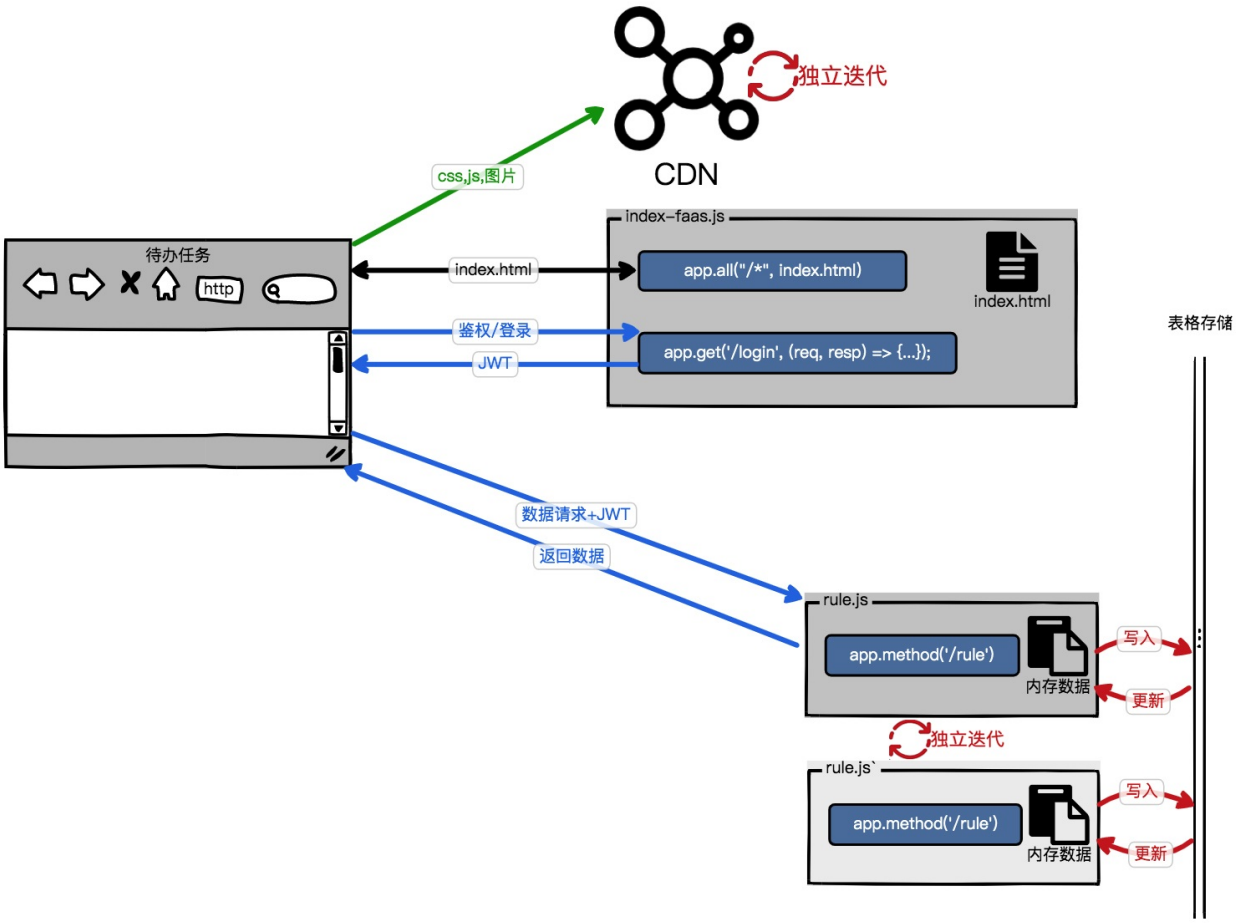
07-后端BaaS化（下）：ContainerServerless

你好，我是秦粤。上节课，我重点给你讲了业务逻辑的拆和合，拆的话可以借助DDD的方法论，也可以用动态网络的思想让微服务自然演进；合的话，我们可以用代码编排，也可以用事件流来驱动。另外，我们还了解了微服务拆解后会带来的安全信任问题，这可以通过微服务的跨域认证JWT方案解决。我们还了解了后端应用要支持快速迭代或发布，可以参考微服务搭建灰度发布流水线，也就是发布管道。其实我们在使用FaaS过程中遇到的很多问题，都可以借助或参考微服务的解决方案。

现在我们再回顾一下BaaS化后的“待办任务”Web服务，我们已经将后端拆解为用户微服务和待办任务微服务，前端用户访问我们的FaaS服务，登录后获取到JWT，通过数据接口+JWT安全地访问我们的微服务。而且我们的FaaS和微服务都具备了快速迭代的能力。



到这里，我要指出我之前rule-faas.js的一个Bug，如果你之前亲自动手做过实验的话，估计也有发现。这个Bug的直接表现是用户初次请求数据时，如果触发了冷启动，返回的待办任务列表就会为空。

究其原因，是因为冷启动时连接数据库会有延时，这直接导致了第一个请求返回的待办任务列表还未同步到消息队列的数据。要解决这个bug，我们可以用之前讲过的预热FaaS或预留实例的方式，但你也要知道，FaaS函数扩容时，新启动的函数副本也会出现同样的问题。

我前面卖了很多关子，其实FaaS在设计时已经考虑到这个问题了，所以在FaaS的“函数配置”里，都会提供一项“函数初始化入口”的选项。但你会发现，它同时也会让你配置初始化时间，最少1秒。还记得我们在[第 2 课]，讲函数执行阶段的一人一半的函数代码初始化时间吗？没错，就是那个时间。

