进击的Serverless

架构师 1周前

架构师（JiaGouX）

我们都是架构师！
架构未来，你来不来？

忽如一夜春风来，千树万树梨花开，云原生的浪潮伴随着云计算的迅速发展仿佛一夜之间，迅速侵袭了技术的每个角落。每个人都在谈论云原生，谈论云原生对现有技术的变革。
Kubernetes已经成为容器编排的事实标准，Servicemesh正在被各大厂商争先恐后的落地实践，Serverless从一个一直以来虚无缥缈的概念，到如今，也被摆在台面，有隐隐约约崛起的势头。
只是没想到，在后端闷头搞容器化、上Kubernetes、尝试Servicemesh的时候，Serverless却先在前端火了起来。从今年的GMTC全球前端技术大会上，Serverless主题的火爆就可见一斑。笔者也看过一些网上流行的讲Serverless的文章，观点大同小异，实践几乎没有，误解与谬论满篇飞。本文倒无意挑起争论，只是试图从一个云计算研发的视角出发，聊一聊我们眼中的Serverless。

诉求：为什么需要Serverless

前端为什么想要上Serverless，其实也很好理解，node.js的普及、前端工程化以及BFF的兴起，越来越多的前端需要关心服务的构建、部署、运维，服务的日志、监控报警等等，严重拖累了前端的开发效率，让前端花很多时间在服务器上排查问题，无疑是痛苦而低效的。
对于前端来说，最原始的诉求是，我不愿意管服务器等底层资源，哪台节点宕机了，麻烦不用通知我；流量太大了，服务需要扩容了，我也不想关心；我只需要写好代码，就可以自动部署到服务器上，代码有bug，能让我看日志和监控排查问题就行。
其实这也不单单是前端的梦想，很多后端或者数据类的研发，也有同样的需求。不过咋看很美好，但是仔细想想，有一些后端的业务很复杂，服务间调用关系以及各种特异化需求其实很难适用于Serverless，想完全不关心底层的服务器，有点困难。
所以，Serverless并非银弹，关键是看业务场景和需求，就算只有50%的业务适合，能解决这50%业务的问题，那也是了不起的成就。

解释：到底什么是Serverless

Faas和Baas又是什么?

了解Serverless的同学，或多或少都听过Faas，Faas即Function as a service，一般都称为函数计算。
作为开发人员，只需要写一个函数，就可以在例如AWS Lambda等各种函数计算平台上运行起来，真正实现了对服务器的无感知，同时可以对外快速暴露API接口，可以基于函数级别的自动扩缩容，可以监听各种事件进行触发。
而且Faas结合云平台的webIDE，如果webIDE设计的足够好，可以给我们带来云平台上更方便的开发体验，结合云上的各种工具和生态，未来会有更大的想象空间。
不过，显而易见，以函数为最小粒度，有一些局限性。
微服务是以功能职责为划分，拆分成一个一个专注于特定功能和需求的服务，为了解决微服务之间的网络调用和流量管理，引入了很多服务治理等相关的功能和组件，可以想象一下，如果把服务模块再拆分为函数的粒度，函数之间的调用关系无疑会爆炸，再思考一下，如何把老的服务改造成Faas形态，如何复用函数之间的逻辑，如何管理大量函数代码，这无疑对开发者带来了很多困扰。
所以，Faas不太适合一般后台长期运行的web服务型应用，真正适合的是那些数据计算、批处理等业务，这些业务逻辑比较单一，运行完可以停止，而且更适合Serverless中基于事件触发的特性，冷启动的延时也无所谓。
Faas的一个基本特征是无状态，那实际上的数据或者状态该如何存储呢，所以说到Faas一般都会提及Baas，即Backend as a service，不过类似的Xaas的名词太多了，Baas这个名词看着就像是有人为了强行补充Faas没有干的活儿而起的。因此有些人粗暴的总结Serverless = Faas + Baas，当然如果你要强行认为Serverless就是函数计算，那这个也没有问题。
不过，我们的观点是：Faas只是Serverless的一种特例。在这个世界上，除了Faas，还有更多的无状态工作负载适合以Serverless的形态去运行。

Serverless的特性

除了服务的粒度不一样之外，无状态工作负载和Faas一般都具有以下Serverless的特性：

- **1-step deploy**

既然是Serverless，开发者真正关心和面对的是代码层面，所以不管是函数还是一个代码工程，一键构建和部署是我们的终极期望。Kubernetes生态下有各种CI/CD解决方案，但是缺乏更加一键式的工具可以帮我们将代码（函数）迅速转变成部署的服务。所以，一个足够好用的本地client工具、一个完善而高效的CI/CD平台很重要。对于Faas，可以让用户便捷的将函数部署到Serverless平台，对于无状态负载，则可以根据用户需求暴露一些构建的自定义配置和流程。

