效明显。
2. **方便部署**：如果系统的部署成本很高，那使用价值就不会很高了，我们很多企业都存在那种动也不敢动，改也不敢改，停也不敢停的系统，除了祈祷它别挂掉好像没有别的办法，或者很多企业都采用了 K8s 这种先进的编排系统，但是在应用部署和上线时，还是走的每周四变更的路子。现代的发布方式 AB、金丝雀、灰度无法采用是因为改造成本过高，或者没有足够的自动化测试来保证改动安全，更别提将发布做到 CICD 里面了。
3. **易于运维**：DevOps 的初衷是建立一种缩短运维与研发距离的文化，让出现问题后更容易处理，希望让大家将视野放在产品上而不是限定自己的工种，这并不是期望运维的同学能够成为 Java 专家，迅速的进行 heap 分析发现问题，我们强调的是运维时的闭环能力。在软件产品层面，我们也

令

2020-03-26
(https://www.kubernetes.org.cn/7055.

英国Monzo银行，用K8s管理1600个微服务实践

2020-03-21
(https://www.kubernetes.org.cn/7001.

Java vs. Go 微服务 – 负载测试(复赛)

2020-03-20
(https://www.kubernetes.org.cn/6988.

与时俱进 – 为什么要使用云原生数据库?

2020-03-15
(https://www.kubernetes.org.cn/6953.

2020年Service Mesh 三大发展方向

2019-12-13
(https://www.kubernetes.org.cn/6255.

社区标签

BoCloud博云
(https://www.kubernetes.org.cn/tags/bocloud%e5%8d%9a%e

CI/CD (https://www.kubernetes.org.cn/tags/cicd)

CNCF
(https://www.kubernetes.org.cn/tags/cncf)

DevOps
(https://www.kubernetes.org.cn/tags

Docker
(https://www.kubernetes.org.cn/tags/c

etcd (https://www.kubernetes.org.cn/tags/etcd)

GO (https://www.kubernetes.org.cn/tags/go)

Helm
(https://www.kubernetes.org.cn/tags/he

Istio
(https://www.kubernetes.org.cn/tags/istio)

Jenkins
(https://www.kubernetes.org.cn/tags/jenk

k8s代码解读
(https://www.kubernetes.org.cn/tags/k8s%e4%bb%a3%e7

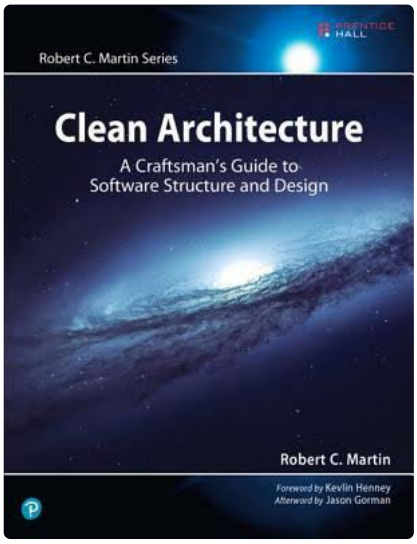
kubeadm
(https://www.kubernetes.org.cn/tags/kube

kuberenetes
(https://www.kubernetes.org.cn/tags/kuberene

希望产品是足够独立的、自治，可以独立部署，能够做到横向扩展，有着完整的可观测性，毕竟当今的硬件成本很多时候是远远小于人力的。

4. **维护成本**：随着时间的推移，给软件增加新功能就会变的越来越难，越是运行长久的项目就会陷入重写还是重构的苦恼。往往风险在与，修改代码会增加破坏已有功能的风险，而且技术债也会越来越多难以偿还，即使是重写某些功能和模块，我们也很难确定是否真的覆盖到了所有的功能，简而言之，don't break anything 的确很难做到。
5. 以及最重要的一点，**演进能力**：良好的架构设计应该能让系统处于易于演进的状态，能够实现给飞驰的汽车换轮胎的能力，而不会被框架、底层的某种数据库、操作系统或者其他东西所绑架，但是这太难以做到了。的确，在项目进行技术选型时，因为某种数据库的特性而有倾向，但是在上层设计中，我们必须保证不依赖于数据库的特性，而将使用这些特性的地方放到底层细节中。我们也需要考虑，不使用 Spring 提供的 Dependency Injection，我们该如何组织我们的 beans，也要考虑将来系统的前端是 web 还是 mobile 还是都要支持?

