《openEuler内核编程》

课程讲稿

第十一章 第3讲

容器与iSula

软件所制

第十一章 第3讲 容器与iSula

**学时：**1学时

**教学目的：** 了解实现容器需要的技术，简单介绍Docker和iSula。

**课程时间线：**

LXC（Linux Container）P4~P6

命名空间（Linux Namespace）P7~P14

控制组（Linux Control Group) P15~P17

Docker P18~P26

iSula P27~P29

15min

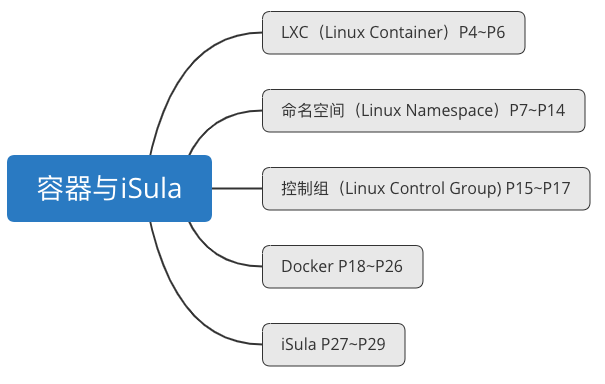
25 min

30min

40min

45 min

**知识框图：**



**课外参考读物：**

1. <https://zhuanlan.zhihu.com/p/19796979>
2. https://www.oschina.net/news/111410/linux-market-growth
3. https://lkml.org/lkml/2020/5/17/314
4. https://www.oschina.net/news/100042/linux-september-2018-stats

**PPT讲稿：**



LXC全称为Linux Container，相比KVM而言，LXC针对的虚拟化层次是应用程序，是一种不需要在内核之上为某一个或某几个应用程序提供虚拟化的手段，将要虚拟执行的应用软件的各个进程打包为一个容器，通过统一的命名空间和共享API来管理容器可接触到的硬件资源；每个容器内部的应用程序，都会认为自己接触到了一个真实的内核和全局的硬件资源。通过限制不同容器的资源使用、对这些容器进行隔离，LXC在应用程序层实现虚拟化。

在Linux中，实现容器机制主要依赖于以下三种技术：chroot，namespace，cgroup。其中，Chroot通过重定向根目录的实际位置来隔离不同容器中的文件系统目录，Namespace对软件的执行环境进行隔离，保证不同容器之间的PID、主机名与域名等命名资源是彼此透明的；cgroup则在内核中提供了cgroup子模块支持，帮助实现不同控制组之间资源使用的隔离。

Chroot最早在Unix系统中作为系统调用被提出，具体的实现效果如图中所示，为修改当前进程的文件系统根目录所指向的实际目录位置，能够对当前进程执行过程中所接触到的文件系统内容进行约束和隔离。根据类Unix系统的文件系统设计，限定了root路径所在位置，就能够防止当前程序访问到文件系统中的其他位置。



命名空间是对全局资源的一种内核级别隔离方法，在Linux内核的进程控制块中，有专门用于管理命名空间的成员，标识了当前进程所在的命名空间，也使新创建的进程的命名空间与原有各进程保持一致。一个进程可以同时分属于多个命名空间管理，但每个命名空间代表该进程的不同方面，比如网络端口、PID等。每一个命名空间能同时管理一组进程的执行环境，同一个命名空间中的所有进程共享该空间的命名资源，特定名称在命名空间内唯一；如在同一个PID Namespace中的所有进程的PID彼此可见，彼此不同。

命名空间支持隔离的资源主要包括一些必须在各个容器间彼此透明的命名资源，否则将可能引起混淆，导致不同进程之间的命名不唯一，运行时出错。包括主机名和域名、进程间通信相关的信号量和消息队列以及共享内存、进程的编号、网络设备名称、挂载点信息、用户和用户组等六大类。

Linux提供了一席与命名空间相关的API，包括clone、unshare和setns、ioctl。其中clone本身是用于创建进程的系统调用，但与其他如fork等不同的是，clone系统调用创建进程时能够指定新进程的命名空间，或为新进程同步创建一个新的命名空间。

1. Unshare接口帮助用户程序改变当前进程所在的命名空间。

Setns接口则是set namespace的缩写，将当前进程加入到一个已存在的命名空间中，对命名空间的操作通过特定目录下的文件描述符，即/proc/pid/ns

使用ioctl+文件系统来对某些接口进行统一设计则是Unix“一切皆文件”思想的体现，命名空间也不例外，ioctl这个广泛使用的系统调用同样能够用于查询特定命名空间的信息。

