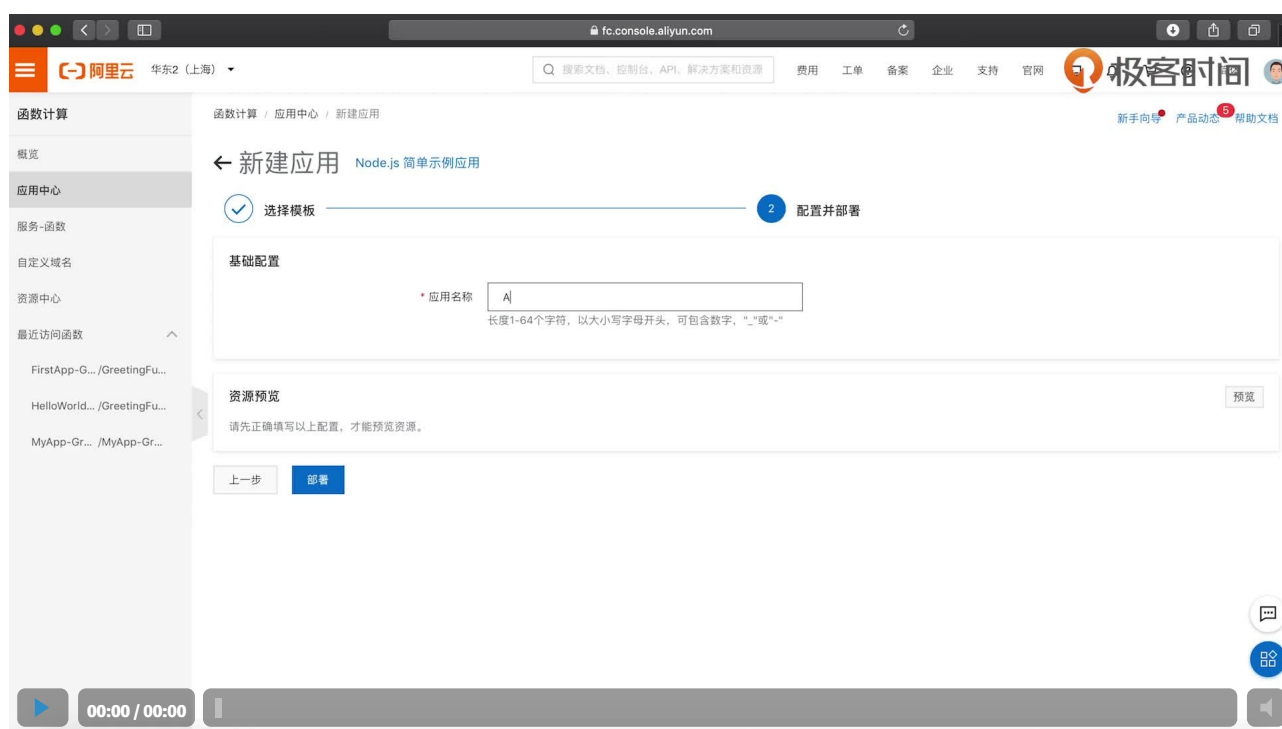


02-原理：通过一个案例，理解FaaS的运行逻辑

你好，我是秦粤。上一讲我们介绍了什么是Serverless，从概念的角度我们已经对Serverless有了一个深入的了解；那从应用角度来看，Serverless对于开发者究竟有什么魔力呢？这一讲，我准备通过快速部署纯FaaS的Serverless应用，给你讲一讲FaaS应用背后的运行原理。

为了让你更好地体验Serverless带来的变革，这节课我们以Serverless版本的"Hello World"实操例子进行展示。鉴于我的熟悉程度，我选择了阿里云，当然，你也可以选择你熟悉的云服务商（我在专栏的最后一课还会讲到如何解除云服务商的限制，混合使用多云运营商服务等等）。

另外，需要注意的是，如果你是跟着我一步步实操练习的，那么开通云服务可能会产生少量费用，遇到充值提示你要自行考虑一下。当然，如果你不着急体验，我觉得看我的视频演示也已经足够了。



