

**企业实习中期报告**

学号： 2014220301019

姓名： 卢杰

专业方向： 嵌入式系统

企业名称： 英特尔亚太研发有限公司

实习岗位名称： 虚拟化开发工程师

企业指导教师： 王震宇

院内代管教师： 廖勇

**信息与软件工程学院**

目 录

[1. 企业实习的进展情况 1](#_Toc479152473)

[1.1. 实习工作完成情况 1](#_Toc479152474)

[1.2. 知识技能学习情况 1](#_Toc479152475)

[1.3. 职业素养学习培养 1](#_Toc479152476)

[1.4. 工程协作与交流情况 1](#_Toc479152477)

[2. 存在问题与解决方案 2](#_Toc479152478)

[2.1. 存在的主要问题 2](#_Toc479152479)

[2.2. 解决方案与可行性研究 2](#_Toc479152480)

[3. 前期任务完成度与后续实施计划 3](#_Toc479152481)

[参考文献 4](#_Toc479152482)

## 

## 1. 企业实习的进展情况

### 1.1. 实习工作完成情况

* + 1. 推理分析

（1）0Container与VM比较

VM的理念是在宿主的系统上，自己虚拟了一个硬件平台，然后运行一个不同的OS，每个vm需要很多资源，所以一台机器上跑不了几个vm。

Container的理念是依托于宿主机提供的内核，仅仅把一个不同的Linux发行版所需的特性打包成一个image，当运行一个ubuntu镜像，感觉就是在一个ubuntu操作环境里，但实际调用的系统接口都来自宿主机。因为共用宿主机内核，所以其所需资源很少，性能开销很小，一台机器可以起上千个Container。其比较如图1-1所示。

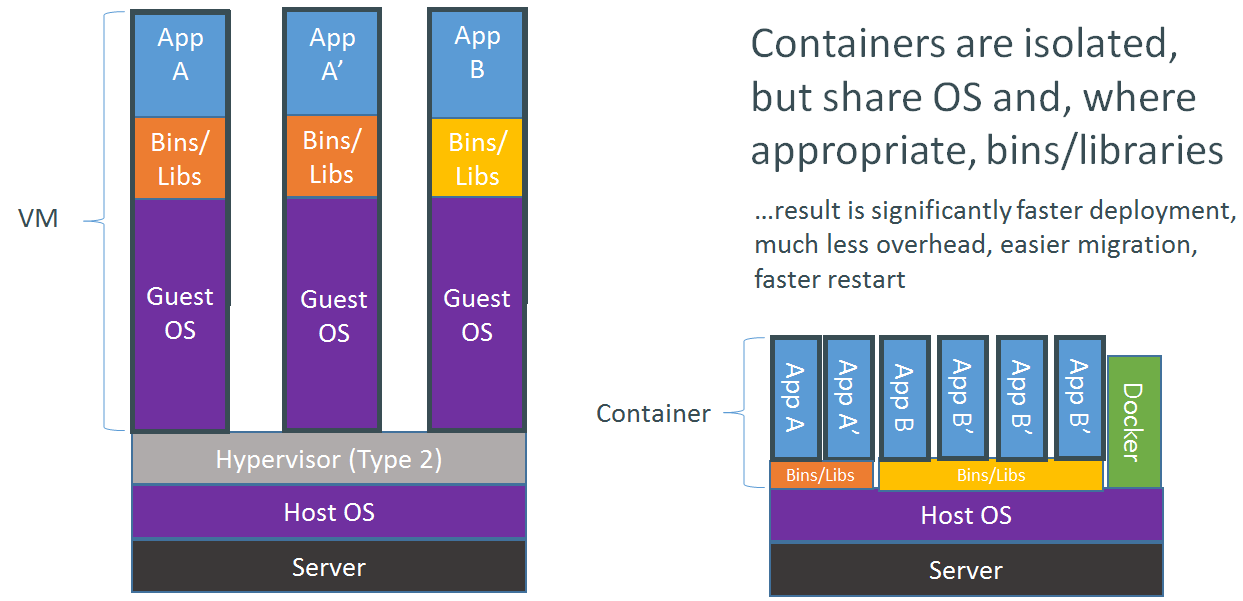


图1-1 Container与VM比较

表1-1 VM与Container关注点

|  |  |  |  |
| --- | --- | --- | --- |
|  | 运行在 | 关注 | 涉及的产品 |
| VM | Root Image运行于设备存储上；  Image存储在对象存储上；  文件存储不在VM关注层次上； | 性能  容量  可拓展 | VSAN，Ceph  存储接口RBD,  Cinder等 |
| Docker Container | Root Image运行于块设备存储  Image存储在对象存储上  Volume是数据存储的接口 | 文件分层  迁移  应用感知  细粒度控制 | Union FS  DeviceMapper  ZFS  存储Flocker |