当你配置了“函数初始化入口”，每次启动FaaS实例时，系统都会先等待初始化函数执行，而且在函数初始化时间结束后，才会继续执行函数。而从目前平台的配置来看，初始化时至少也需要1秒的时间（你去云

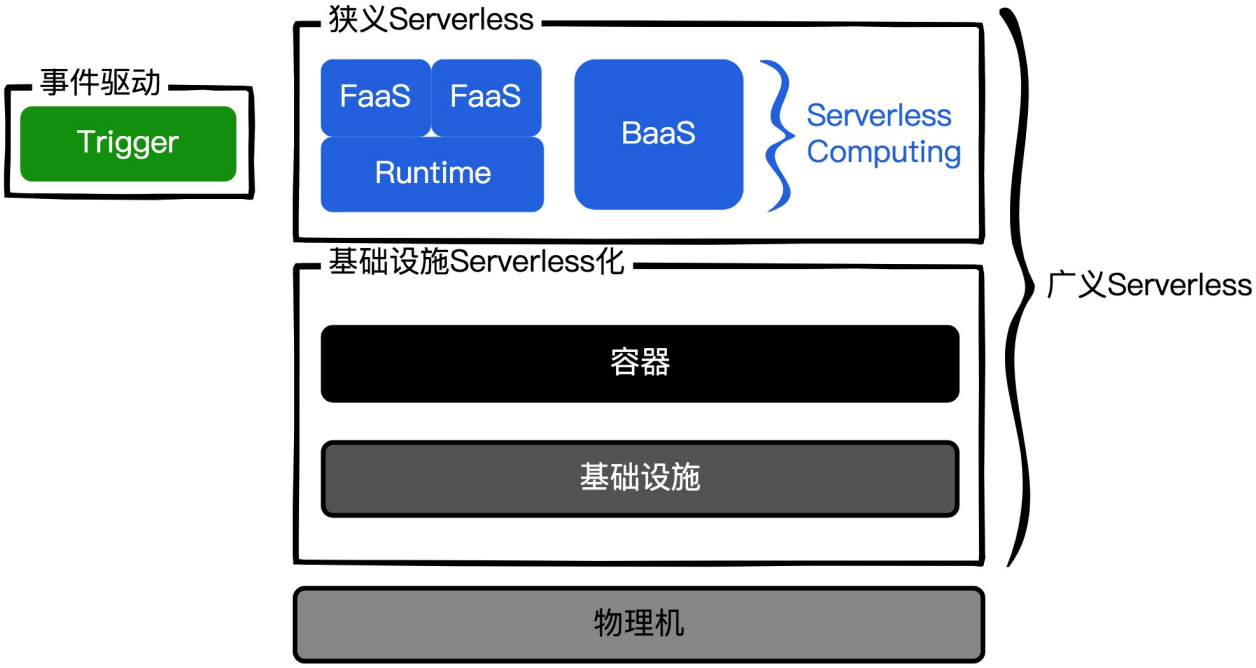
服务商后台就能看到）。

对很多事件触发的应用场景，FaaS增加几秒的初始化时间，影响并不大。但对很多后端应用则不然，尤其是Java等语言，如果软件包比较大，启动和初始化的时间会更长。

要缩短或绕过初始化的时间，我们要尽可能地利用Runtime里面给我们提供的内置能力，例如我们BaaS化一直提倡使用服务编排和HTTP接口，就是因为云服务商SDK和HTTP协议，所有语言的Runtime里都内置了。

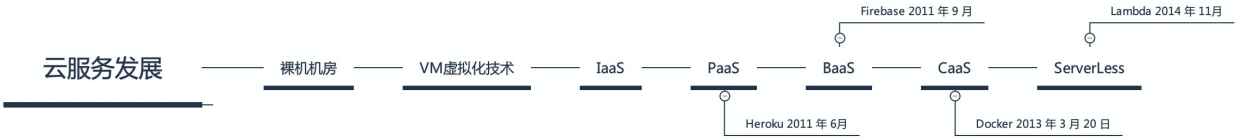
那什么是Runtime呢？Runtime其实就是运行我们代码所需要的函数库和二进制包。FaaS中的Runtime是云服务商预先设计好的，会放一些常用的函数库和二进制包进去，相当于是平台的能力。

但是当我们后端应用BaaS化后，想采用FaaS方案部署的话则会碰到Runtime这个拦路虎。FaaS虽然已经支持多数主流后端语言，但后端应用根据业务需求，要依赖特殊的函数库或二进制包，FaaS的官方Runtime就无法支持。而且像Java等语言，在代码包较大的情况下，FaaS启动速度很慢，也不适合。例如Node.js的Runtime，其实也包括我们自己安装的NPM包，所以相当于我们可以部分定制。但是你也注意到了，我们是整个目录包括node_modules 一起上传的，也就是说涉及编译的NPM包是无法跨操作系统生效的。这种场景下FaaS的Runtime不可控就会成为我们难以绕过的问题了。当我们遇到FaaS无法解决的场景，我们就可以考虑下降一层，使用FaaS的底层支撑技术Docker容器了。