我们从上面的演示也看到了，会用Serverless这个目标我觉得不难实现，但这不是我们这节课的终极目的。今天我就想带着你打开这个FaaS "Hello World"应用的引擎盖，来看看它内部到底是如何运行的。为什么要急着给你讲原理呢？因为如果你不理解原理的话，后面在应用Serverless化的时候就无从下手了。

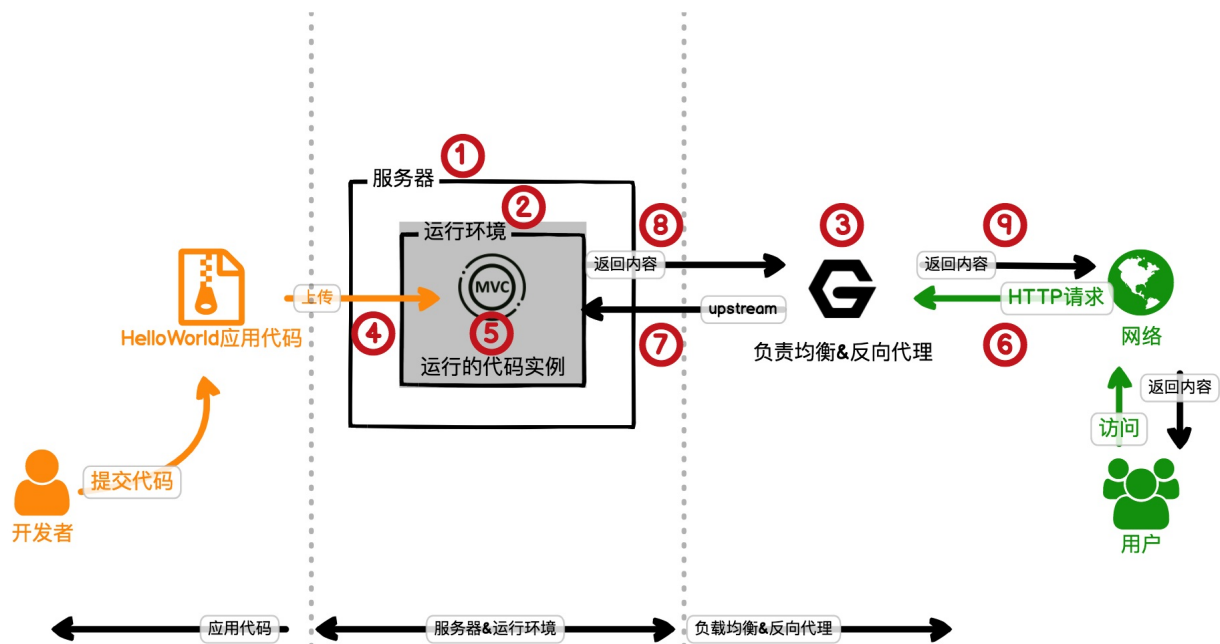
FaaS是怎么运行的？

现在大家都觉得Serverless是个新东西，是个新风口，刚才在演示的视频里你也能看到，它确实很方便。但你也别把它想得多复杂，运行应用的那套逻辑还没有变化，Serverless只是用技术手段帮我们屏蔽了复杂性，这点它和其他的云技术没有任何差别。

你可以想想，在Serverless出现之前，我们要部署这样一个"Hello World"应用得何等繁琐。首先为了运行我们的应用，我们要在服务端构建代码的运行环境：我们要购买虚拟机服务，初始化虚拟机运行环境，安装我们需要的应用运行环境，尽量和本地开发环境保持一致；紧接着为了让用户能够访问我们刚刚启动的应用，我们需要购买域名，用虚拟机IP注册域名；配置Nginx，启动Nginx；最后我们还需要上传应用代码，启动应用。

