

59 | 数据中心操作系统：上市敲钟

2019-08-11 刘超

趣谈Linux操作系统

[进入课程 >](#)



讲述：刘超

时长 16:18 大小 14.94M



在这门课程里面，我们说了，在内核态有很多的模块，可以帮助我们管理硬件设备，最重要的四种硬件资源是 CPU、内存、存储和网络。

最初使用汇编语言的前辈，在程序中需要指定使用的硬件资源，例如，指定使用哪个寄存器、放在内存的哪个位置、写入或者读取那个串口等等。对于这些资源的使用，需要程序员自己心里非常地清楚，要不然一旦 jump 错了位置，程序就无法运行。

为了将程序员从对硬件的直接操作中解放出来，提升程序设计的效率，于是，我们有了操作系统这一层，用来实现对于硬件资源的统一管理。某个程序使用哪个 CPU、哪部分内存、哪部分硬盘，只需要调用 API 就可以了，这些都由操作系统自行分配和管理。

其实操作系统最总要的事情，就是调度。因此，在内核态就产生这些模块：进程管理子系统、内存管理子系统、文件子系统、设备子系统和网络子系统。

这些模块通过统一的 API，也就是系统调用，对上提供服务。基于这些 API，用户态有很多的工具可以帮我们使用好 Linux 操作系统，比如用户管理、软件安装、软件运行、周期性进程、文件管理、网络管理和存储管理。

但是到目前为止，我们能管理的还是少数几台机器。当我们面临数据中心成千上万台机器的时候，仍然非常“痛苦”。如果我们运维数据中心依然像的运维一台台物理机的前辈一样，天天关心哪个程序放在了哪台机器上，使用多少内存、多少硬盘，每台机器总共有多少内存、多少硬盘，还剩多少内存和硬盘，那头就大了。

因而对应到数据中心，我们也需要一个调度器，将运维人员从指定物理机或者虚拟机的痛苦中解放出来，实现对于物理资源的统一管理，这就是 Kubernetes。

Kubernetes 究竟有哪些功能，可以解放运维人员呢？为什么它能做数据中心的操作系统呢？

我列了两个表格，将操作系统的功能和模块与 Kubernetes 的功能和模块做了一个对比，你可以看看。

Linux操作系统	Kubernetes
用户管理	RBAC（Role-Based Access Control）
安装软件apt-get/yum	Helm
运行程序 （交互命令行运行、后台运行、服务方式运行）	Job、Deployment & Service、DaemonSet
周期性进程	CronJob
文件管理	对接对象存储和分布式文件系统Volumes
网络管理	Networking Model & NetworkPolicy
存储管理	对接分布式块存储Volumes

用户态功能

Linux操作系统	Kubernetes
进程管理子系统	Docker、Pod、Controller、Scheduler（亲和性）
内存管理子系统	Memory requests/limits
文件子系统	对接对象存储和分布式文件系统CSI
设备子系统	对接分布式块存储CSI
网络子系统	CNI、Calico、Flannel

内核态功能

Kubernetes 作为数据中心的操作系统还是主要管理数据中心里面的四种硬件资源：CPU、内存、存储、网络。

对于 CPU 和内存这两种计算资源的管理，我们可以通过 Docker 技术完成。它可以将 CPU 和内存资源，通过 namespace 和 cgroup，从大的资源池里面隔离出来，并通过镜像技术，实现计算资源在数据中心里面的自由漂移。

就像我们上面说的一样，那没有操作系统的时候，汇编程序员需要指定程序运行的 CPU 和内存物理地址。同理，数据中心的管理员，原来还需要指定程序运行的服务器及使用的

CPU 和内存。现在，Kubernetes 里面有一个调度器 Scheduler，你只需要告诉它，你想运行 10 个 4 核 8G 的 Java 程序，它会自动帮你选择空闲的、有足够资源的服务器，去运行这些程序。

对于操作系统上的进程来讲，有主线程做主要的工作，还有其它线程做辅助的工作。对于数据中心里面的运行的程序来讲，可以也会有一个主要提供服务的程序，例如上面的 Java 程序，也会有一些提供辅助功能的程序；例如监控、环境预设值等。Kubernetes 将多个 Docker 组装成一个 Pod 的概念，在一个 Pod 里面，往往有一个 Docker 为主，多个 Docker 为辅。