这里引用 Robert C·Martin（Uncle Bob）的原语，"软件产品是有两方面的价值，一方面是实现功能的价值，另一方面是架构的价值，**而架构的价值可能更重要一些，因为它代表着软件 soft 的特性**。"



> 本书例子过少，而且缺乏现有流行框架的重构或者改进建议，有点形而上，但是在方法论层面笔者还是认为值得一读。Robert C·Martin 对数据库（特指 RDBMS）的态度很值得讨论，首先他认为数据库是一种细节，在架构中应该与业务解耦，他强调业务代码与数据库的无关性。同时在我们的代码进行计算时，表格往往不是理想的数据结构，比如有些场景会使用树、DAG 等等。可以回想一下，当你需要把一个树存入数据库时，你该如何实现?

微服务是一种软件架构，不要扩展它

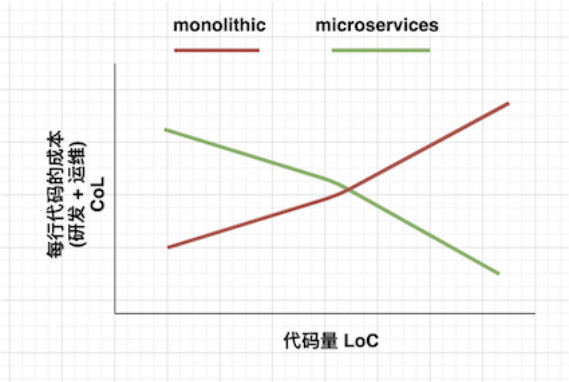
根据我们之前的讨论，后端系统采用微服务是不会影响到其功能上的价值，本质上微服务化和单体应用的差别并不会表达在功能上，很多微服务进展不顺利的同学会经常说到：这东西用单体写早就完事儿！的确是这样，**这侧面也印证了微服务只是一种软件架构，而不是别的神奇的东西，并不是某个业务需求必须要使用微服务完成，我们看中微服务，也是看中了架构方面的优势，即那些非功能性需求**。也有人使用 pattern 来描述它，也有人说和 SOA 基本上是一个东西，只是粒度不同，所以我们一开始就别相信这个世界有灵丹妙药，也别期望有个什么技术能够瞬间替代 Oracle。

作为开发"企业级后端应用"的同学，我们经常会面临很多非业务需求上的苦恼：有时我们需要同时支持移动端、移动 web、桌面端三种客户端；有时候我们需要支持不同的协议比如 JSON 或 XML；有时我们又需要使用不同的中间件传递消息；或者在研发时，我们知道有一个地方写的不好，我们想在未来补课重构；我们想尝试最新的技术但是代价过高；系统无法扩容，或者成本极高；系统过于复杂无法在本地运行导致