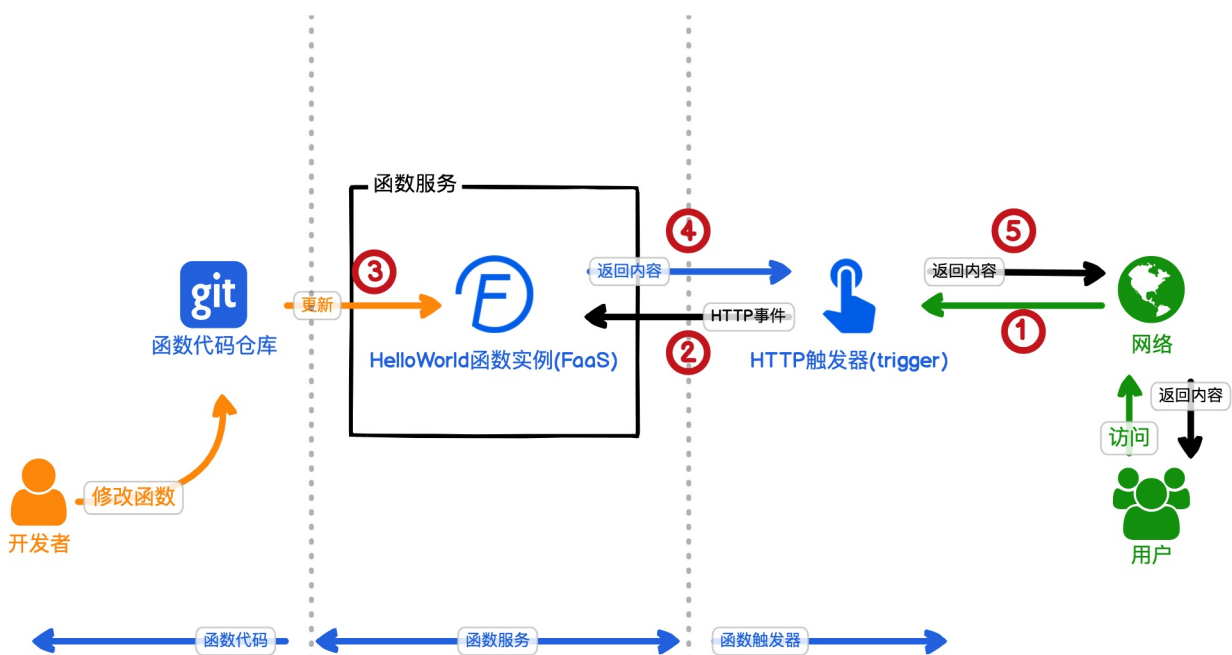
你可以闭上眼睛想想是不是我说的这样，当然，为了方便你理解，我还画了张图。前面5步都准备好了，用

户在第6步才能成功访问到我们的应用。



与上面传统流程形成鲜明对比的是，我们刚刚的Serverless部署只需要简单的3步，而且目前这样操作下来，没有产生任何费用。上一课我们讲过，**Serverless是对服务端运维体系的极端抽象**。注意，这句话里面有个关键词，“抽象”，我没有用“革新”“颠覆”之类的词语，也就是说，用户HTTP数据请求的全链路，并没有质的改变，Serverless只是将全链路的模型简化了。

具体来说，之前我们需要在服务端构建代码的运行环境，而FaaS应用将这一步抽象为函数服务；之前我们需要负载均衡和反向代理，而FaaS应用将这一步抽象为HTTP函数触发器；之前我们需要上传代码和启动应用，而FaaS应用将这一步抽象为函数代码。



触发器、函数服务……噢，是不是发现开始出现了一些陌生名词？不用着急，还是对照着上面这张图，我给你再串下"Hello World"这个纯FaaS应用的数据请求链条。理解了这些链条，你自然就理解了这几个新名词的背景了。

咱们先从图的右边开始看，图上我标注了次序。当用户第一次访问HTTP函数触发器时，函数触发器就会Hold住用户的HTTP请求，并产生一个HTTP Request事件通知函数服务。

紧接着函数服务就会检查有没有闲置的函数实例；如果没有函数实例，就去函数代码仓库中拉取你的代码；初始化并启动一个函数实例，执行这个函数，传入这个HTTP Request对象作为函数的参数，执行函数。

再进一步，函数执行的结果HTTP Response返回函数触发器，函数触发器再将结果返回给等待的用户客户端。

如果你还记得的话，我们刚刚的视频演示，你可以看到我们的纯FaaS "Hello World"应用例子中，默认创建了3个服务。

第一个"GreetingServiceGreetingFunctionhttpTrigger"函数触发器，函数触发器是所有请求的统一入口，当请求发生时，它会触发事件通知函数服务，并且等待函数服务执行返回后，将结果返回给等待的请求。