（2）Container 组成

一个Container大体上由namespaces，resource control，filesystem组成，其中namespaces用于限制使用对象所能看见的空间，resource control用于限制使用对象能够使用的资源多少；filesystem的上层用于保存当前的修改，底层用于读取image。一个Container的组成如图1-2所示。

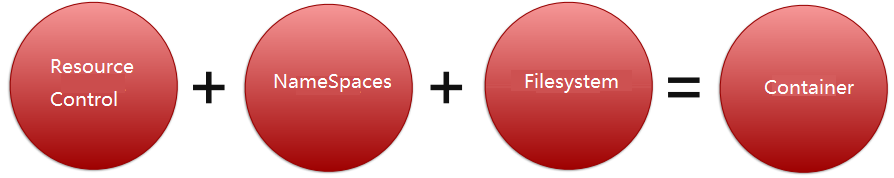


图1-2 Container基本组成要素

（3）Container cgroup

cgroup 是 Linux 内核的一个特性，此特性可以帮助用户对一组进程进行资源使用的限制、统计以及隔离。 Docker 领域也不例外， Docker 利用对 cgroup 的支持，完成对 Docker 容器进程组的资源限制、统计以及隔离。与 cgroup 内容相关的 Cgroup 定义，位于 ./docker/libcontainer/cgroups/cgroups.go，如下:

|  |
| --- |
| type Cgroup struct {  Name string  Parent string  AllowAllDevices bool  // name of parent cgroup or slice  // If this is true allow access to any  kind of device within the container. If false,  allow access only to devices explicitly listed  in the allowed devices list.  AllowedDevices []\*devices.DeviceMemory 工 nt64  MemoryReservat 工。 n int64  MemorySwap int64  CpuShares int64  CpuQuota int64  CpuPeriod int64  CpusetCpus string  第 14 章 libcontainer 介绍 e争 225  // Memory limit (in bytes)  // Memory reservation or soft\_limit (in bytes)  // Total memory usage (memory + swap); set  、 -1' to d 工 sable swap  // CPU shares (re1ative weight vs. other  conta 工 ners)  // CPU hardcap limit (in usecs). Allowed cpu  time in a given period.  // CPU period to be used for hardcapping (in  usecs). 0 to use system default.  // CPU to use  Freezer FreezerState // set the freeze value for the process  Sl 工 ce string // Parent slice to use for systemd |

从结构体 Cgroup 的定义可见， Docker 对于 cgroup 的支持，主要有以下 5 方面:设备( device)、内存 (Memo可)、 CPU 、 Freezer 以及 systemd。在容器设备方面， Docker 支持让用户选择容器可以使用的设备。在内存方面，支持为容器的运行设定用量限额。在 CPU 方面，支持容器进程之间拥有相对的运行时间片。 Freezer 则可以使容器挂起，节省 CPU 资源，Docker 命令中 pause 与 unpause 命令即采用了 cgroup 的 Freezer 子系统。需要注意的是，容器进程组挂起，并不意味着进程已经终止，从 Linux 内核的角度来看，被挂起的进程拥有完整的 task struct ，占用相应的内存，但是 Freezer 子系统保证容器中的进程不会被 CPU 调度到。 Slice 属性则属于 systemd 方面的配置。

（4）volume plugin

Volume是个interface，Docker实现两种Volume：①基于主机文件系统。②基于Volume Plugin。

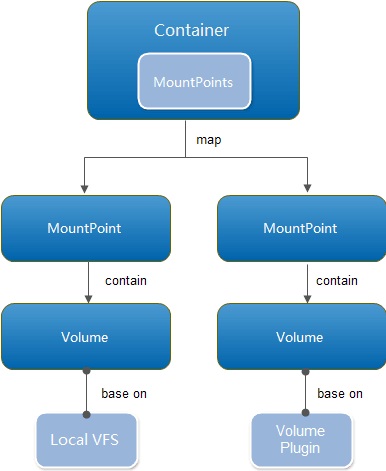


图1-3 Container中的Volume

Docker Volume Plugin的总体框架主要包括Docker Daemon对Volume的管理；docker volume的管理；docker plugin实现原理三个主要部分，通过分析其总体框架，得以分析plugin机制是否能够满足我们对于driver版本检查的目的。

1. Docker Daemon对Volume的管理

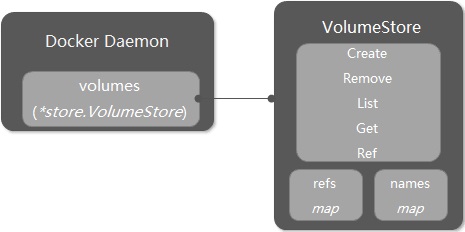


图1-4 Daemon 管理 Volume

如上图1-4所示，Docker Daemon结构中有个成员Volumes，类型是VolumeStore类型，包含一组管理Volume的函数：Create、Remove、List、Get、Refs ...。通过两个变量，管理Container和Volume的关系。 names : map结构，Key是Volume的name，value是实现Volume接口的结构对象。存储该Daemon内所有的Volume。 ReFS: map结构，key是volume的name，value是string数组保存引用该Volume的Container ID。 Docker Daemon通过Volumes变量，就可以管理所有的Volume，提供如下命令：