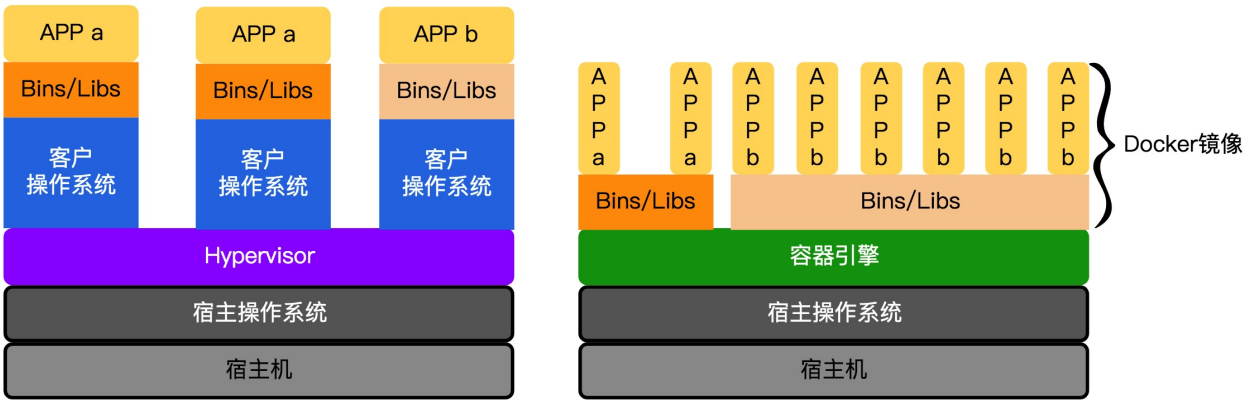
还记得我们[第 1 课] 中的广义Serverless的图吗？基础设施中的容器，一般情况下指的就是Docker容器。Docker 这个技术你肯定或多或少已经用过了，使用它们可以将应用代码和代码依赖的Runtime一起打包成一个Docker镜像。这个Docker镜像，可以在云上、自己的笔记本电脑、同事的电脑上无畅运行，并且完全不用担心环境依赖的问题，因为我们应用的依赖也打包在一起了。

到这里，我们又引入了Docker的概念，Docker出现之后，CaaS（Container as a Service）很快也就流行起来了。为了帮助你理解IaaS、PaaS、BaaS、CaaS这几者的关系，我画了张图，希望能从云服务发展的角度，帮你梳理下脉络。



图片很好理解，我就不解释了。不过这里要解决一个你可能会有的疑惑：为什么BaaS的出现，比Serverless FaaS还要早三年？那是因为早期出现的BaaS其实是mBaaS（移动后端即服务），这概念当时也曾经火过一段时间，现在已经被Google收购的FireBase其实就是做的这个生意。mBaaS设计之初是专门为移动端提供后端服务的，例如用户管理、角色服务、文件存储服务等等。除了服务对象是移动端之外，跟我们说的BaaS概念很像。移动端可以将BaaS的所需鉴权算法放到客户端中，并随着客户端的版本定期更新秘钥。但前端却做不到，因此mBaaS只局限在移动端，没有火起来。直到FaaS的出现，才诞生了BaaS的使用场景，mBaaS也开始转型，支持更广范围的前端场景了。

VM是一种虚拟化技术，这个我们都知道，其实Docker也是一种虚拟化技术，只不过它是利用新版Linux内核提供Namespace、Cgroups和Union File System特性，模拟操作系统的运行环境。它跟虚拟机VM最大的区别在于，Docker是进程模型的。怎么理解这句话呢？我们需要画一张图。



从图中我们可以看出，虚拟机是在宿主机上启动一个虚拟机监视器HyperVisor管理控制虚拟机的。而虚拟机自身包含整个客户操作系统、函数库依赖和用户的应用，虚拟机操作系统之间相互都是隔离的。经典的HyperVisor就是VirtualBox[1]，你感兴趣的话可以下载体验一下。你也可以试想一下，如果我们我们一台虚拟机部署一个微服务，其实资源利用率是很低的。

容器实例则只包含函数库依赖和用户的应用，操作系统是依赖宿主机的用户态模拟的，也就是说容器之间是共享宿主机内核的。所以Docker更加轻量，启动速度更快，代码执行速度也更快。

Docker的容器呢，因为只包含函数库依赖和用户的应用，可以部署到任意Docker引擎上，而Docker引擎呢，可以安装在你的个人电脑、公司机房或者云上。通常我们容器移动时，是移动容器的硬盘快照，内存中的状态我们比较难复制，这个硬盘快照就是Docker镜像。我们给Docker的镜像打上标签，标签就像是镜像的唯一标识符URI，打上后它就可以到处移动了。这个也是Docker的核心概念：build、ship、run。

其实你会不会觉得，Docker的这个层级结构有些眼熟？是的，这正是我们[第 2 课] 中讲的FaaS分层。我们回想一下，我当时所说的操作系统容器就是Docker模拟的，Runtime其实就是Bins/Libs层。云服务商的冷启动加速，就是利用Docker镜像缓存加速。我想你也应该猜到了，其实很多云服务商FaaS和PaaS的底层技术就是容器即服务CaaS。

那么接下来，我们就体验一下Docker的便利吧。我们的“待办任务”Web服务，只要添加一个文件，就可以让它变成Docker镜像了。

Docker再探

这里我建议你，最好自己在电脑上安装体验一下Docker。在主流的操作系统数安装Docker都不难的，只需要下载安装包就可以了。

build: Dockerfile

构建是Docker最重要的一环。Docker镜像是硬盘快照，那我们就看看标准的Docker硬盘快照如何制作吧。下面就是我们在代码根目录下增加的文件，供你参考：

```
# FROM是指我们的镜像基于哪个镜像来构建，我们这里是基于jessie linux的Node.js镜像
FROM registry.cn-shanghai.aliyuncs.com/jike-serverless/nodejs
# ENV是设置环境变量
ENV HOME /home/myhome/myapp
# RUN是执行一段命令
RUN mkdir -p /home/myhome/myapp/public
# COPY是将当前目录下的内容拷贝到容器中
COPY . /home/myhome/myapp
COPY public /home/myhome/myapp/public/
COPY node_modules /home/myhome/myapp/node_modules/
# WORKDIR是设置进入容器后的工作目录
WORKDIR /home/myhome/myapp/
# ENTRYPOINT是容器启动后执行的脚本
ENTRYPOINT [ "node","index.js" ]
```