对于容器虚拟化而言，命名空间解决了逻辑环境的隔离。容器虚拟化需要的效果，是容器内的进程对自己本身所在的运行环境真实性不可感知。因此在容器中运行的进程，与在真实环境中运行的进程在编号特征上应该保持一致；例如根进程的PID为0，而如果不隔绝各个容器之间的命名情况，无法满足这些条件，就会造成虚拟化的环境被应用程序感知。更进一步，命名空间只能隔绝不同容器间的命名情况，避免名称分配时的相互影响，但任意容器内的进程都能通过系统调用对全局资源产生影响，这与我们希望达到的资源隔离明显不符。而使用Cgroup能够实现资源使用的隔离

Cgroup是Linux控制组，能够限制、控制、分离一组进程使用的资源，确保组内进程使用的资源数量收到限制，对组与组之间进行资源协调和倾斜，对不同容器的资源使用情况进行审计、核查。Cgroup能够隔离的资源包括CPU使用时间、占用的系统内存、传输时的网络带宽等。

同样，基于“一切皆文件”的设计思想，Cgroup也对用户态暴露了文件系统相关的接口，在内核中将对应路径的文件读写重定向为cgroup模块的操作。例如创建一个控制组在用户态仅表现为在特定路径下创建一个目录，而创建目录之后目录内自然会存在相应的文件，通过将参数写入文件来控制进程组内的资源使用如CPU资源等等。



基于LXC等技术，Docker新瓶装旧酒，用Go语言开发了一整套易用的框架，让开发者能够方便地将应用部署到容器内、管理容器。Docker项目贡献了一个开源的应用容器引擎，基于Go语言开发，遵从Apache2.0开源协议。从图中可以看出，Docker内部能够提供内核之上的各层次模拟，最直观的是在内核之上模拟各种Linux发行版，是Web服务器等基于Linux系统的应用，在部署服务方面易用性大大增加。

Docker依赖于Cgroup和Namespace这两项成熟技术，同时结合了UnionFS的文件系统实现。能够自动执行重复性的任务，如搭建、配置开发环境和生产环境等，让用户方便地创建和使用容器，对容器本身进行版本控制、复制、分享、修改等。

在LXC技术之上，Docker真正提出了将任何需要运送的代码封装、转移、管理的目标，并在一定程度上将各种环境下的服务部署、环境搭建等统一起来。Docker的使用让服务维护人员真正关注服务本身，而无需对服务之下依赖的系统发行版等过多关注。

在继续介绍Docker之前我们先对Docker中的基本概念做一个介绍，在Docker的设计中，运行时的Linux容器实例被称为“容器”，而持久化保存到文件系统中的“容器”被称为“镜像”，对“镜像”进行管理、分发的远端网站则被称为“仓库”。

镜像中包含了一个容器运行所需要的所有静态支持，将以文件的形式出现在运行环境中，包括程序的可执行文件、程序库、配置文件等静态文件，匿名卷信息、环境变量、用户数据等用于容器启动的信息。但镜像中不会包含任何动态数据，一个镜像在构建之后其内部的数据将不能更改，但可以作为另一个镜像的基础，用于构建新的镜像。

容器的实质是资源受控制的进程组，且对组之外的其他进程彼此透明。镜像中保存着启动容器需要的各种信息和文件系统内容，Docker引擎负责解析这些信息、挂载文件系统，来创建容器，容器被创建之后能够启动、停止、删除、暂停，运行着的容器也可以被再次保存，成为新的镜像，但只有一部分运行时信息会被完整保存并复现，类似内存中的程序运行时信息如果没有写入到文件系统，将会消失。容器存储层的生命周期仅与容器相同，而宿主挂载的外部卷和已经绑定的宿主目录，写入到其中的内容能够被持久性保存。

仓库是集中存放镜像的位置，这里的仓库与容器的关系可以被类比为github与其上的开源代码的关系。代码能够被上传到github，从而被存储、分发，并进行版本控制。Docker同样提供的类似功能。

1. 这里是几个常见的Docker命令，包括构建进项、运输镜像、启动容器实例等。

iSula是一种新型的容器解决方案，通过提供统一架构来满足不同的需求，相比Docker而言更轻巧，底层开销更小。

1. iSula引擎对外提供三种接口，能够供Kubernets和HASEN使用，同时提供lcrc服务。服务端的lcrc服务主要包括Content Service、Rootfs Service、Execution Service，用来管理容器实例的创建、运行和保存。iSulad底层以插件形式给出拓展，支持多种插件，默认为lcr插件。