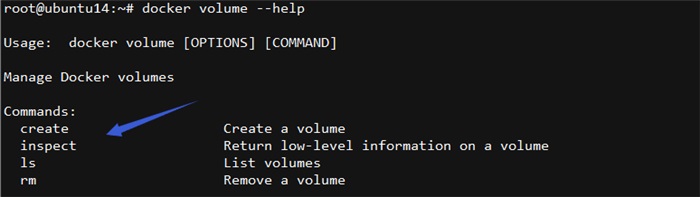


图1-5 Volume 命令

1. docker volume管理

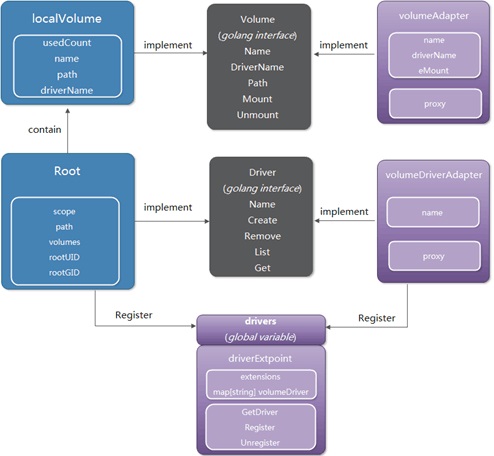


图1-7 基于本地文件系统的volume框架

Docker提供两个接口Volume和Driver，所有提供给Docker使用的Volume必须实现Volume接口。后端驱动需要实现Driver接口。Driver是对提供出去的Volume进行管理。目前Docker实现了两种Volume &Driver。

1. **基于本地文件系统的Volume**

可以在执行Docker create或Docker run时，通过-v参数将主机的目录作为容器的数据卷。这部分功能便是基于本地文件系统的volume管理。上图中蓝色部分LocalVolume和Root。这两个结构就是对主机目录和文件进行管理，具体的对应关系如下图：

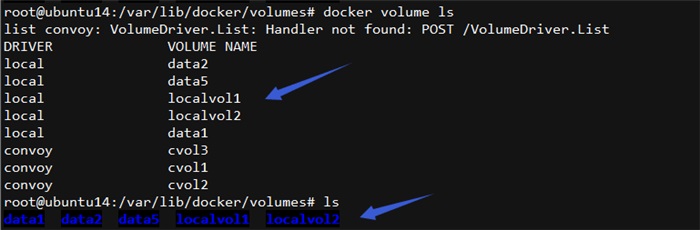


图1-8 localvolume 管理

从上图1-8可以看出，基于本地文件系统的卷管理，Driver便是卷的根目录/var/lib/docker/volumes。一个卷就是根目录下的一个子目录。

② 适配Plugin的Volume Docker为了支持第三方存储方案，在1.8版本引入Volume Plugin机制，Volume Adapter和Volume Driver Adapter分别实现了接口Volume和Driver接口，用于表示由Plugin提供的Volume和Plugin driver。在下一小节详细描述。

基于本地文件系统的Volume框架中，全局变量drivers保存了所有注册到Docker Daemon的Driver。Docker Daemon需要对Volume进行管理操作时，通过GetDriver函数从drivers变量中获取指定名称的Driver，通过Driver可以通过Create，Remove，List，Get对Driver中Volume进行管理。通过Get函数获取Volume结构，可以对卷进行管理操作：Name，DriverName，Path，Mount，Unmount。

1. docker plugin实现原理

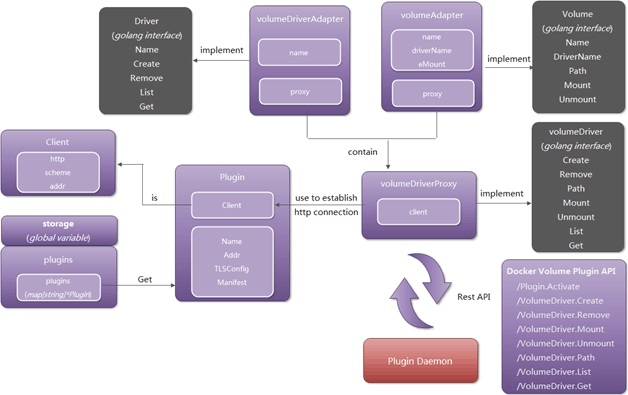


图1-9 Volume Plugin实现原理

1. Docker Plugin机制

Docker针对Volume Plugin实现了两个适配类型Volume Driver Adapter和Volume Adapter。

Volume Driver Adapter : 实现Driver接口，用于抽象各种Plugin的驱动，该类型可以适配所有符合规范的Volume Plugin，对Plugin进行管理。