通常我们使用Docker前，先去Docker Hub上，找合适的基础镜像。看看基础镜像上都安装了哪些函数库或二进制包，再对比一下要运行自己的应用还缺少哪些函数库和二进制包。在基础镜像的层级上，加上自己的依赖。这样我们就构建好适合自己的镜像了。以后我们就能FROM自己的基础镜像，构建自己的应用了。

然后你在项目下执行：`docker build` 命令，就可以帮你构建Docker镜像了。

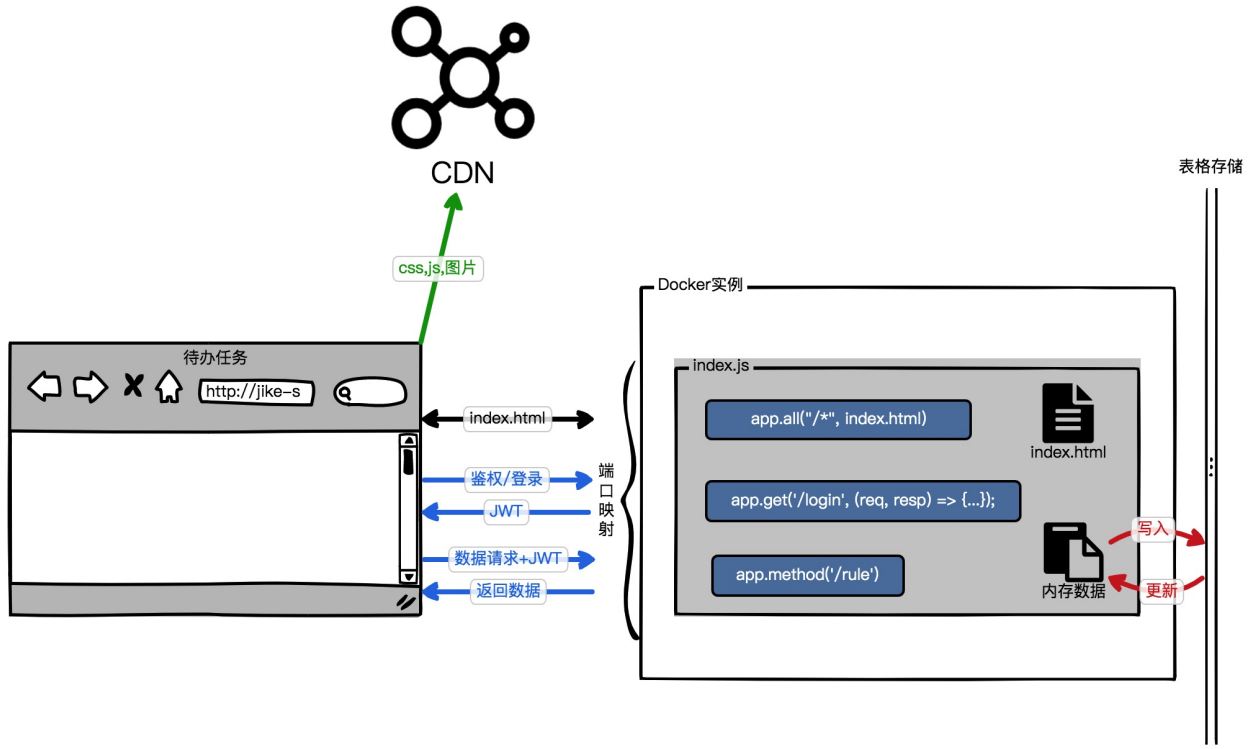
```
docker build --force-rm -t myapp -f Dockerfile .
```

构建好的镜像，我们可以通过 `docker run` 这个命令在本地调试。

```
docker run -d -p 3001:3001 -e TEST=abc  myapp:latest
```

然后我们通过浏览器访问http://127.0.0.1:3001，就可以看到我们刚刚构建的Docker内容了。你会发现，这不就是我们云上运行的版本吗？是的，既然FaaS是用CaaS技术实现的。我们当然也可以利用Docker实现我们的“待办任务”Web服务。

为了方便Docker例子的展示，这节课的项目代码index.js，包含了rule.js的逻辑。你会发现index.js中，我们启动的时候，不用关心初始化需要等待多少秒了，而是直到初始化完成后，才监听3001端口，而Node.js连接数据库的时间通常也只需要几十毫秒。



另外为了方面你观察和调试Docker容器实例，我这里给出一个登录Docker容器实例的命令。

```
docker exec -it 容器ID bash
```

ship: Docker Registry

本地调试完，我们再看看如何部署到云上。Docker官方的镜像仓库[2]速度太慢，现在的云服务都支持私有的容器镜像仓库[3]，上面构建Dockerfile里面的Node.js镜像，就是用我自己的私有容器仓库搭建的。

你可以登录云服务的容器镜像服务，他们都会告诉你如何 `Docker login` 到私有Registry，如何打标签，以及如何上传。

用我们的例子来说，大致会有下面的命令。

未登录过的话，要先登录Registry。

```
docker login --username=极客时间serverless registry.cn-shanghai.aliyuncs.com
```