操作系统上的进程会在 CPU 上切换来切换去，它使用的内存也会换入换出。在数据中心里面，这些运行中的程序能不能在机器之间迁移呢？能不能在一台服务器故障的时候，选择其它的服务器运行呢？反正我关心的是运行 10 个 4 核 8G 的 Java 程序，又不在于它在哪台上运行。Kubernetes 里面有 Controller 的概念，可以控制 Pod 们的运行状态以及占用的资源，如果 10 个变 9 个就选一台机器添加一个，10 个变 11 个，就随机删除一个。

操作系统上的进程有时候有亲和性的要求，比如它可能希望再某一个 CPU 上运行，不切换 CPU 从而提高运行效率，或者两个线程要求在一个 CPU 上，从而可以使用 Per CPU 变量不加锁，交互和协作比较方便。有的时候，一个线程想避开另一个线程，不要共用 CPU，以防相互干扰。Kubernetes 的 Scheduler 也是有亲和性功能的，你可以选择两个 Pod 永远运行在一台物理机上，这样本地通信就可以了，也可以选择两个 Pod 永远不要运行在同一台物理机上，这样一个挂了不影响另一个。

你可能会问，Docker 可以将 CPU 内存资源进行抽象，在服务器之间迁移，那数据应该怎么办呢？如果数据放在每一台服务器上，其实就像散落在汪洋大海里面，用的时候根本找不到，所以必须要有统一的存储。正像一台操作系统上多个进程之间，要通过文件系统保存持久化的数据并且实现共享，在数据中心里面也需要一个这样的基础设施。

统一的存储常常有三种形式，我们分别来看。

第一种方式是**对象存储**。

顾名思义，这种方式是将文件作为一个完整对象的方式来保存。每一个文件对我们来说，都应该有一个唯一标识这个对象的 key，而文件的内容就是 value。对象可以分门别类地保存在一个叫作**存储空间**（Bucket）的地方，有点儿像文件夹。

对于任何一个文件对象，我们都可以通过 HTTP RESTful API 来远程获取对象。由于是简单的 key-value 模式，当需要保存大容量数据的时候，我们就比较容易根据唯一的 key 进行横向扩展，所以对象存储往往能够容纳的数据量非常大。在数据中心里面保存文档，视频等很好的方式，当然缺点就是，你没办法像操作文件一样操作它，而是要将 value 当成整个的来对待。

第二种方式是**分布式文件系统**。

这种是最容易习惯的，因为使用它和使用本地的文件系统几乎没有什么区别，只不过是通过网络的方式访问远程的文件系统。多个容器能看到统一的文件系统，一个容器写入文件系统的，另一个容器能够看到，可以实现共享。缺点是分布式文件系统的性能和规模是个矛盾，规模一大性能就难以保证，性能好则规模不会很大，所以不像对象存储一样能够保持海量的数据。

第三种方式是**分布式块存储**。

这是云硬盘，也即存储虚拟化的方式，只不过将盘挂载给容器而不是虚拟机。块存储没有分布式文件系统这一层，一旦挂载到某一个容器，可以有本地的文件系统，这样缺点是一般情况下，不同容器挂载的块存储都是不共享的，好处是在同样的规模的情况下，性能相对分布式文件系统要好。如果为了解决一个容器从一台服务器迁移到另一台服务器，如何保持存储的数据的问题，块存储是一个很好的选择。它不用解决多个容器共享数据的问题。

这三种形式，对象存储使用 HTTP 进行访问，当然任何容器都能访问到，不需要 Kubernetes 去管理它。而分布式文件系统和分布式块存储，就需要对接到 Kubernetes，让 Kubernetes 可以管理它们。如何对接呢？Kubernetes 提供 Container Storage Interface (CSI) 接口，这是一个标准接口，不同的存储可以实现这个接口来对接 Kubernetes。是不是特别像设备驱动程序呀。操作系统只要定义统一的接口，不同的存储设备的驱动实现这些接口，就能被操作系统使用了。