第二个"GreetingService"函数服务，当函数触发器通知的“事件”到来，它会查看当前有没有闲置的函数实例，如果有则调用函数实例处理；如果没有，则会创建函数实例，等实例创建完毕后，再调用函数实例处理事件。

第三个"GreetingServiceGreetingFunction"函数代码，“函数服务”在第一次实例化函数时，就会从这个代码仓库中拉取代码，并构建函数实例。

理解了FaaS应用调用链路，我想你可能会问，“真够复杂，折腾来折腾去，怎么感觉它的这套简化逻辑很像以前新浪的SAE或者Heroku那样的NoOps应用托管PaaS平台？”不知道你是不是有这样的疑问，反正我当时第一次接触Serverless时就有类似的疑问。

其实，FaaS与应用托管PaaS平台对比，**最大的区别在于资源利用率**，这也是FaaS最大的创新点。FaaS的应用实例可以缩容到0，而应用托管PaaS平台则至少要维持1台服务器或容器。

你注意看的话，在上面"Hello World"例子中，函数在第一次调用之前，实际的服务器占用为0。因为直到用户第一次HTTP数据请求过来时，函数服务才被HTTP事件触发，启动函数实例。也就是说没有用户请求时，函数服务没有任何的函数实例，也就不占用任何的服务器资源。而应用托管PaaS平台，创建应用实例的过程通常需要几十秒，为了保证你的服务可用性，必须一直维持着至少一台服务器运行你的应用实例。

打个比方的话，FaaS就有点像我们的声控灯，有人的时候它可以很快亮起来，没人的时候又可以关着。对比传统的需要人手动开关的灯，声控灯最大的优势肯定就是省电了。但，你想想，能省电的前提是有人的时候，声控灯能够找到比较好的方式快速亮起来。

FaaS也是这样，它优势背后的关键点是可以极速启动。那它是怎么做到的呢？要理解极速启动背后的逻辑，这里我就要引入冷启动的概念了。

FaaS为什么可以极速启动？

冷启动本来是PC上的概念，它是指关闭电源后，PC再启动仍然需要重新加载BIOS表，也就是从硬件驱动开始启动，因此启动速度很慢。

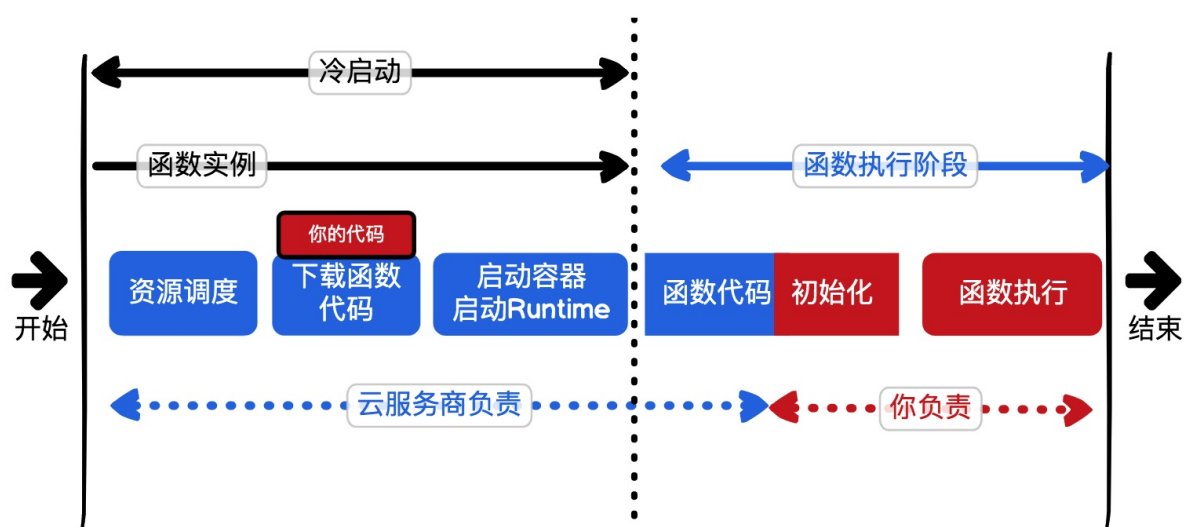
现在的云服务商，线上物理服务器断电重启几乎是不太可能的。**FaaS中的冷启动是指从调用函数开始到函**