Volume Adapter : 实现Volume接口，用于抽象所有Plugin提供的Volume，该类型可以适配所有符合规范的Volume Plugin提供的类型，对Volume进行管理。 通过抽象，对于Plugin和其提供的Volume，Docker Daemon结构和Container结构使用上述两个适配类型，对Plugin和Volume进行管理和使用。

上述两个类型都有一个Volume Driver Proxy结构变量proxy，用于与Plugin进行通信。Volume Driver Proxy结构实现了与plugin通信的接口volume Driver，提供Create、Remove、Path、Mount、Unmount、List、Get接口与通信。Volume Driver Proxy结构的client变量，使用这个变量与Plugin Daemon进行通信。client变量是Plugin结构中的Client变量是Client结构类型，包含了http.Client类型对象。

图1-9 中左下方，Plugins结构类型的全局变量Storage保存所有被Docker Daemon发现的Plugin结构。

② Docker Plugin的发现过程

Docker Daemon通过Unix域套接字与Plugin Daemon进行通信。所以Plugin需要让Docker Daemon知道Plugin的Unix域套接字文件的路径。再执行命令：docker run ... -v volumename:/data --volume-driver=convoy

发现步骤： Docker Daemon首先会在/run/docker/plugins搜索对应的套接字文件，套接字文件名必须和Volume Driver名一致，如上述命令，便是搜索convoy.sock。 如果上一步搜索不到，则到/etc/docker/plugins和/usr/lib/docker/plugins两个目录搜索spec或json文件。文件中指定套接字文件的URL。如：unix:///var/run/convoy/convoy.sock 。 根据前面两步发现的Unix域套接字URL，构建Plugin对象，并将新建对象加入到全局变量storage的Plugins字段中。

③ Docker Volume Plugin的使用

执行docker run命令时，指定参数--volume-driver=convoy。Docker Daemon会先从storage.plugins中寻找Plugin结构，如果没有找到，就发起②的发现流程。找到则直接使用Client构建volume Driver Proxy。

④ Docker Daemon与Plugin Daemon通信的API

前面已经提到过，Docker Daemon和Plugin Daemon基于Unix域套接字，使用Restful API进行通信，下面是详细的API：

Plugin.Activate : 发送一个请求到Plugin，Plugin返回其类型，如convoy就返回Volume Driver。相当于Docker和Plugin Daemon直接的连接建立的握手报文。

VolumeDriver.Create : 创建一个卷，Docker会发送卷名称和参数发送给插件，卷插件会根据Docker发送过来的参数创建一个卷，并和这个卷名称关联。

VolumeDriver.Mount : 挂载一个卷到本机，Docker会把卷名称和参数发送给参数。插件会返回一个本地路径给Docker，这个路径就是卷所在的位置。Docker在创建容器的时候，会将这个路径挂载到容器中。

VolumeDriver.Path : 一个卷创建成功后，Docker会调用Path API来获取这个卷的路径，随后Docker通过调用Mount API，让插件将这个卷挂载到本机。

VolumeDriver.Unmount : 当容器退出时，Docker daemon会发送Umount API给插件，通知插件这个卷不再被使用，插件可以对该卷做些清理工作（比如引用计数减一，不同的插件行为不同）。

VolumeDriver.Remove : 删掉特定的卷时调用，当运行"docker rm -v"命令时，Docker会调用该API发送请求给插件。

VolumeDriver.List : 执行docker volume ls命令时，会向plugin发送该请求，获取volume list。

VolumeDriver.Get : 获取Plugin Volume的详细信息。

（5）docker 性能测试

下面是我用mediaSDK软件测试的关于docker container性能的数据柱状图，有助于判断采用docker container方案的一个性能的变化趋势。

图1-9 docker container性能分析

如上图1-9所示，在创建1个容器时，container的性能接近宿主机转解码性能；随着创建的container数量的增多，其性能与宿主机相比会有一个下降，但是相对还是一个比较合理的性能区间，这一点可以从图中两个柱状图的高度差的扩大可以发现。通过这些数据可以发现采用docker的方案是可行的。