根据容器镜像仓库的URI，打标签。镜像ID可以通过 `docker images` 查看。

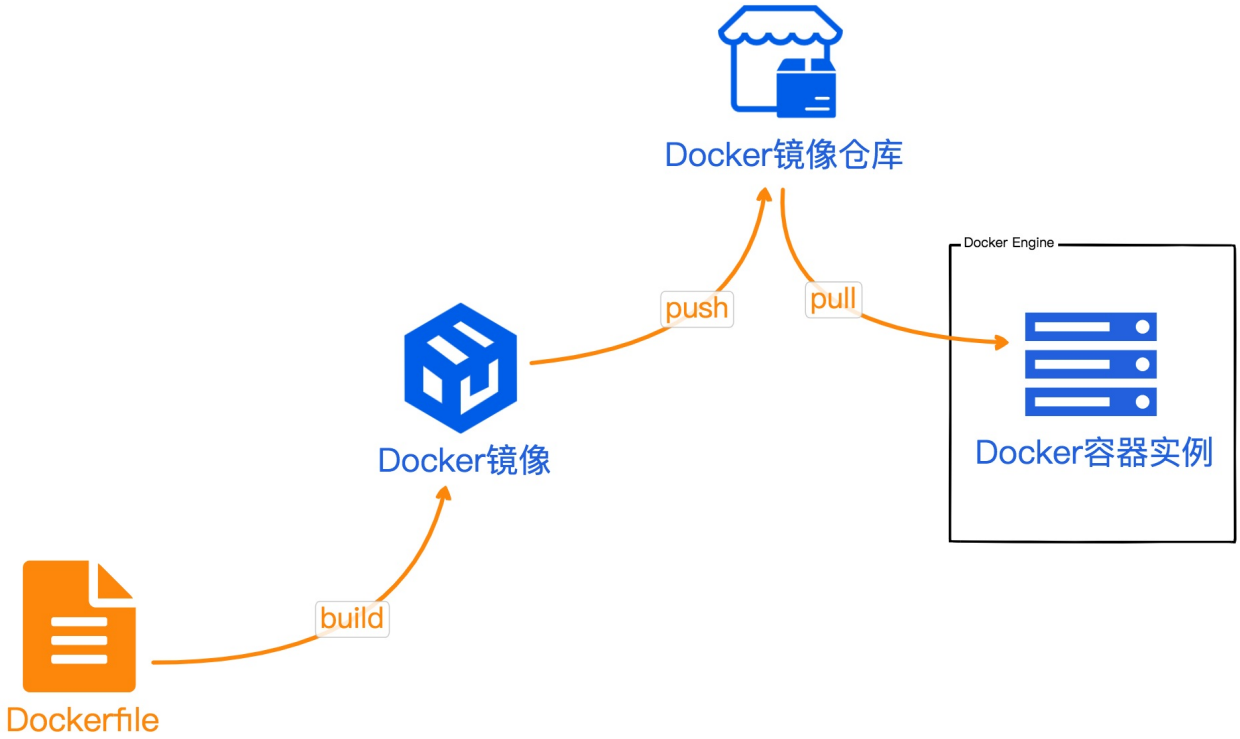
```
docker tag 你本地的镜像ID registry.cn-shanghai.aliyuncs.com/jike-serverless/todolist
```

上传镜像到私有的Registry仓库。

```
docker push registry.cn-shanghai.aliyuncs.com/jike-serverless/todolist
```

run: Docker Engine

运行Docker镜像就很简单了，云服务都支持容器服务CaaS。你只需要购买好容器服务，就可以从自己私有的容器镜像仓库Docker Registry中下载镜像，并运行了。你要是执行过上面的例子，那相信你也能体会到为什么FaaS的冷启动速度这么快了。不过需要提醒你一下，这里会产生费用。



另外，我们的专栏是讲Serverless，但是为了让你理解Serverless底层的实现，所以我们也讲到了Docker容器的部分内容。对于我们专栏来说，你不用尝试部署云上容器了。

恭喜你，获得DevOps奖章一枚！

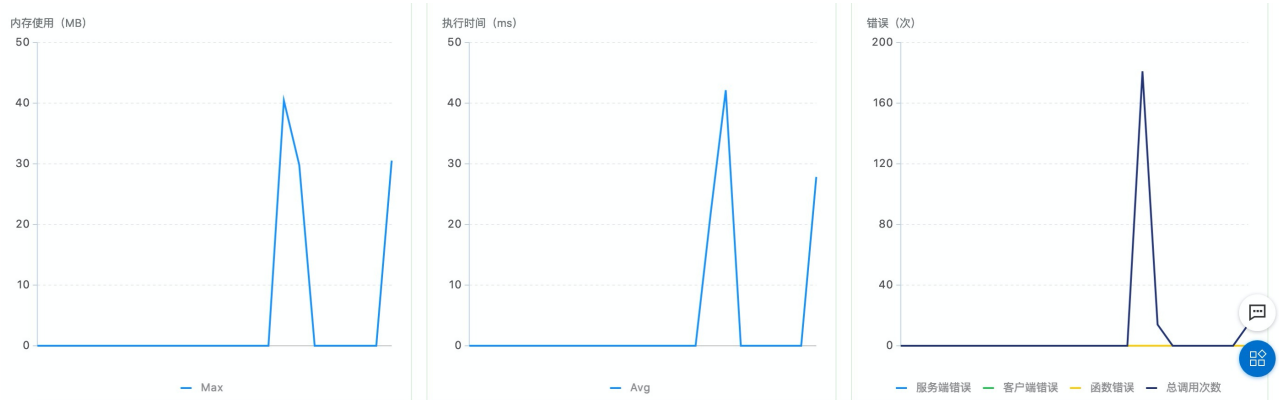
现在我们了解了容器，也知道了FaaS是构建在容器上的。我们的微服务和整个应用都可以部署在CaaS之上，容器对我们来说更加可控，因为我们可以通过Dockerfile安装我们需要的各种函数库依赖或者二进制文件。但这里还有一个问题，FaaS的扩缩容是怎么做到的呢？我们如果采用了容器，容器如何做到扩缩容呢？