Kubernetes (https://www.kubernetes.org.c
Kubernetes1.5 (https://www.kubernetes.org.cn/tags/kubernetes1-5)
Kubernetes1.6 (https://www.kubernetes.org.cn/tags/kubernetes1-6)
Kubernetes1.7 (https://www.kubernetes.org.cn/tags/kubernetes1-7)
Kubernetes1.8 (https://www.kubernetes.org.cn/tags/kubernetes1-8)
Kubernetes1.9 (https://www.kubernetes.org.cn/tags/kubernetes1-9)
Kubernetes1.10 (https://www.kubernetes.org.cn/tags/kubernetes1-10)
OpenStack (https://www.kubernetes.org.cn/tags/openstack)
PaaS (https://www.kubernetes.org.cn/tags/paas)
Pod (https://www.kubernetes.org.cn/tags/pod)
Prometheus (https://www.kubernetes.org.cn/tags/prometheus)
Rainbond (https://www.kubernetes.org.cn/tags/rainbond)
Rancher (https://www.kubernetes.org.cn/tags/rancher)
Serverless (https://www.kubernetes.org.cn/tags/serverles
Service (https://www.kubernetes.org.cn/tags/service)
service mesh (https://www.kubernetes.org.cn/tags/service-mesh)
云原生 (https://www.kubernetes.org.cn/tags
云计算 (https://www.kubernetes.org.cn/tags/%e4%ba%91%e8%ae%a1%e7%ae%97)
企业案例 (https://www.kubernetes.org.cn/tags/%e4%
存储 (https://www.kubernetes.org.cn/tags/%e5%ad%98
安全 (https://www.kubernetes.org.cn/tags/%e5%ae%89
容器 (https://www.kubernetes.org.cn/tags/%
容器云 (https://www.kubernetes.org.cn/tags/%e5%ae%b9%e5%99%a8%
容器云平台 (https://www.kubernetes.org.cn/tags/%e5%ae%b9%e5%99%a8%e4%ba%91%
微服务 (https://www.kubernetes.org.cn/tags
日志 (https://www.kubernetes.org.cn/tags/%e6%97%a5%e5%bf%

极低的效率……这些苦恼才是采用微服务的主要驱动力，回到我们对软件架构的讨论之中，我们希望的是通过足够松耦合的独立服务，来降低组件之间变化的成本，也就是说今天更新发送通知的功能，并不会影响到用户查看购物车，也不会让研发人员半天改完，再等三天才能上线。

但是世界上没有免费的午餐，虽然我们知道微服务有很多很好的特性，比如组件即服务、松耦合、独立部署、面向业务、高维护性、高扩展性等等，这里并不想展开讨论它的好处，我们先考虑投入成本。假设我们每个同学都完整的学习了微服务的所有知识，对市面上的框架、产品非常熟悉，摩拳擦掌准备开始，在拆解完几个服务后，我们会发现，没有足够的自动化手段，靠手动的方式进行测试、编译、部署、监控，这是显而易见的会降低体验，如果没有优化好的部署策略，所有的服务都在某个发布日上线，那更是一种灾难。



> 随着规模的扩大，单体应用的代码改动成本会越来越大。很多时候我们微服务的架构实践是存在误区的，我们总认为流量经过某个 gateway 后直达某个服务，确忽视了服务之间调用的场景，理想的微服务架构应该是一张网，每个节点都是独立的、自治的服务。

一些之前使用单体很容易做到的场景，在分布式的环境下会更加困难。比如我们可以通过 RDBMS 提供的数据库事务来支撑一致性，但是如果订单服务和价格服务分离，势必要进行分布式事务来保证一致性（往往是最终一致性），而分布式事务的成本和难度就不用赘述了。在单体环境下，我们可以很轻松的使用切面进行权限验证，而在微服务的场景中，服务之间相互调用是难以控制的。

拆分服务或者服务边界划分是另一件很难做到的事情，最吃香的理论也许是根据 DDD 去进行划分，天然的领域或者子域（domain）貌似都能对应一个服务，因为足够的界限上下文（bounded context）能够保持服务的独立性，使其细节被隐藏在界限之内，听起来是个不错的主意。但是现实却十分残酷，使用 DDD 生搬硬套去进行软件开发的例子不在少数，成功例子也难以复制。

虽然我在实践中也经常使用业务领域去进行服务划分，但是我并不认为这是 DDD 的做法，没有必要规定有多少个 domain 就有多少服务，也不需要规定 sub domain 能否独立服务。**与其进行顶层设计一揽子的解决方案，我更相信演进的力量，如果你真的需要拆分一个服务，足够的基础设施与自动化工具应该允许你低成本的去做，而不是一开始就画好所有的架构图**。这就跟所有的改革一样，革命派往往不是一步功成，而是逐渐的积累的。所以使用微服务，当你能够负担的起（only you can afford it），也表示你能负担的失败一样，技术世界不存在一蹴而就，all in 非常危险。