存储的问题解决了，接下来是网络。因为不同的服务器上的 Docker 还是需要互相通信的。

Kubernetes 有自己的网络模型，里面是这样规定的。

IP-per-Pod，每个 Pod 都拥有一个独立 IP 地址，Pod 内所有容器共享一个网络命名空间。

集群内所有 Pod 都在一个直接连通的扁平网络中，可通过 IP 直接访问。

所有容器之间无需 NAT 就可以直接互相访问。

所有 Node 和所有容器之间无需 NAT 就可以直接互相访问。

容器自己看到的 IP 跟其它容器看到的一样。

这其实是说，里面的每一个 Docker 访问另一个 Docker 的时候，都是感觉在一个扁平的网络里面就可以了。

要实现这样的网络模型，有很多种方式，例如 Kubernetes 自己提供 Calico、Flannel。当然，也可以对接 Openvswitch 这样的虚拟交换机，也可以使用 brctl 这种传统的桥接模式，也可以对接硬件交换机。

看，这又是一种类似驱动的模式，和操作系统面临的问题是一样的。Kubernetes 同样是提供统一的接口 Container Network Interface (CNI，容器网络接口)。无论你用哪种方式实现网络模型，只要对接这个统一的接口，Kubernetes 就可以管理容器的网络。

至此，Kubernetes 作为数据中心的操作系统，内核的问题解决了。

接下来是用户态的工具问题了。我们能不能像操作一台服务器那样操作数据中心呢？

使用操作系统，需要安装一些软件，于是，我们需要 yum 之类的包管理系统，使得软件的使用者和软件的编译器分隔开来，软件的编译器需要知道这个软件需要安装哪些包，包之间的依赖关系是什么，软件安装到什么地方，而软件的使用者仅仅需要 yum install 就可以了。Kubernetes 就有这样一套包管理软件 Helm，你可以用它来很方便的安装，升级，扩容一些数据中心里面的常用软件，例如数据库，缓存，消息队列。

使用操作系统，运行一个进程是最常见的需求。第一种进程是**交互式命令行**，运行起来就是执行一个任务，结束了马上返回结果。在 Kubernetes 里面有对应的概念叫做 Job，Job 负责批量处理短暂的一次性任务 (Short Lived One-off Tasks)，即仅执行一次的任务，它保证批处理任务的一个或多个 Pod 成功结束。

第二种进程是**nohup 长期运行的进程**。在 Kubernetes 里对应的概念是 Deployment，使用 Deployment 来创建 ReplicaSet。ReplicaSet 在后台创建 pod。也即 Deployment 里

面会声明我希望某个进程以 N 的 Pod 副本的形式运行，并且长期运行，一旦副本变少就会自动添加。

第三种进程是**系统服务**。在 Kubernetes 里面对应的概念是 DaemonSet，它保证在每个节点上都运行一个容器副本，常用来部署一些集群的日志、监控或者其它系统管理应用。

第四种进程是**周期性进程**，也即**crontab**，常常用来设置一些周期性的任务。在 Kubernetes 里面对应的概念是 CronJob 定时任务，就类似于 Linux 系统的 crontab，在指定的时间周期运行指定的任务。

使用操作系统，我们还需使用文件系统，或者使用网络发送数据，虽然在 Kubernetes 里面有 CSI 和 CNI 来对接存储和网络，在用户态，不能让用户意识到后面具体设备，而是应该有抽象的概念。

对于存储来讲，Kubernetes 有 Volume 的概念，Kubernetes Volume 的生命周期与 Pod 绑定，

容器挂掉后 Kubelet 再次重启容器时，Volume 的数据依然还在，而 Pod 删除时，Volume 才会清理。数据是否丢失取决于具体的 Volume 类型。Volume 的概念是对具体存储设备的抽象，就像当我们使用 ext4 文件系统不用管它是基于什么硬盘一样。

对于网络来讲，Kubernetes 有自己的 DNS，有 Service 的概念，Kubernetes Service 是一个 Pod 的逻辑分组，这一组 Pod 能够被 Service 访问。每一个 Service 都有一个名字，Kubernetes 会将 Service 的名字作为域名进行解析，称为一个虚拟的 Cluster IP，然后通过负载均衡，转发到后端的 Pod，虽然 Pod 可能漂移，IP 会变，但是 Service 会一直不变。