FaaS与Docker的扩缩容

FaaS中的扩缩容，我们可以通过云控制台看到在FaaS函数配置中的“单实例并发度”。FaaS的做法比较简

单粗暴，需要我们告诉函数服务，我们的单个函数实例可以承载多少的并发量，如果事件触发并发量超出了这个并发度，则会自动扩容。扩容的数量N，就是这个事件触发量除以单机并发量取整。例如我们设定，我们rule-faas函数的单实例并发度为10，那么当用户并发量是11时就会扩容2个实例。如果用户并发量达到100，就会扩容出10个实例。

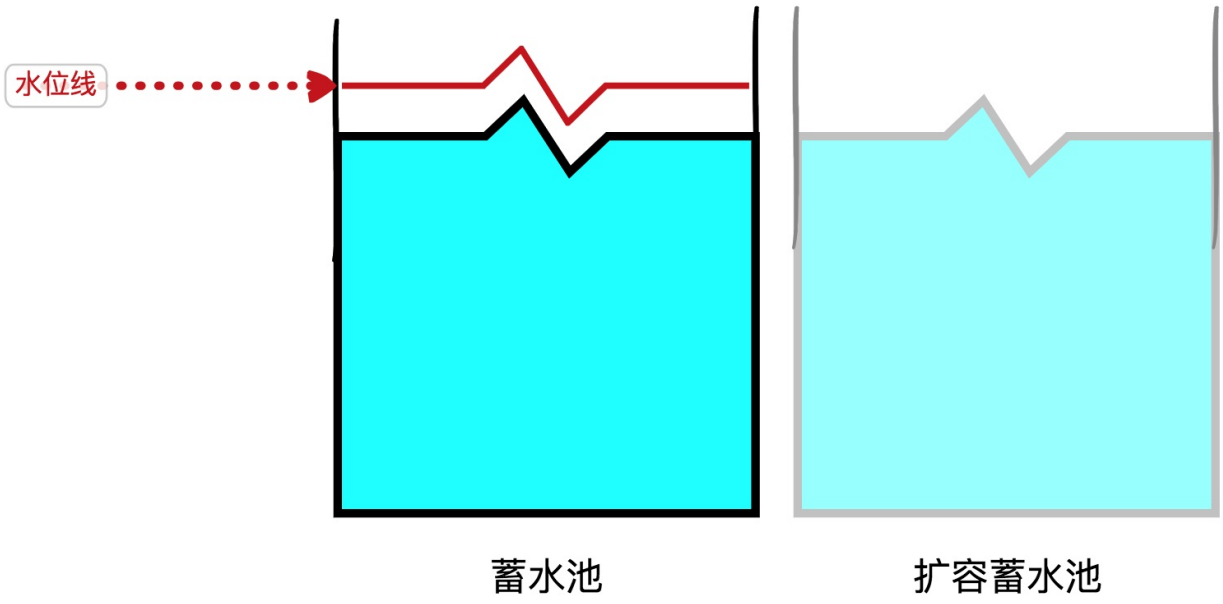
这种做法其实是比较机械式的，我们再看看FaaS的“函数指标”监控图。你有没有想到，我们其实可以利用实时监控，去控制扩缩容？例如当单个函数实例承受不了时，内存使用率会越来越高，它的执行时间也会越来越长。



是的，我们在使用Docker时，要考虑的就是：监控指标metrics以及扩容水位。

监控指标metrics就像FaaS“函数指标”监控图那样，是一系列需要我们关心的单个容器核心指标，包括CPU利用率、内存使用率、硬盘使用率、网络连接数等等。

我们将监控指标中的各项数值想象成蓄水，我们监控就像一个蓄水池，一旦某一项超过了蓄水池，水就会溢出。所以，我们要设定**水位**告警。我们在蓄水池里面画上刻度，当水位到这个刻度，我们就马上给这个蓄水池扩容。



Docker容器本身并不提供扩缩容机制，但我们要让Docker自动化扩缩容，就可以用监控指标和水位去设计扩缩容机制。我们需要实时监控容器状态，当容器状态的某一项数值过高时，我们就需要给容器扩容。FaaS的弹性做法很简洁，而Docker的扩缩容机制弹性更高、更加可控。但是Docker容器，通常需要我们至少保持1个容器实例在线。相信你也知道了FaaS的预留实例是怎么做到的了吧？

讲到这里，不知道你发现没有，我们BaaS化的三讲，已经默默地实现了微服务的10要素。这也是为什么我一直说BaaS化和微服务高度重叠。

我们再来回顾一下微服务10要素。其中，API、服务调用、服务发现，我们可以通过RESTful HTTP接口实现；日志、链路追踪，我们没有展开，但我们可以依赖云服务商提供的日志采集解决；鉴权我们可以用跨域认证JWT解决；发布管道，需要我们搭建流水线发布管道；容灾性、监控、扩缩容，我们可以通过实时监控和扩缩容解决。到这儿呢，我们的高铁车厢也终于完成了。