- **Automatically**

在Kubernetes上一般服务实际的运行都或多或少的需要我们创建很多的Kubernetes资源，例如service、ingress等，而Serverless会做更多的自动化操作，以便更方便的提供服务。例如，Serverless平台会自动提供流量入口和路由，部署完成后可以迅速对外提供服务，同时提供类似蓝绿发布、灰度等流量管理等功能。

- **Auto-scale**

毫无疑问，Kubernetes也有HPA可以提供自动扩缩容。不过，HPA敢让服务副本数缩为0吗？当然不敢，试想一下，如果服务的副本数为0，相当于不再运行了，用户的流量如何导入呢，用户连服务的接口都调不通了，HPA更没有metric数据来感知去扩容服务了。HPA无法缩容为0，对于某些短运行的计算类服务来说，是无法接受的，因为这样就不能真正的做到无服务，不实际运行时不占资源不计费。

当然Serverless可以做到，让服务在没有请求时自动缩为0，在有流量的时候从0启动，或者流量增大时快速的扩容，迅速应对流量的变化。

不过，还有一个Serverless业界都很关注的点，就是服务从0扩容为多副本时启动的延时时间，一般称为冷启动的问题。如果冷启动时间太长，对于用户的第一次请求肯定有很大影响，业内也有很多大厂在做一些优化。但是如果不是直接面向用户流量的服务，例如我只想跑个数据处理算法，其实也不在乎这几百毫秒的启动延迟，如果是类似前端的web服务，恐怕大部分人还是宁愿空跑一个单副本的服务，也不愿意冒这个风险吧。

- **Eventing**

Serverless的另外一个特征是基于eventing事件进行触发，事件实际上是一个比较抽象的说法，很多东西都可以理解为事件。例如，用户的请求可以认为是一个事件，git的webhook可以认为是事件，kafka上有了消息可以理解为一个事件，包括Kubernetes的各种资源操作等等都是。所以，其实事件触发我们并不陌生，我们的平时开发和设计架构里经常都会有意无意的使用到事件触发的机制，只是太过平常，反而没有人去注意和抽象出这么一个理念。

现在大家都在倡导云原生，很多服务都是往云上迁移和部署，事件触发机制在云上可以有更多的扩展性和想象力。例如，我们的Serverless应用可以监听云上的中间件或者基础组件的事件，通过这些事件，触发特定的Serverless应用，从而打通云上的Paas服务，实现云上服务的一体化。

总结下来，虽然目前Serverless很火但我们更应该静下心来思考，为什么会有Serverless的诞生，Serverless最原始的需求和驱动力在哪？是Kubernetes不够好用还是Servicemesh不够友好？

Kubernetes被认为是下一代的分布式操作系统，操作系统上必然会运行各种各样千奇百怪的程序，有的需要直面系统内核，有的只是提供用户更好的UI，不过，有一类程序可以以更便捷的方式去编译、运行，而提供这一切的工具与平台就是Serverless。所以，Serverless其实只是一种云原生应用更为特殊的实现和表现方式，也有很多的应用并不适合以Serverless的方式去运行。无服务器固然是愿景，大量的封装和抽象让开发者无需感知很多东西，但这个宇宙运行的规律可能并非直白的线性系统，混沌和复杂性才是常态。如果有人告诉你，Serverless是所有应用的终极目标。

适合Serverless的场景

基于Serverless的特性，我们也可以推导出比较适合Serverless的应用都有哪些：

- 前端、小程序、爬虫等
- 事件触发或定时的批量数据处理
- 大数据、实时流处理、机器学习的场景
- 经常应对流量突发的推广活动等无状态服务
- 视频转码等处理服务

其实还有很多，不过需要指出的是，这些都能在我们常规的容器云平台上构建部署运行，只不过，有了Serverless更高层次的抽象和封装，我们可以更快的开发构建部署，服务可以有更好的运行姿态，从而一步步接近我们想象中的那个只用写代码，不关心服务器的美好愿景。

Serverless现状与局限

作为Faas鼻祖的AWS Lambda其实早就已经推出了，但最近的几年内其他云厂商才慢慢跟上推出函数计算服务。 相信国内的很多公司也在尝试Faas或者Serverless架构，不过可以猜测出大家或多或少都心存疑虑或者有不放心之感，不敢真正的放上自己的线上服务。