数实例准备完成的整个过程。冷启动我们关注的是启动时间，启动时间越短，我们对资源的利用率就越高。现在的云服务商，基于不同的语言特性，冷启动平均耗时基本在100~700毫秒之间。得益于Google的JavaScript引擎Just In Time特性，Node.js在冷启动方面速度是最快的。

100~700毫秒的冷启动时间，我不知道你听到这个数据的时候是不是震惊了一下。

下面这张图是FaaS应用冷启动的过程。其中，蓝色部分是云服务商负责的，红色部分由你负责，而函数代码初始化，一人一半。也就是说蓝色部分在冷启动时候的耗时你不用关心，而红色部分就是你的函数耗时。至于资源调度是要做什么，你可以先忽略，我后面会提到。

例如从刚才演示视频的云服务控制台我们可以看到，"Hello World"的单次函数耗时是0.0125 CU-S，也就是说耗时12.5毫秒，实际我们抓数据包来看，除去建立连接的时间，我们整个HTTPS请求到完全返回结果需要100毫秒。我们负责的红色部分耗时是12.5毫秒，也就是说云服务商负责的蓝色部分耗时是87.5毫秒。



注意，FaaS服务从0开始，启动并执行完一个函数，只需要100毫秒。这也是为什么FaaS敢缩容到0的主要原因。通常我们打开一个网页有个关键指标，响应时间在1秒以内，都算优秀。这么一对比，100毫秒的启动时间，对于网页的秒开率影响真的极小。

而且可以肯定的是，云服务商还会不停地优化自己负责的部分，毕竟启动速度越快对资源的利用率就越高。例如冷启动过程中耗时比较长的是下载函数代码。所以一旦你更新代码，云服务商就会偷偷开始调度资源，下载你的代码构建函数实例的镜像。请求第一次访问时，云服务商就可以利用构建好的缓存镜像，直接跳过冷启动的下载函数代码步骤，从镜像启动容器，这个也叫**预热冷启动**。所以如果我们有些业务场景对响应时间比较敏感，我们就可以通过**预热冷启动或预留实例策略**[\[1\]](#)，加速或绕过冷启动时间。

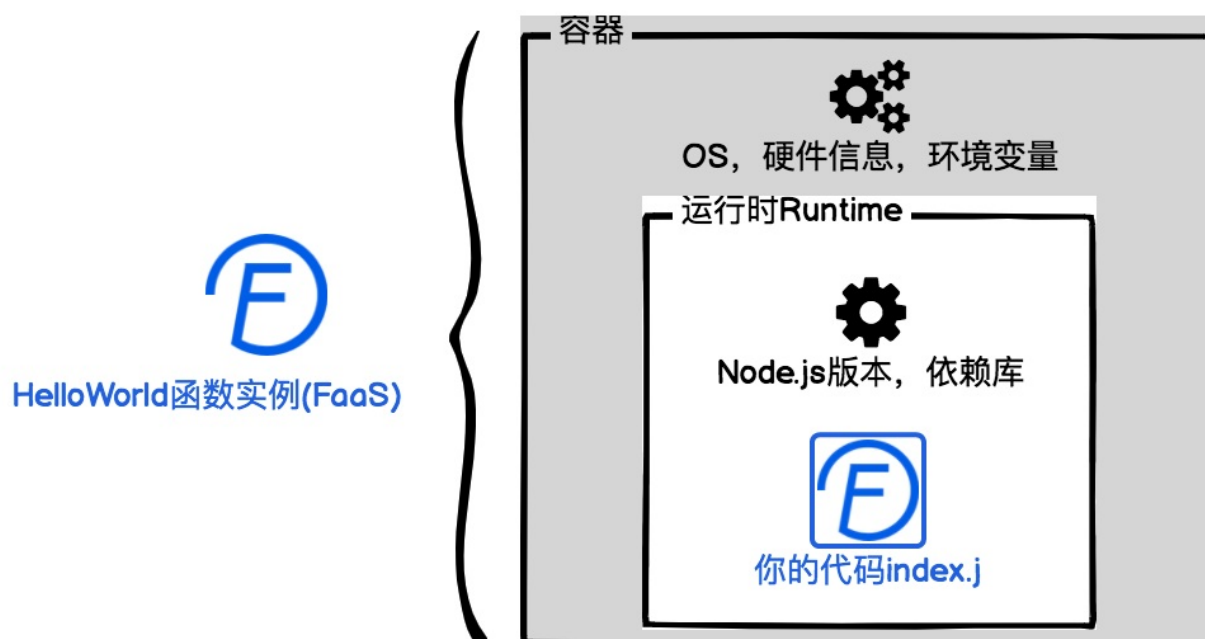
了解了冷启动的概念，我们再看看为什么FaaS可以极速启动，而应用托管平台PaaS不行？