卫报网站（Guardian）的微服务改造就是一个很好的例子，网站核心依旧是一个巨大的单体，但是新功能通过微服务实现，这些微服务调用单体所提供的 API 来完成功能。对于常常出现的市场活动（比如某个体育比赛的专用板块），这种方式能够快速实现活动页面与功能，完成业务需求，并在活动结束后删除或丢弃。我之前参与项目中，也通过等量替换与重构，慢慢绞杀（Strangler Pattern）掉一个巨大的陈旧的 JBoss 应用。

PlayStation 首席设计师 Mark Cerny 在今年的 PS5 新主机的技术分享中提到，游戏主机需要平衡好演进与革命（balance the evolution and revolution），我们不想丢掉多年来开发者的积累，在复用过去的成功经验时，我们也希望大家能够使用更先进的技术。

人人都爱恨 Spring Cloud

看起来，在 Java 世界中，Spring Cloud 貌似是微服务的最优解了，甚至在很多同学的简历上，Spring Cloud 几乎可以和微服务划等号了，不止一次的有人告诉我说：公司的技术栈不是 Java，所以搞不了微服务很难受，并不是我没有学习精神和冒险精神云云。很遗憾，对于软件架构来说，跟可没有规定编程语言，设计模式不是也出了很多版本吗？归根结底还是 Spring Cloud 的全家桶策略更吸引人，什么事儿都不如加上几个 jar 就能拥有的神奇次时代架构更有吸引力。

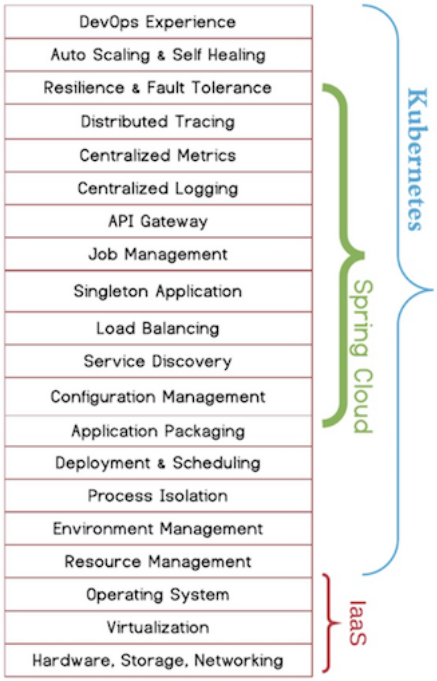
不可否认，我在学习 Spring Cloud 的时候也惊叹其完整性，几乎常见的微服务需求都有足够完整的解决方案，而大多数方案是做在应用层，具有良好的适配性，比如 eurake 的注册发现、zuul 网关与路由、config service、hystrix circuit breaker 等等，通过统一的编程范式（基于 annotation 的注入与配置），足够丰富的功能选择（常用功能甚至都有两种选择），以及较好的集成方式。前有 Netflix 的成功经历，后随着微服务的浪潮，再加上足够庞大的 Java 社区，可以说是王道中的王道。但并非 Spring Cloud 没有弱点，反倒这些功能设计与随后的容器化浪潮产生了分歧，至今融合 Spring Cloud 与 Kubernetes 都是热门话题，这里我们展开说说它的不足或者限制（limitation）。

侵入性与语言绑架

这可能是最大的问题，基本只能使用 Java 作为研发语言，这一点在国内也备受争议，因为不论是作为架构师还是入门的程序员，都需要尝试新的技术栈来进行储备或是采用新的功能，而且比如使用自制的 client 去实现 ribbon 的负载均衡也是很难的，但是如果不用 Java，做到这一点也很难，不是说 Java 语言不够优秀，而是我们对未来应该有更多的选择，对于一个技术公司来说编程语言应该不会成为限制，试问这个时代谁不想学习一点 golang 或者 rust 或者 scala 呢？其他服务比如 SSO、Config Service 也过于整体，如果想进行某项适配，则必须进行大量的修改（还好是开源的）。我们很担心这种情况都会随着框架的老去而面临推到重来的境界，Ruby on Rails 可能就是前车之鉴吧。侵入性是另一个问题，还记得我们在讨论软件架构时所提倡的实践规则吗？尽量不要让顶层设计依赖底层的框架或者某种细节，但是满屏幕的 annotation 与 jar 的直接引用，无疑告诉我们想去掉它们还是非常难的。