这节课的内容挺多的，需要你动手消化吸收一下。下节课我们再看看，如何利用K8s调度我们的Docker容器。

总结

BaaS化的内容，到今天，我们用了三节课讲完了，现在我们一起回顾一下。

我们讲后端应用BaaS化的问题，转换为后端应用NoOps微服务。所以我们先了解了微服务的10要素，并且看到微服务中如何通过消息队列将Stateful的节点，变成Stateless的。在微服务的拆解过程中，我们学习了微服务的拆解思想DDD和动态网络演进，以及拆解后带来的信任问题，需要我们加密身份认证。合并时，除了代码里面编排HTTP请求结果，我们还学习了事件流触发获取结果。为了让微服务能够快速迭代，我们还需要自己搭建流水线发布管道。

这节课我们通过FaaS和BaaS的初始化问题，向你介绍了FaaS和BaaS依赖的底层实现容器Docker。Docker也是一种虚拟化技术，它的核心理念是：build、ship、run。通过将我们的应用代码和应用依赖的函数库、二进制包，打包在一起的方式，解决应用开发环境和运行环境的一致性问题。我们可以借助Docker容器维持我们的应用实例，来解决初始化慢的问题。

我们还了解了FaaS的扩缩容逻辑是通过用户配置的“函数初始化入口”，以及固定的“初始化超时时间”配合“单实例并发度”来实现的。而我们在容器Docker，其实可以采用实时监控配合扩容水位，来做到更加弹性的扩缩容策略。

接下来我会通过K8s实践我们目前所学到的内容，我们的“待办任务”Web服务，将在我们本地搭建的CaaS环境和Serverless环境中开发和调试。

作业

首先，拉取我们lesson07的代码，为“待办任务”部署的rule-faas函数添加初始化入口。

这节课的作业呢，就是我们要在本地完全通过Docker容器，搭建起我们的“待办任务”Web服务。除了css和js静态资源是来自CDN，其它内容都将运行在Docker容器里。

相信你可以通过这个作业，体验到FaaS的底层CaaS的运行机制。

当然如果你有预算，也可以将Docker镜像上传到云服务商的Registry，在云上购买容器服务就可以部署你的Docker镜像，并在云上运行我们的“待办任务”Docker版本了。这样你就拥有了一个永不停机的Docker服务。

另外，我也希望你可以帮助我继续优化我们的课程作业代码。如果你有更好的建议，也可以通过Github的

MergeRequest告知我。

接下来就期待你的作业和建议了。如果今天的内容让你有所收获，也欢迎你把文章分享给身边的朋友，邀请他加入学习。

参考资料

[1] https://www.virtualbox.org/

[2] https://docs.docker.com/get-docker/

[3] https://hub.docker.com/

精选留言：

- 我来也 2020-05-02 13:52:54
  在完成课后作业时,将docker镜像部署到了本地,阿里云ECI,阿里云k8s,一切都很美好.
  但是部署到阿里云serverless k8s的时候,遇到了一个坑,给大家分享一下.

  现象: 无法打开todolist页面.
  但是控制面板上看pod和容器组状态正常,无重启记录.

  排查过程:
  1. 使用golang服务制作了一个镜像,端口号也使用3001,部署的服务可以正常访问.
  (说明操作流程和相关配置无问题)
  2. 在控制面板上看,对应的pod运行状态健康,遂在index.js底部添加日志.
  发现初始化的日志出来了,说明服务确实是跑起来了.
  3. 由于控制面板上无法执行kubectl exec -it 操作进入容器.
  顾开启了一个实例,安装了kubectl客户端,登录容器.
  [由于创建serverless k8s集群时未开通API Server 公网连接端点,所以只能在VPC网络中访问k8s API]
  4. 进入容器后,发现curl http://localhost:3001提示连接拒绝.
  使用netstat命令提示命令不存在.
  在容器内直接使用apt-get update ; apt-get install net-tools 由于网络问题,长时间安装不成功.
  5. 修改dockerfile,在制作镜像时执行安装操作.
  6. 重新推送镜像,部署服务,进入容器.
  发现3001端口果真未监听.
  但是netstat -antple中,显示连接了一个阿里云的otsIP地址处于SEND_SYNC阶段.怀疑是网络问题.
  结果ping baidu.com果真是ping不通.
  7. 尝试还原index.js中监听服务的代码
  直接使用: app.listen(PORT, () => console.log(`Example app listening at http://localhost:${PORT}`))
  不再放在client.getRow函数中.
  8. 重新推送镜像,部署服务,进入容器.
  发现可以正常访问todolist主页,只是没有数据.
  9. 猜测是 serverless k8s 默认无法访问公网,需要NAT网关才可以访问公网.
  提交工单,得到证实.
  需要在创建服务时,添加一个注解,在创建pod时绑定一个eip.
  照着工单中的方法,重新还原index.js后部署了一遍,果真可以正常访问todolist首页,且历史记录也有了.

  就一个简单的服务不可用,排查起来是相当的麻烦.还要不走弯路.

对[suke]同学留言中的一句话很赞同:
"如果想完全hold住serverless 真是的类似全栈这样的工程师来做"
[3赞]

作者回复2020-05-03 00:39:53
你领先太多了，我还没介绍到K8s呢。
我们使用很多技术时认知是有一个过程的，一个是理论认知，一个是实践认知。理论认知通过学习就好，实践认知则需要自己去动手碰撞出边界。
这点做得好的，不得不说AWS和Google云了。基本上理论认知和实践认知很接近。国内云由于各种受限，总有很多坑要踩。所以我更希望大家可以自己动手体验一下。可喜的是，国内云的进步也很快，很多大家发现的问题，都在跟进处理。
我也回复了suke同学的提问：我们试过单纯使用Serverless，在日常有合适的事件触发函数场景就可以了，不用hold住全部。但理解Serverless引擎盖下的机制，有助于以后可以我们深入这个领域。