首先应用托管平台PaaS为了适应用户的多样性，必须支持多语言兼容，还要提供传统后台服务，例如MySQL、Redis。

这也意味着，应用托管平台PaaS在初始化环境时，有大量依赖和多语言版本需要兼容，而且兼容多种用户的应用代码往往也会增加应用构建过程的时间。所以通常应用托管平台PaaS无法抽象出轻量的可复用的层级，只能选择服务器或容器方案，从操作系统层开始构建应用实例。

FaaS设计之初就牺牲了用户的可控性和应用场景，来简化代码模型，并且通过分层结构进一步提升资源的利用率。学到这里，我们得来看看隐藏在FaaS冷启动中最重要的革新技术：分层结构。

FaaS是怎么分层的？



你的FaaS实例执行时，就如上图所示，至少是3层结构：容器、运行时Runtime、具体函数代码。

容器你可以理解为操作系统OS。代码要运行，总需要和硬件打交道，容器就是模拟出内核和硬件信息，让你的代码和Runtime可以在里面运行。容器的信息包括：内存大小、OS版本、CPU信息、环境变量等等。目前的FaaS实现方案中，容器方案可能是Docker容器、VM虚拟机，甚至Sandbox沙盒环境。

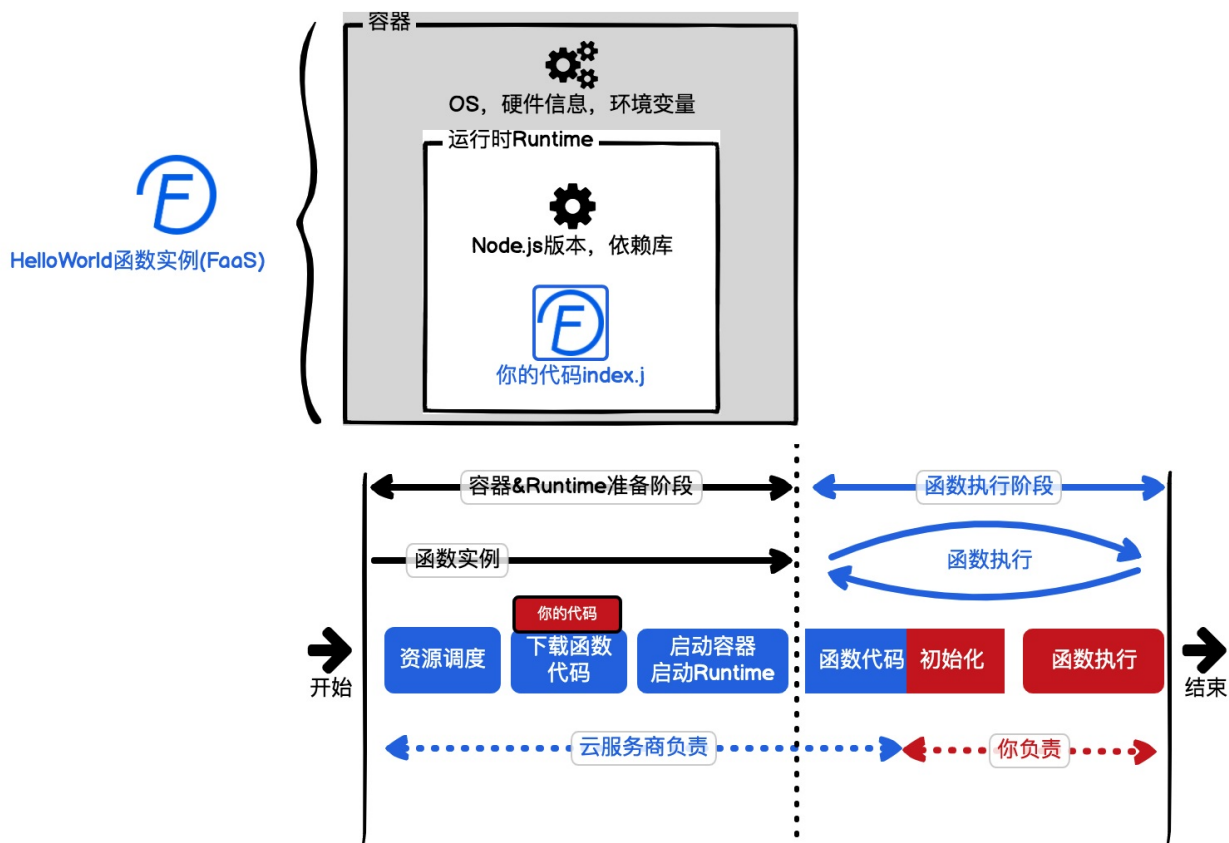
运行时Runtime [2]，就是你的函数执行时的上下文context。Runtime的信息包括：代码运行的语言和版本，例如Node.js v10, Python3.6；可调对象，例如aliyun SDK；系统信息，例如环境变量等等。

关于FaaS的3层结构，你可以这么想象：容器层就像是Windows操作系统；Runtime就像是Windows里面的播放器暴风影音；你的代码就像是放在U盘里的电影。