* + 1. 总体设计

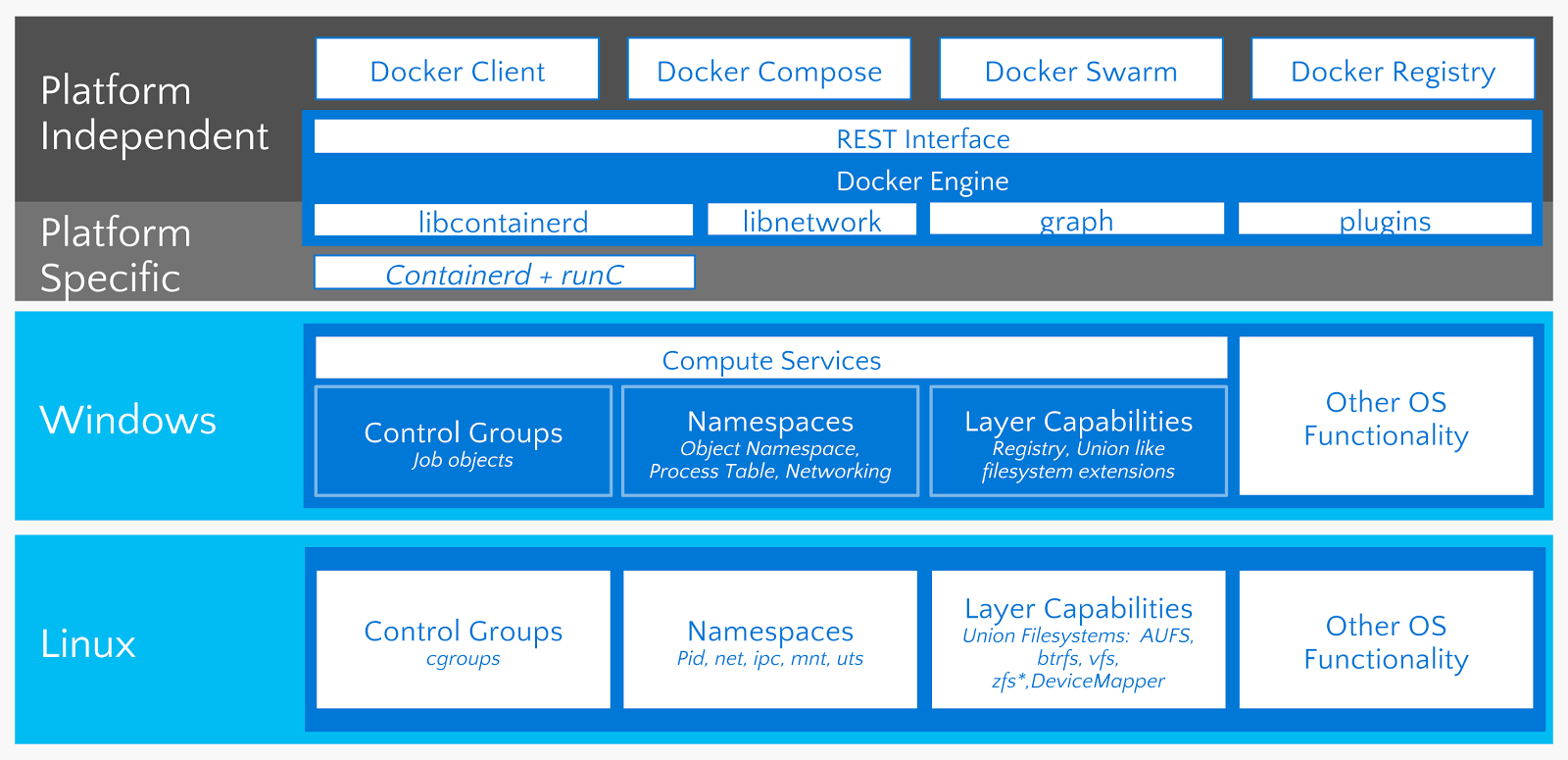


图1-10 总体架构图

Libcontainer提供原生go去用namespaces，cgroup，capabilities和filesystem access control去创建一个container。RunC在运行时是一个轻量级通用容器，提供一个命令行界面，根据OCI规范运行容器。Containerd对于容器来说是作为一个daemon，管理容器在宿主机系统下的整个生命周期的活动，比如image 传输和存储，容器创建和挂起等。Docker是一个用于build，ship和run container的一个开源产品。其流程如下图1-11所示。