- qinsi 2020-05-03 15:04:52
即便是用了docker，遇到需要编译的npm包，直接复制node_modules目录也可能会因为宿主系统和镜像系统的差异而导致问题吧

作者回复2020-05-03 20:21:32
是的，严谨的流程来说是copy时，应该忽略node_module，而重新执行npm install --production安装node_module依赖。
不过大部分的nodejs的库没有依赖底层编译库，除了少部分例如CSS SaaS之类的需要关注操作系统编译。所以目前FaaS，也是支持zip包上传，或者运行构建命令。

- suke 2020-05-02 01:30:28
老师我觉得，专栏看到这里，如果想完全hold住serverless 真是的类似全栈这样的工程师来做，如果只是让偏前端的工程师开发页面以及简单的业业务逻辑，在真正开发 以及调试的阶段，很难去完全cover ，但是如果把前端页面以外的工作交给后端工程师，实现起来一个是拆分的太细，想想本来一个很简单的列表查询 我不仅要写业务逻辑的查询 我还要把基础的数据查询接口封装到有状态的服务，开发、维护起来很麻烦，后期维护起来排查问题难度都很大，调用链条还变长，怎么看我都觉得serverless有点舍近求远，不知道老师觉得我这个想法有什么问题

作者回复2020-05-03 00:26:26
其实我是想通过这门课程也让你了解后端工程师的很多工作。我比较赞同的是解决问题工程师，而不是分前后端。

如果要分前后端，的确如果要hold住整个Serverless架构需要懂运维的全栈工程师。但我们实际使用Serverless协助我们日常的工作并不需要那么深入。你可以理解，你使用FaaS就像可以随时随地调用一个函数处理事件一样。

不过我觉得理解serverless内部的运行原理，对于你日后想深入学习研究，会大有帮助。而不是简单教你如何用Serverless搭建应用，这方面已经有很多课程了。
对于前端工程师来说，理想的状态应该是自己负责前端代码和数据编排接口BFF，也就是SFF。而后端工程师给你提供BaaS。
你的前端资源和数据编排接口一起发布。

- 我来也 2020-05-01 21:43:12
课后作业:

1. 由于镜像中有aliyunConfig,所以不建议将镜像公开.(即不要推送到默认的 https://hub.docker.com/)

2. 我是在template.yml文件中添加了一行`Initializer: index.initializer`实现了作业1:部署的 rule-faas 函数添加初始化入口

3. 使用阿里云ECI部署时,需要申请弹性公网EIP.
注意: 这里需要指定端口号3001访问服务, 默认的80端口是无法访问到服务的!!!
与本地调试一样,某些操作会触发容器重启.

4. 本地调试时,使用get方法访问`http://localhost:3001/api/rule`可以正常获取结果.
但是完成/删除(使用put/delete方法)时,会提示403"用户得到授权，但是访问是被禁止的。"
docker反馈的日志如下:
```
/home/myhome/myapp/node_modules/tablestore/lib/request.js:66
throw err;
^
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
at ServerResponse.setHeader (_http_outgoing.js:470:11)
at ServerResponse.header (/home/myhome/myapp/node_modules/express/lib/response.js:771:10)
at ServerResponse.send (/home/myhome/myapp/node_modules/express/lib/response.js:170:12)
at ServerResponse.json (/home/myhome/myapp/node_modules/express/lib/response.js:267:15)
at Response.<anonymous> (/home/myhome/myapp/index.js:151:16)
at Request.<anonymous> (/home/myhome/myapp/node_modules/tablestore/lib/request.js:162:18)
at Request.callListeners (/home/myhome/myapp/node_modules/tablestore/lib/sequential_executor.js:113:20)
at Request.emit (/home/myhome/myapp/node_modules/tablestore/lib/sequential_executor.js:81:10)
at Request.emit (/home/myhome/myapp/node_modules/tablestore/lib/request.js:189:14)
at Request.transition (/home/myhome/myapp/node_modules/tablestore/lib/request.js:57:10)
```
由于不太懂node.js,所以也不知道如何解决.
之前部署到阿里云FC时是没有遇到过该问题的.

作者回复2020-05-03 11:02:07
我刚刚看了一下，是因为JWT token的原因。这个分支加了JWT验证，如果不是浏览器访问请求中带JWT token过来，或者token校验失败，put/post/delete都会报403.

- 北冥神功 2020-05-01 15:14:58
我感觉谷歌的Cloud Run冷启动要比faas这种快，在腾讯部署的go服务冷启动要2s，cloud run只需要几百毫秒。cloud run是基于knative，不知道是不是因为是容器所以快?

作者回复2020-05-01 21:45:17
这个要看底层实现，FaaS的底层也可以是容器的。不过要云服务商通常都会根据自己的主流用户群体去优化启动速度，毕竟这也是给他们自己节省钱。

- 一步 2020-05-01 11:33:01
利用实时监控，去控制扩缩容,
向文中说的，云上有什么服务可以实时监控 一些指标进行扩缩容呢？ 现在基本上都是根据并发数来进行扩缩容的

作者回复2020-05-01 21:32:23
后面的课程会讲怎么搭建实时监控metrics，云上很多服务都会提供metrics数据面板。

- 一步 2020-05-01 11:27:31
  这个把服务 docker 化，不就没有办法使用 FaaS 服务了？ FaaS 有么有哪个地方可以结合 容器化服务一起使用的?

  作者回复2020-05-01 21:28:35
  docker化，可以使用FaaS。FaaS的HTTP加密触发器，就是需要你在应用中用算法去加密调用的。
  另外我目前讲的Docker化是BaaS。BaaS是个SFF提供RESTFul API的后端服务接口。