这样分层有什么好处呢？容器层适用性更广，云服务商可以预热大量的容器实例，将物理服务器的计算资源碎片化。Runtime的实例适用性较低，可以少量预热。容器和Runtime固定后，下载你的代码就可以执行了。通过分层，我们可以做到资源统筹优化，这样就能让你的代码快速低成本地被执行。

理解了分层，我们再回想一下FaaS分层对应冷启动的过程，其实你就不难理解云服务商负责的就是容器和Runtime的准备阶段了。而开发者自己负责的则是函数执行阶段。一旦容器&Runtime启动后，就会维持一段时间，这段时间内的这个函数实例就可以直接处理用户数据请求。当一段时间内没有用户请求事件发生（各个云服务商维持实例的时间和策略不同），则会销毁这个函数实例。

具体你可以看下下面这张图，以辅助你理解。



总结

这一讲，我带你体验了只需要三步就能快速部署纯FaaS的Web应用上线，我们也打开了FaaS引擎盖，介绍了FaaS的内部运行机制。现在我们就来总结一下这节课的关键点。

1. 纯FaaS应用调用链路由函数触发器、函数服务和函数代码三部分组成，它们分别替代了传统服务端运维的负载均衡&反向代理，服务器&应用运行环境，应用代码部署。
2. 对比传统应用托管PaaS平台，FaaS应用最大的不同就是，FaaS应用可以缩容到0，在事件到来时极速启动。Node.js的函数甚至可以做到100ms启动并执行。
3. FaaS在设计上牺牲了用户的可控性和应用场景，来简化代码模型，并且通过分层结构进一步提升资源的利用率，这也是为什么FaaS冷启动时间能这么短的主要原因。关于FaaS的3层结构，你可以这么想象：容器层就像是Windows操作系统；Runtime就像是Windows里面的播放器暴风影音；你的代码就像是放在U盘里的电影。

作业

最后，给你留个作业吧。我知道整个原理你听起来肯定还不是那么好理解。实践是检验真理的唯一标准，如果你有时间并且方便的话，可以试着自己动手Run一个FaaS的Hello World例子，然后思考其中的原理。