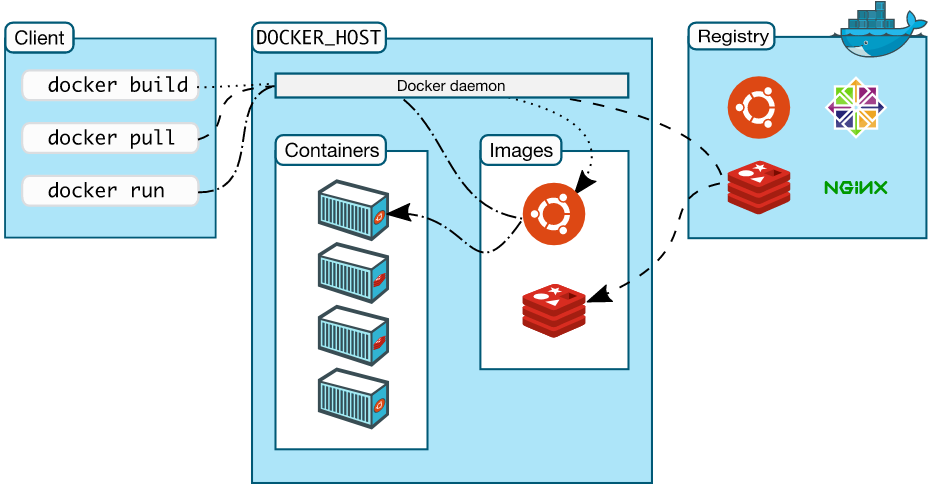


图1-11 client/server架构

添加kernel层的GPU memory和priority cgroup的控制，这是从系统层角度需要的设计；同时

* + 1. 详细设计

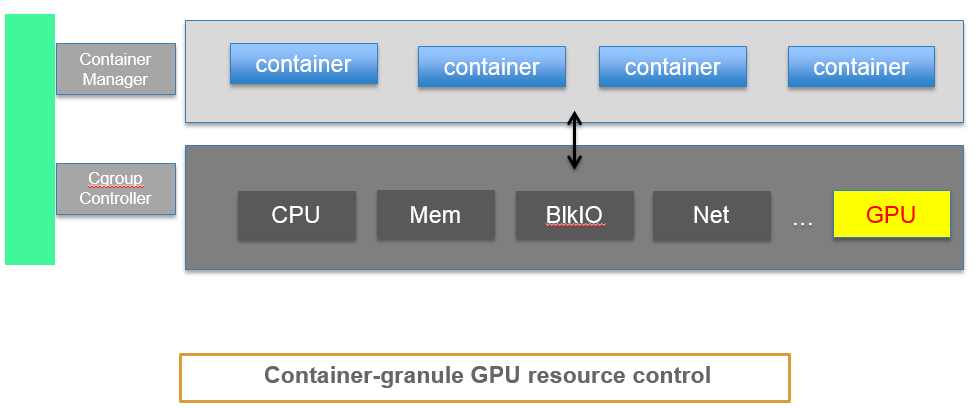


图1-12 详细设计图

在Docker所支持的从group类别基础上再添加针对GPU资源的控制，包括对kernel的GPU cgroup的添加和Docker Daemon及CLI的cgroup和参数添加。具体设计如下：

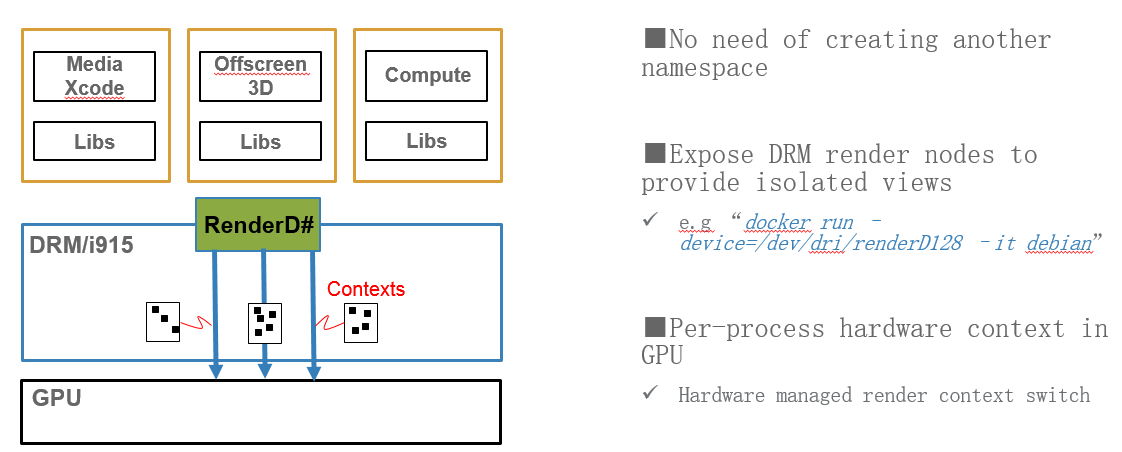


图1-13 资源控制层次图

需要一个新的子系统，gpu\_subsys，进行cgroup控制主要在三个方面需要，使用GPU百分比，工作任务的优先级以及最大分配的内存。在DRM/I915层需要支持任务调度过程中的GPU分配百分比和优先级，设定内存最大限制等。上层RunC里需要新的config.json文件，在该文件里面添加上关于GPU的参数定义。

* + 1. 已完成工作

1. 在kernel cgroup里添加GPU资源控制，包括GPU priority和最大memory控制；
2. 在docker server端添加GPU资源控制；
3. 在RunC里添加GPU资源控制；
4. 在docker client端的CLI添加GPU资源控制；
5. 编写关于用Docker在linux container里使能GPU的wiki；
6. 成功在container里面运行yami的测试样例。
   * 1. 分析与解释

如果采用volume plugin来进行容器创建前的driver版本检查的话，有一个最直接的问题需要考虑的是volume主要的范围还是一个数据的持久化存储和数据迁移与共享，在运行时主要涉及的还是一个数据处理的过程，没有参与执行脚本的倾向。这对我们组采用volume plugin来进行拓展检查容器内driver版本的功能产生的困扰。

### 1.2. 知识技能学习情况

1.2.1 开发环境说明

本组项目的开发是broadwell/skylake server上进行的，CPU为i5-5300U @2.3GHz（Max Speed 2.9GHz），4Core，16G Mem，GPU 900MHz，Turbo Ratio 57；软件环境为自己编译的ubuntu 4.12.0-rc3+版本；Docker选用的docker-ce 17.03版本；验证使用的mediaSDK和yami，驱动libva API 版本为0.39.0。