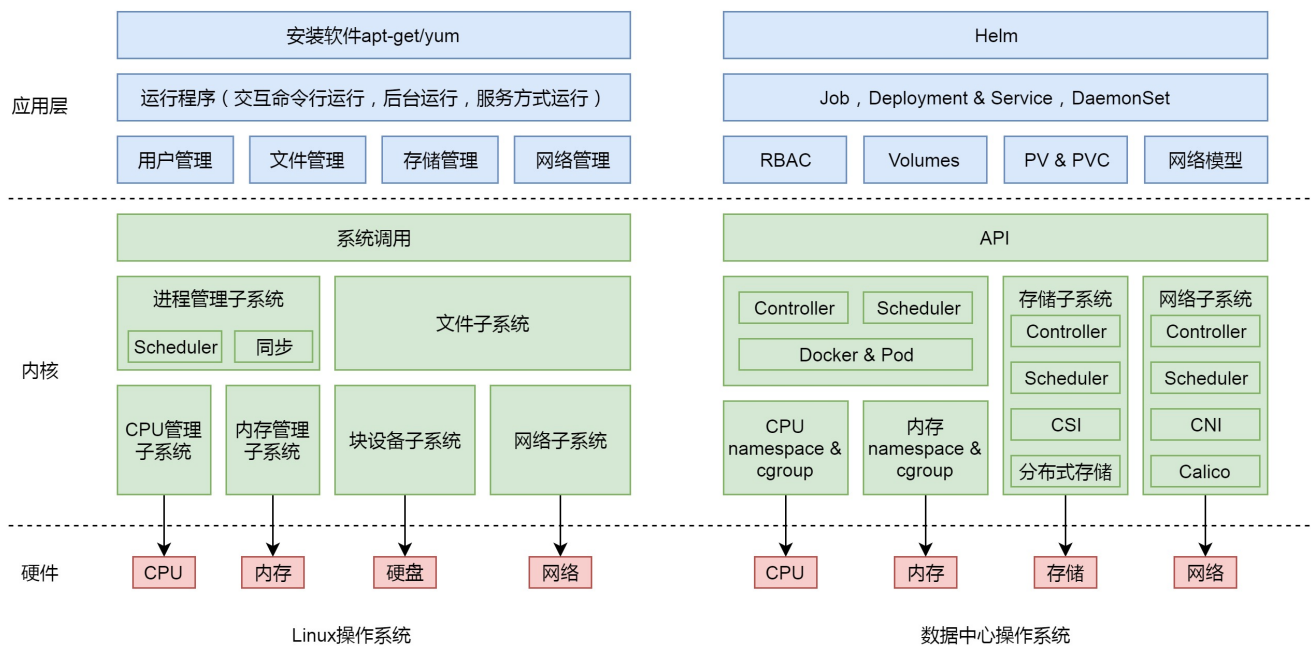
对应到 Linux 操作系统的 iptables，Kubernetes 在有个概念叫 Network Policy，Network Policy 提供了基于策略的网络控制，用于隔离应用并减少攻击面。它使用标签选择器模拟传统的分段网络，并通过策略控制它们之间的流量以及来自外部的流量。

看，是不是很神奇？有了 Kubernetes，我们就能像管理一台 Linux 服务器那样，去管理数据中心了。

如果想深入了解 Kubernetes 这个数据中心的操作系统，你可以订阅极客时间的专栏 [“深入剖析 Kubernetes”](#)。

总结时刻

下面，你可以对照着这个图，来总结一下这个数据中心操作系统的功能。



课堂练习

Kubernetes 有一个简单的版本，可以用一台虚拟机进行尝试，请你按照官方文档安装一个进行尝试。

欢迎留言和我分享你的疑惑和见解，也欢迎收藏本节内容，反复研读。你也可以把今天的内容分享给你的朋友，和他一起学习和进步。

趣谈 Linux 操作系统


像故事一样的操作系统入门课

刘超

网易杭州研究院

云计算技术部首席架构师



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 58 | CGroup技术：内部创业公司应该独立核算成本

精选留言 (1)

 写留言



安排

2019-08-12

通俗易懂

展开