云原生的融合问题

对于云原生，不论是 CNCF 还是 Pivotal （VMWare）都在强调容器化、微服务、面向云环境等，CNCF 围绕 Kubernetes 开始发展壮大，也随着这种先进的容器编排技术的流行人们渐渐发现它和 Spring Cloud 在功能上还是存在很多重叠，虽然 k8s 与 IaaS 没有重叠，但是现在还有多少厂商再推纯 IaaS 呢？既然有功能重叠，就有取舍，考虑到 Spring Cloud 的全家桶属性，这个分歧处理一直都不能很好的解决。



集中式的资源

不论是 Config 、Eureka 都是聚合的单点，及时它们有集群的方式达到近乎 100% 的可靠性，但在逻辑架构上，所有的微服务都依赖它们，这些集中式的资源的耦合是非常强的，它们会一直存在在你的生产环境之中，直到最后一个使用它们的系统下线。我们在架构中需要避免使用共享的实例与资源，一个应用不会因为不能写日志而崩溃，也不应该因为本地没有 eureka 而无法启动。

畏惧平台绑定

诚然，在进程之内解决服务注册发现、负载均衡是很好的，它代表了最好了平台无关性，但平台的其他能力也很难享受的到了。绑定 k8s 貌似是个更好的选择，因为相对于 Spring Cloud 它更灵活，也能做到不会被基础的云平台绑定，但也更难以掌握与运维。当然我也不是认为 K8s 必须作为微服务的选择，作为容器的编排平台，它可以做更多的事情（比如跑数据库、中间件等），运行微服务应用只是其中之一。

方法论的落地能力

2020年已经过了一半，从技术上来说，Serverless 已经进入成熟期，Kubernetes 也更加成熟，已经成为事实的标准。但是很多时候我们的方法论与架构设计是跟不上的技术发展的，很多同学可能还在经历每周的发布日，很多同学还没办法改进团队内老旧的技术，很多同学的 Jenkins 还是停留在打包的阶段，很多同学机器上还是没有安装 docker，很多时候并不是框架或者平台的问题，而是方法论还停留在过去。

保持演进

应用程序也应该践行开闭原则，对扩展开放使得我们在未来有更多的选择。**微服务是一个很好的机会能让我们真正的演进架构而不需要付出过多的代价，当我们需要组件化系统时，组件的关键特性正是可独立替换或升级，我们可以不影响其他部分去进行替换和重构，这样的成本是显然低于抛弃旧的巨型框架而重写的**。有着正确的态度和工具，我们可以更快、更频繁的控制变更，我们可以激进的选择新的技术栈，也可以合并两个耦合过紧的服务，随着服务的不断聚合、抽出，你会发现系统的逻辑架构会越来越清楚，再进行修改就会信心倍增了。我们可以针对每个服务使用不同的存储技术，我们可以使用 OSS 处理文件，而不是继续往 Oracle 里面塞图片和视频。

Istio 的解法与问题，以及 Mesh 还缺少什么

> The old adage applies: if you can't design a modular system properly, what makes you think you can design a *distributed* modular system properly? But this is an unusual case where a new version of a product is going to be less modular than previous versions. So I say to the Istio team: you didn't do a good job of designing a distributed modular system, what makes you think you'll do a better job of designing a monolith modular system?

