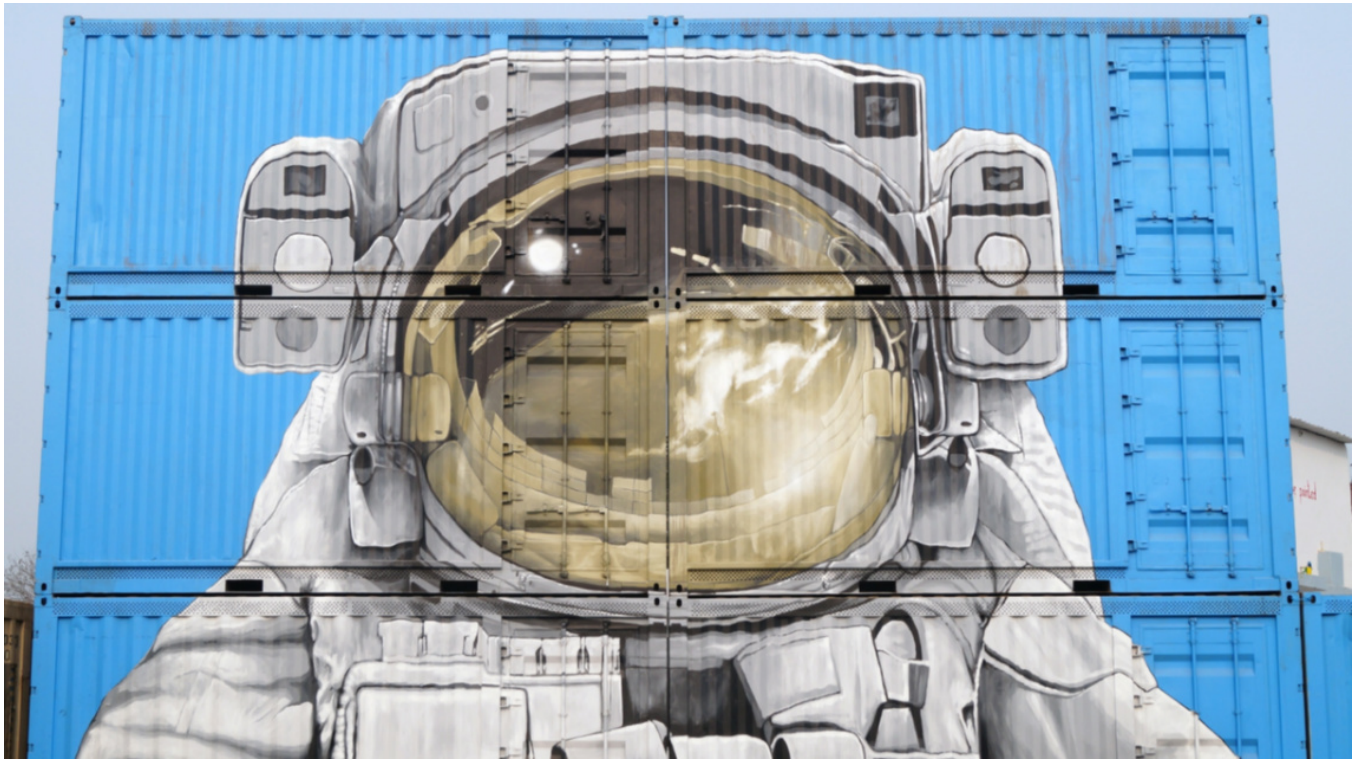
27 | 微服务容器化运维：容器调度和服务编排

2018-10-23 胡忠想



27 | 微服务容器化运维：容器调度和服务编排
朗读人：胡忠想　10′35″ | 4.85M

专栏上一期，我给你讲解了容器运维平台的两个关键组成：镜像仓库和资源调度。复习一下，镜像仓库解决的是 Docker 镜像存储和访问的问题，资源调度决定了 Docker 镜像可以分发到哪些机器上的问题。这两个问题解决后，你就该考虑如何在集群中创建容器，也就是容器如何调度的问题；容器创建后如何运作才能对外提供服务，也就是服务如何编排的问题。下面我们就一起看看容器调度和服务编排都是如何解决的。

容器调度

容器调度的问题，说的是现在集群里有一批可用的物理机或者虚拟机，当服务需要发布的时候，该选择哪些机器部署容器的问题。

比如集群里只有 10 台机器，并且已经有 5 台机器运行着其他容器，剩余 5 台机器空闲着，如果此时有一个服务要发布，但只需要 3 台机器就行了，这个时候可以靠运维人为的从 5 台空闲的机器中选取 3 台机器，然后把服务的 Docker 镜像下载下来，再启动 Docker 容器服务就算完成发布。但如果集群机器的规模扩大到几十台或者上百台时，要发布的服务也有几十个或者上