当然，如果今天这节课让你有所收获，也欢迎你把它分享给更多的朋友。

参考资料

[1] 预留实例介绍，https://help.aliyun.com/document_detail/138103.html

[2] Node.js Runtime介绍，https://help.aliyun.com/document_detail/58011.html

精选留言：

- 罗祥 2020-04-20 09:20:24

采用留言式学习法学习本专栏，小结：

- 1、Serverless 是对服务端运维体系的极端抽象。
- 2、关于 Faas 三层结构非常形象的描述，容器是操作系统，Runtime 是播放器，代码是小电影。
- 3、容器、Runtime 和函数实例的关系，一旦容器和 Runtime 启动后，就会维持一段时间，这段时间内函数实例可以直接处理用户数据请求。当一段时间内没有用户请求时，则会销毁函数实例，这样子可以达到缩容为 0。这里举手提个小问题，文章中提到的一段时间通常是多久呢？

今天把作业完成了再来留言 ~~

[2赞]

作者回复2020-04-20 11:29:55

阿里云是15分钟，AWS是5分钟，腾讯云还不清楚这个数据。知道的同学，可以帮忙回复一下。

- wc 2020-04-20 08:42:06

“一旦容器 & Runtime 启动后，就会维持一段时间，这段时间内……数据请求。当一段时间内没有用户请求事件发生（各个云服务商维持实例的时间和策略不同），则会销毁这个函数实例。”

请问老师，函数应用实例具体会在什么时候（或者说多少时间无访问后）被缩容释放？各个主流云平台具体是多少？ [2赞]

作者回复2020-04-20 11:24:41

阿里云是15分钟无请求。AWS是5分钟。腾讯云不太清楚。

- 唔多志 2020-04-20 00:32:43

runtime 是一个进程吗？ [2赞]

作者回复2020-04-20 11:23:57

runtime是进程的上下文，node.js的话你可以理解为就是require加载进内容的依赖模块。

- 炒鸡辣鸡 2020-04-20 21:42:07

serverless是基于rpc实现的吗？对请求链路的抽象是不是相当于隐藏了rpc调用过程中的command处理，最后直接返回结果？

- liuyu5337 2020-04-20 20:43:55

复杂应用场景适合吗？比如要用到连接池 数据库 redis Kafka等中间件，这些中间件都是一个函数实例吗？

作者回复2020-04-20 22:22:04

现在有很多人尝试用FaaS去实现复杂场景，云服务商也还在发展，所以会遇到很多FaaS的限制。有些场景定制Runtime，也无法处理。所以复杂场景，我建议还是采用CaaS方案。

- 一步 2020-04-20 14:15:11

老师有个疑问，一个函数实例就会启动一个 runtime 吗？会不会出现一个 runtime 运行多个函数实例？每个函数实例都初始化一个runtime 会不会浪费资源？

还有在新建函数的时候都会有个函数运行内存，这个是一个runtime 限制的内存吗？如果超过这个内存限制会怎么样的？

作者回复2020-04-20 14:50:58

好问题，说明你在认真学习。

每个函数实例对应一个runtime。runtime运行多个函数实例是可以的，就是常驻进程模型，我们下节课会讲到。每个函数初始化一个runtime不会浪费资源，如果函数执行完就销毁，就是用完即毁型，我们下节课会讲到。

函数运行的内存，是容器层的限制，不是runtime。如果内存超过了内存限制，会报内存溢出的错误。

- 郭嵩阳 2020-04-20 09:18:35

老师能不能说一下如果是java服务的流程应该是怎样启动的，还有就是FaaS如果调用的是java的服务 这个java的微服务是否应该是先部署的

作者回复2020-04-20 11:36:19

Java FaaS的启动流程是一样的，拉取你的Java代码构建docker镜像。调用Java的时候则从镜像启动。不过有些云厂商可能会用JVM黑科技，加速。Java做微服务，往往受限FaaS的runtime，适用FaaS的场景比较少。因此微服务还是推荐CaaS。我课程后面的专栏会讲到后端应用BaaS化，介绍FaaS的runtime限制如何通过CaaS打破。