> 这个开幕雷击虽然槽点满满，但并没有降低社区对 Istio 的信心，反倒是渐渐发现这次的大改动使 Istio 变得有点好用了，可以在生产中采用而不需要付出太多代价了。当然，漂亮话永远好说。

2017 年的时候 Service Mesh 还是一个襁褓中的概念，现在已经成为了微服务领域的未来之选，但遗憾的是目前只有 Istio 足够成熟能代表这项技术，当然我也有幸实践过类似的 Sidecar 来进行反向代理、日志收集、性能监控、健康检查等功能，但是距离 Mesh 的愿景还是有大的差距。

今天并不想展开 Service Mesh 或者 Istio 的优势，进程之外的解决方案能够确保系统的灵活性，而流量控制、服务治理、端对端的传输安全、限流、发现注册等等，**我们希望工程师能够聚焦业务，实现架构的灵活性，聚焦真正的价值，而剩下的进行配置就好**。所以我想这也是 Serverless 被无限看好的原因，既然我们想 delegate 对基础设施的控制，那为什么还需要关心容器呢？Istio + k8s 的方案已经足够好，至少我们团队正在认真学习，在向客户提供最佳实践之前，我们依旧有很多问题需要解决，下面列出一些代表性的：

弹性与自恢复问题

使用系统度量、参数等触发弹性伸缩是常见的需求，这里我们是使用云监控？还是自己搭一套？我一直倾向数据与用途解耦，使用 pub sub 模式解决问题，比如 CPU 过高可能会触发多个行为：触发警报，触发弹性规则，展示在 dashboard 上。我们可以增加 pod 来分担服务的压力，或者因为某个 pod 异常退出后，启动新的 pod 来完成自恢复，这系列动作也是需要我们自己解决的。

监控与可观测性

日志、警报等可观测性的问题，这一方面的实践较多，唯一比较担心的是 ARMS 或者 Newrelic 这种 APM 功能目前没在目标平台上实践过，我们希望能够清晰的看到每个服务的实时性能，目前这一部分还缺乏考虑与设计。

限流与降级的实践

Istio 的流量控制能力是非常强大的，如何对服务采取降级、限流这种常见的治理操作，也是需要总结出实践经验的。避免串流错误（cascade failure）在微服务领域也很常见，也需要避免故障蔓延。

多种自动化部署

蓝绿、灰度、金丝雀，这些多样的部署方式也需要落地，我们可能会写一些 deployment util 之类的小脚本，部署的方式需要和 CICD 打通。一个部署工具，一个配置文件，一个 CICD 组成未来单个应用的部署方式。

ABAC 与 OPA

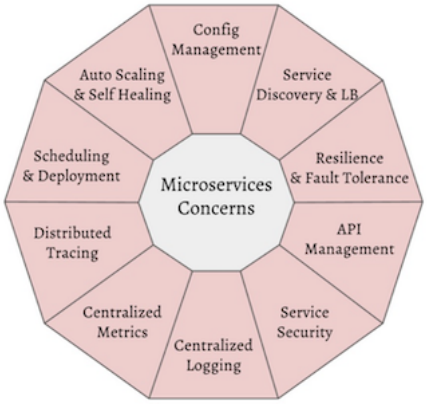
基于属性的权限控制以及 OPA 的可定制性我们是非常看好的，而且 sidecar 可以在进程之外解决权限验证问题，的确值得尝试，刚好也学习下 golang 用于定制 OPA。

Saga 与补偿事件

分布式事务是微服务实践中的大坑，我曾经也写过类似的文章，项目经验中更多的是将事务放在单一的服务内，再加上我自己也没机会写过真正的网店系统。补偿事件也是常用的最终一致性方案，总之放在一起验证。

高可用和混沌工程

基于 K8s 实现应用的高可用应该不难，容器的天生优势就是易于启动与管理，如果随机杀掉 pod 不会影响系统可用性，这就算是实现了类似于 SLB + ECS + ASG 的能力，真正危险的是其他 非业务型 pod，比如 istio 的各种 supervisor。