百个的时候，由于每个服务对容器的要求，以及每台机器上正在运行的容器情况变得很复杂，就不太可能靠人肉运维了。

这时就需要有专门的容器调度系统了，为此也诞生了不少基于 Docker 的容器调度系统，比如 Docker 原生的调度系统Swarm、Mesosphere 出品的Mesos，以及 Google 开源的大名鼎鼎的Kubernetes。下面我就结合微博的实践经验，给你讲讲容器调度要解决哪些问题。

1. 主机过滤

主机过滤是为了解决容器创建时什么样的机器可以使用的问题，主要包含两种过滤。

- 存活过滤。也就是说必须选择存活的节点，因为主机也有可能下线或者是故障状态。
- 硬件过滤。打个比方，现在你面对的集群有 Web 集群、RPC 集群、缓存集群以及大数据集群等，不同的集群硬件配置差异很大，比如 Web 集群往往用作计算节点，它的 CPU 一般配置比较高；而大数据集群往往用作数据存储，它的磁盘一般配置比较高。这样的话如果要创建计算任务的容器，显然就需要选择 Web 集群，而不是大数据集群。

上面这两种过滤方式都是针对主机层次的过滤方式，除此之外，Swarm 还提供了容器层次的过滤，可以实现只有运行了某个容器的主机才会被加入候选集等功能。

2. 调度策略

调度策略主要是为了解决容器创建时选择哪些主机最合适的问题，一般都是通过给主机打分来实现的。比如 Swarm 就包含了两种类似的策略：spread 和 binpack，它们都会根据每台主机的可用 CPU、内存以及正在运行的容器的数量来给每台主机打分。spread 策略会选择一个资源使用最少的节点，以使容器尽可能的分布在不同的主机上运行。它的好处是可以使每台主机的负载都比较平均，而且如果有一台主机有故障，受影响的容器也最少。而 binpack 策略恰恰相反，它会选择一个资源使用最多的节点，好让容器尽可能的运行在少数机器上，节省资源的同时也避免了主机使用资源的碎片化。

具体选择哪种调度策略，还是要看实际的业务场景，通常的场景有：

- 各主机的配置基本相同，并且使用也比较简单，一台主机上只创建一个容器。这样的话，每次创建容器的时候，直接从还没有创建过容器的主机当中随机选择一台就可以了。
- 在某些在线、离线业务混布的场景下，为了达到主机资源使用率最高的目标，需要综合考量容器中跑的任务的特点，比如在线业务主要使用 CPU 资源，而离线业务主要使用磁盘和 I/O 资源，这两种业务的容器大部分情况下适合混跑在一起。
- 还有一种业务场景，主机上的资源都是充足的，每个容器只要划定了所用的资源限制，理论上跑在一起是没有问题的，但是某些时候会出现对每个资源的抢占，比如都是 CPU 密集型或

者 I/O 密集型的业务就不适合容器混布在一台主机上。

所以实际的业务场景，对调度策略的要求比较灵活，如果 Swarm 提供的 spread 和 binpack 满足不了的话，可能就需要考虑自行研发容器调度器了。

服务编排

1. 服务依赖

大部分情况下，微服务之间是相互独立的，在进行容器调度的时候不需要考虑彼此。但有时候也会存在一些场景，比如服务 A 调度的前提必须是先有服务 B，这样的话就要求在进行容器调度的时候，还需要考虑服务之间的依赖关系。

为此，Docker 官方提供了Docker Compose的解决方案。它允许用户通过一个单独的 docker-compose.yaml 文件来定义一组相互关联的容器组成一个项目，从而以项目的形式来管理应用。比如要实现一个 Web 项目，不仅要创建 Web 容器比如 Tomcat 容器，还需要创建数据库容器比如 MySQL 容器、负载均衡容器比如 Nginx 容器等，这个时候就可以通过 docker-compose.yaml 来配置这个 Web 项目里包含的三个容器的创建。

Docker Compose 这种通过 yaml 文件来进行服务编排的方式是比较普遍的算法，以微博的业务为例，也是通过类似 yaml 文件的方式定义了服务扩容的模板，模板除了定义了服务创建容器时的镜像配置、服务池配置以及主机资源配置以外，还定义了关联依赖服务的配置。比如微博的 Feed 服务依赖了 user 服务和 card 服务，假如 user 服务扩容的模板 ID 为 1703271839530000，card 服务扩容的模板 ID 为 1707061802000000，那么 Feed 服务的扩容模板里就会像下面这样配置，它代表了每扩容 10 台 Feed 服务的容器，就需要扩容 4 台 user 服务的容器以及 3 台 card 服务的容器。