目前看来，虽然Serverless市面上都吹的很火，但实际落地的寥寥可数，星星点点的一些火花，还是难以形成燎原之势。Serverless这片土地十分辽阔，但是各大云厂商却都是在自己和自己过家家，至于其他人怎么玩的，就不怎么关心了。

所以，目前一个很大的问题就是我们在一家函数计算平台跑了自己的服务之后，基本上就和这个平台绑定了，特别是如果你还用到Eventing，由于都是基于平台内部特定的事件触发机制，迁移成本还是比较高的。

终极原因，目前还没有一个强势而大一统的框架和平台，可以让大家甘愿臣服。想当年，容器编排领域兴起，Kubernetes和Mesos大战，两边各自有人站队，云厂商也各自押宝，如今Kubernetes一统江湖，大家都默默的建立起了基于Kubernetes的容器云平台，于是围绕着Kubernetes的云原生生态蓬勃发展。由于没有统一的平台，针对Serverless目前还存在的一些局限，大家做的优化和改进也是各自为战，难以落地生根。

虽然如今已然是2019年了，但我们还是看到了一些希望和未来的苗头。CNCF云原生计算基金会的Landscape中专门对Serverless分出一页（https://landscape.cncf.io/format=Serverless），上面总结了Serverless相关的各种平台和框架，其中目前最为火热和最有前景的便是Knative了。

实现：Knative

Knative是谷歌开源的Serverless架构方案，旨在提供一套简单易用的Serverless平台，把Serverless 标准化。Knative不局限于Faas，而是期望能够运行所有的无状态工作负载。像其他绝大部分的Faas或者Serverless平台一样，Knative也是基于Kubernetes，不过，Knative还基于Istio或者Gloo网关等实现流量的分发和管理。

还在不久之前，Knative分为三个组件：

- Build
- Serving
- Eventing

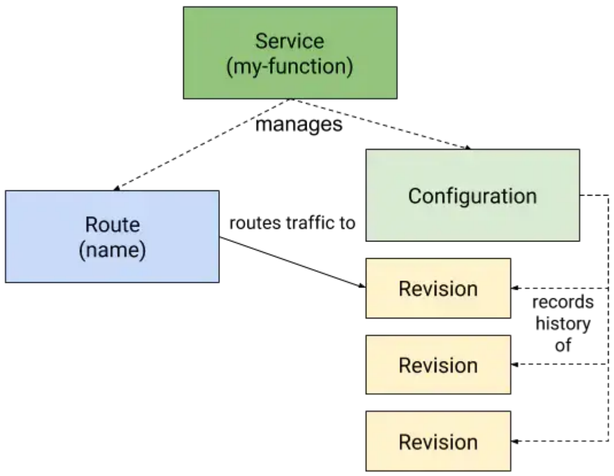
Build负责将代码转换成我们需要的容器镜像，Serving则是提供Serverless的运行方式，Eventing则致力于提供标准化的事件触发机制。

不过，现在Build模块已经被弃用，被由Build的设计思路而发起的Tekton项目替换（参考：《Kubernetes原生CI/CD工具：Tekton探秘与上手实践》）。当然你也可以使用其他合适的CI/CD工具替代。

Serving模块主要做的工作本质上就两个：

- **流量入口的自动创建和管理**

以基于istio为例，Knative自动创建istio的ingressgateway，服务的service等，将流量导入新部署的服务，而不需要手动的创建各种service，ingress暴露服务和流量入口。同时，将不同版本对应不同的deployment，可以方便的实现蓝绿、灰度等发布部署方式。如下图所示：



- **冷启动和自动扩缩容**

Knative有自身基于流量请求算法的metric自动扩缩容（KPA）的方式，也支持Kubernetes原生的HPA实现自动扩缩容。同时，在服务缩容为0之后，Serving会将服务流量路由到冷启动的组件，缓存请求，然后扩容服务，再将流量导入启动后的服务副本。

Knative Eventing则联合 CNCF Serverless WG制定一套事件格式规范并推广（https://github.com/cloudevents/spec/blob/master/spec.md#design-goals），只需要各个云厂商都按照这个规范，我们的Serverless服务就可以进行跨平台的事件触发，也不会被特定的云厂商绑定。

相关阅读：

- Knative 全链路流量机制探索与揭秘

架构师

我们都是架构师！

喜欢此内容的人还喜欢

别再用 kill -9 了，这才是微服务上下线的正确姿势！

架构师

小团队真的适合引入SpringCloud微服务吗?

架构师

微服务等于Spring Cloud？了解微服务架构和框架

架构师