> 实践微服务，作为架构师所考虑的东西远远大于只是实现业务，但是一旦铺平道路，下来的研发与迭代将会更加顺利。

参考资料

- https://martinfowler.com/articles/microservices.html (https://martinfowler.com/articles/microservices.html)
- https://medium.com/swlh/stop-you-dont-need-microservices-dc732d70b3e0 (https://medium.com/swlh/stop-you-dont-need-microservices-dc732d70b3e0)
- https://medium.com/capital-one-tech/10-microservices-best-practices-for-the-optimal-architecture-design-capital-one-de16abf2a232 (https://medium.com/capital-one-tech/10-microservices-best-practices-for-the-optimal-architecture-design-capital-one-de16abf2a232)
- https://www.geekgirlauthority.com/what-you-need-to-know-from-sonys-the-road-to-the-ps5/ (https://www.geekgirlauthority.com/what-you-need-to-know-from-sonys-the-road-to-the-ps5/)
- https://dev.to/bosepchuk/why-i-cant-recommend-clean-architecture-by-robert-c-martin-ofd (https://dev.to/bosepchuk/why-i-cant-recommend-clean-architecture-by-robert-c-martin-ofd)
- https://istio.io/latest/blog/2020/tradewinds-2020/ (https://istio.io/latest/blog/2020/tradewinds-2020/)
- https://www.openpolicyagent.org/docs/latest/ (https://www.openpolicyagent.org/docs/latest/)

作者简介

张羽辰（同昭）阿里云交付专家，阿有着近十年研发经验，是一名软件工程师、架构师、咨询师，从 2016 年开始采用容器化、微服务、Serverless 等技术进行云时代的应用开发。同时也关注在分布式应用中的安全治理问题，整理《微服务安全手册》，对数据、应用、身份安全都有一定得研究。

> “阿里巴巴云原生 (http://mp.weixin.qq.com/s?__biz=MzUzNzYxNjAzMg==&mid=2247492001&idx=2&sn=8026eda2c508ba8955dfe4220ebe4a5a&chksm=fae6ea6ecd916378addd18d07774d9bb21ea999f798f70ed883cb3bf9b9af96aff76382b300b&token=442001322&lang=zh_CN#rd)关注微服务、Serverless、容器、Service Mesh 等技术领域、聚焦云原生流行技术趋势、云原生大规模的落地实践，做最懂云原生开发者的公众号。”

关注微信公众号，加入社区

上一篇: 阿里云李响荣获 2020 中国开源杰出贡献人物奖，我们找他聊了聊开源和云原生
(https://www.kubernetes.org.cn/8075.html)

下一篇: SpringCloud 应用在 Kubernetes 上的最佳实践 — 部署篇(工具部署)
(https://www.kubernetes.org.cn/8078.html)

标签： Kubernetes (https://www.kubernetes.org.cn/tags/kubernetes)

云原生 (https://www.kubernetes.org.cn/tags/%e4%ba%91%e5%8e%9f%e7%94%9f)

微服务 (https://www.kubernetes.org.cn/tags/%e5%be%ae%e6%9c%8d%e5%8a%a1)

架构 (https://www.kubernetes.org.cn/tags/%e6%9e%b6%e6%9e%84)

相关推荐

- Porter 进入 CNCF 云原生全景图，新版本即将发布！ (https://www.kubernetes.org.cn/8118.html)
- 掌门教育微服务体系 Solar | 阿里巴巴 Nacos 企业级落地上篇 (https://www.kubernetes.org.cn/8117.html)
- SpringCloud 应用在 Kubernetes 上的最佳实践 —— 开发篇 (https://www.kubernetes.org.cn/7958.html)
- 进击的 Kubernetes 调度系统（二）：支持批任务的 Coscheduling/Gang scheduling (https://www.kubernetes.org.cn/8034.html)
- SpringCloud 应用在 Kubernetes 上的最佳实践 — 部署篇（开发部署） (https://www.kubernetes.org.cn/8049.html)
- SpringCloud 应用在 Kubernetes 上的最佳实践 — 部署篇(工具部署) (https://www.kubernetes.org.cn/8078.html)
- 云原生五大趋势预测，K8s 安卓化位列其一 (https://www.kubernetes.org.cn/7995.html)
- 阿里云李响荣获 2020 中国开源杰出贡献人物奖，我们找他聊了聊开源和云原生 (https://www.kubernetes.org.cn/8075.html)

评论 抢沙发