复制代码

```
{"Sid":1703271839530000,"Ratio":0.4}
{"Sid":1707061802000000,"Ratio":0.3}
```

2. 服务发现

容器调度完成以后，容器就可以启动了，但此时容器还不能对外提供服务，服务消费者并不知道这个新的节点，所以必须具备服务发现机制，使得新的容器节点能够加入到线上服务中去。

根据我的经验，比较常用的服务发现机制包括两种，一种是基于 Nginx 的服务发现，一种是基于注册中心的服务发现。

- 基于 Nginx 的服务发现

这种主要是针对提供 HTTP 服务的，当有新的容器节点时，修改 Nginx 的节点列表配置，然后利用 Nginx 的 reload 机制，会重新读取配置从而把新的节点加载进来。比如基于 Consul-

Template 和 Consul，把 Consul 作为 DB 存储容器的节点列表，Consul-Template 部署在 Nginx 上，Consul-Template 定期去请求 Consul，如果 Consul 中存储的节点列表发生变化，就会更新 Nginx 的本地配置文件，然后 Nginx 就会重新加载配置。

- 基于注册中心的服务发现

这种主要是针对提供 RPC 服务的，当有新的容器节点时，需要调用注册中心提供的服务注册接口。注册中心的服务发现机制在专栏第 5期我有过详细讲解，你可以再回顾一下它的原理。在使用这种方式时，如果服务部署在多个 IDC，就要求容器节点分 IDC 进行注册，以便实现同 IDC 内就近访问。以微博的业务为例，微博服务除了部署在内部的两个 IDC，还在阿里云上也有部署，这样的话，内部机房上创建的容器节点就应该加入到内部 IDC 分组，而云上的节点应该加入到阿里云的 IDC。

3. 自动扩缩容

容器完成调度后，仅仅做到有容器不可用时故障自愈还不够，有时候还需要根据实际服务的运行状况，做到自动扩缩容。

一个很常见的场景就是，大部分互联网业务的访问呈现出访问时间的规律性。以微博业务为例，白天和晚上的使用人数要远远大于凌晨的使用人数；而白天和晚上的使用人数也不是平均分布的，午高峰 12 点半和晚高峰 10 点半是使用人数最多的时刻。这个时候就需要根据实际使用需求，在午高峰和晚高峰的时刻，增加容器的数量，确保服务的稳定性；在凌晨以后减少容器的数量，减少服务使用的资源成本。

常见的自动扩缩容的做法是根据容器的 CPU 负载情况来设置一个扩缩容的容器数量或者比例，比如可以设定容器的 CPU 使用率不超过 50%，一旦超过这个使用率就扩容一倍的机器。

总结

今天我给你讲解了容器运维平台的另外两个关键组成：容器调度和服务编排，并给出了常用的解决方案。你的业务团队在选择解决方案时，要根据自己的需要选择合适的方案，而不是理论上最好的。

比如 Kubernetes 解决方案在容器调度、服务编排方面都有成熟的组件，并且经过大业务量的实际验证。但是要考虑到 Kubernetes 本身的复杂性以及概念理解的门槛，对于大部分中小业务团队来说，在生产环境上使用 Kubernetes 都会显得大材小用，并且还需要部署并运维 Kubernetes 周边的一些基础设施，比如 etcd 等。

相比之下，Docker 原生自带的解决方案 Swarm 和 Compose 就要简单得多，但是功能也比较有限，如果不能满足你的业务需求的话，也不好再二次开发。

在了解了镜像仓库、资源调度、容器调度、服务编排后你会发现，微服务容器化后最大的挑战其实来自于原有运维设施如何支持容器的运维，是在原有运维平台上升级还是完全采用新的容器运维平台，这才是关键，往往不能一蹴而就，需要逐步按照业务进行替换升级。但是考虑到微服务容器化后所带来的种种好处，采用新的运维模式势在必行。

思考题

容器调度方面，业界最有名的莫过于 Swarm、Mesos 和 Kubernetes 了，你认为它们的优缺点是什么？分别适合什么业务场景？

欢迎你在留言区写下自己的思考，与我一起讨论。

写留言

精选留言

_CountingStars 1

不管什么场景直接选kubernetes绝对不会错 其他两个基本不用考虑 k8s 已经是业界标准 扩展很方便 唯一的缺点是需要理解k8s的一套概念 但是这对技术人来说不是事儿

2018-10-23

Stalary 0

cpu超过50%就自动扩容，那如果是代码逻辑错误引起的呢？这个时候会有相应的策略吗

2018-10-23