1.2.2 工具学习使用

在个人使用中最多使用的是MobaXterm Personal Edition工具，用于ssh连接远程服务器进行相关的操作。其经常使用功能如下：

（1）windows下支持多标签的终端

通过MobaXterm进行远程终端链接，你可以创建 SSH, Telnet, Rlogin, RDP, VNC, XDMCP, FTP, SFTP or 串口等链接，如图1-14所示。你的每次链接都会自动保存并且出现在左侧链接窗口中。

全能型开源远程终端：MobaXterm

图1-14 可支持的终端链接

对于本机，MobaXterm让你可以在windows下运行Unix命令，如:ls, cd, grep, awk, tail, cut, sed, wget, rsync, telnet, ssh, rlogin, rsh…等等Unix基本命令。而且MobaXterm有很多免费的插件可以实现更多的需求。

（2）图形化的SFTP 浏览器

当你链接一个ssh终端的时候，左侧窗口就会出现一个图形化的SFTP 浏览器，它可以让你通过安全的SFTP 链接拖放文件来管理远程服务器。不喜欢可以关闭。

### 

图1-15 操作界面

（3）分屏显示及多任务执行

MobaXterm支持多分屏显示，方便管理多台服务器，并且你可以仅输入一次，让一条命令同时在这些不同的服务器终端执行。

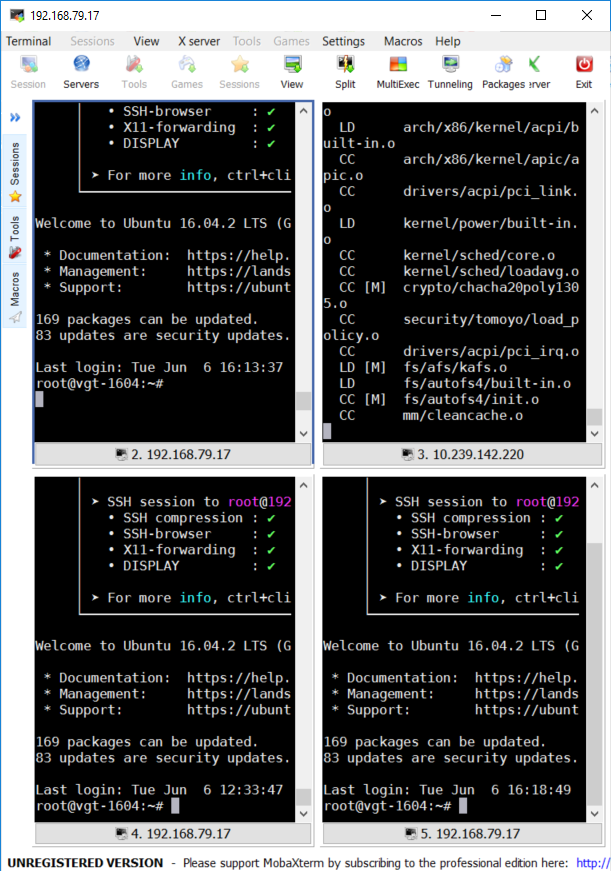


图1-16 分屏操作

1.2.4 其他学习内容

（1）mediaSDK安装与配置

当前版本的mediaSDK可以安装在CentOS7.2.1511版本或者添加了patch的linux4.4 kernel，其中CentOS上完全支持脚本自动化安装，如采用linux 4.4 kernel需要将软件包目录下的patch添加进代码中进行编译kernel，之后再安装libva等driver。

（2）yami安装与配置

Yami的安装需要libva-dev的支持

### 1.3. 职业素养学习培养

1.3.1 软件工程系统

1. 在评价复杂软件工程的解决方案的影响之前，我们需要知道工程理论知识是能够将数学、自然科学、工程基础和专业知识用于解决复杂工程问题的专用知识集合，因此自然而然需要我们做到：
2. 掌握数理及自然科学的基础知识
3. 掌握软件工程基础及专业知识

③ 运用解决复杂工程问题方法

1. 问题分析：能够应用数学、自然科学和工程科学的基本原理，识别、表达、并通过文献研究分析复杂工程问题，以获得有效结论。
2. 运用数理方法制定分析模型
3. 运用草稿、图表、流程表等工程方法描述问题

③ 通过文献研究分析复杂工程问题

1. 设计/开发解决方案：能够设计针对复杂工程问题的解决方案，设计满足特定需求的系统、单元（部件）或工艺流程，并能够在设计环节中体现创新意识，考虑社会、健康、安全、法律、文化以及环境等因素。

① 掌握软件生命周期要素，了解软件开发过程管理模型、ISO国际质量标准、CMMI软件评价标准

② 设计符合需求、具有创新性的解决方案的能力，同时识别社会、健康、安全、法律、文化以及环境等因素

③ 构建符合规格及限制的单元及系统

1. 研究：能够基于科学原理并采用科学方法对复杂工程问题进行研究，包括设计实验、分析与解释数据、并通过信息综合得到合理有效的结论。

① 设计实验项目的能力，包括资料收集、参数值范围设定、及统计分析方法的运用能力

② 分析与解释数据的能力，包括实施实验，并以预测或预期结果进行实验比较

1. 通过信息综合得到合理有效的结论的能力，包括从实验中呈现资料、解释资料或结果，并总结结论及给予建议，形成报告
2. 使用现代工具：能够针对复杂工程问题，开发、选择与使用恰当的技术、资源、现代工程工具和信息技术工具，包括对复杂工程问题的预测与模拟，并能够理解其局限性。
3. 了解现代工程工具和信息技术工具的使用方法
4. 掌握开发、选择与使用现代工具
5. 对复杂工程问题进行预测与模拟，并能够理解其局限性

1.3.2 职业道德规范

作为一名软件工程师，为了实现对工程系统质量，安全和利益的承诺目标，我觉得应当坚持以下八项原则：

1. 公众----作为软件工程师应当始终如一地以符合公众利益为目标；
2. 客户和雇主----在保持与公众利益一致的原则下，我们软件工程师应满足客户和雇主的最高利益；
3. 产品----作为软件工程师应当确保我们的产品和相关的改进符合可能达到的最高专业标准；
4. 判断----作为软件工程师在进行相关的专业判断时应该坚持正直，诚实和独立的原则；
5. 管理----作为软件工程的管理和领导人员在软件开发和维护的过程中，应当自觉遵守，应用并推动合乎道德规范的管理方法；
6. 专业----作为软件工程师应当自觉推动本行业所提倡的诚实，正直的道德规范，并自觉维护本行业的声誉，使软件行业更好的为公众利益所服务；
7. 同僚----作为软件工程师对其同僚应持平等互助和支持的态度；
8. 自身----作为软件工程师应终身不断地学习和实践其专业知识，并在学习和实践的过程中不断提高自身的道德规范素养。

### 1.4. 工程协作与交流情况

（阐述在实习执行过程中，针对特定的目标或问题，与工程项目组成员，包括与其他学科的成员合作并开展工作的情况。）

本项目在公司第二季度开始实施的，每周二中午都会进行各自工作的汇总和下一步工作的重点分析。本项目需要完成kernel添加和docker修改以及plugin的分析，我的导师负责的kernel部分的cgroup的添加；另一个工程师负责docker部分关于cgroup的添加和一些参数的添加；我负责研究docker plugin并参与部分讨论，验证一些方案可行性等等。

第一次讨论会主要关于整个项目的一个总体分析和可以采取的技术分析，我阐述了关于docker如何将device资源尤其是GPU device分配给container的过程分析；我们主要讨论了如下几点:

1. 研究如何给docker添加新的cgroup支持，是否需要通过插件或者修改docker server端代码；
2. 由于mediaSDK测试需要特定的kernel，所以讨论了将测试软件换成yami；
3. Docker 源码编译中的一些问题，比如需要设置代理等。

第一次讨论的核心就是怎么添加cgroup，这也成为我们后期工作的主线

## 2. 存在问题与解决方案

### 2.1. 存在的主要问题

（分析、总结和归纳实习执行过程中存在的主要问题与具体的应对措施，能通过具体事例来说明在心理素质培养以及应对项目风险与挑战的能力锻炼等方面是否有所收获。）

2.1.1 问题总结

对于docker-ce cgroup的控制流程了解得不完善(初期);

对于利用plugin进行版本检查的可能性判断不清楚;

2.1.2 解决方式

对于docker-ce cgroup流程问题采用runc来替代分析，runc也是一个可以用于管理容器的部分，里面对于cgroup的代码流程相对要清晰许多；对于plugin 检查driver版本的问题，因为plugin的类别是docker daemon规定好的，而采用volume plugin的话，如果要在创建容器之前检查起内部driver版本，则需要通过比如docker volume inspect方式获取其内部的驱动信息，而问题在于比如mediaSDK这种软件好像并没有显示的版本信息，只有通过运行小的测试来判断匹配与否。什么时候运行版本检测脚本？什么方式运行？这些仍然是我们面临的很大的疑惑和困难。

2.1.3 收获

通过本次项目，对kernel cgroup有了更加深入的理解，同时对docker和docker plugin的操作和实现原理有了一些了解，也收获了团队协作的精神品质和认真负责的职业操守。

### 2.2. 解决方案与可行性研究

2.2.1 尚未解决的问题

设计方案，用于在创建容器之前进行驱动版本检查，以避免因驱动版本问题造成的影响。

2.2.2 解决方案

解决方案一：静态检测法，获取driver lable进行直接比较；

解决方案二：动态检测法，通过运行标准测试用例，根据结果进行判断；

解决方案三：后台隐藏法，第一个起的container处于后台运行模式，在该模式下运行预置的脚本和测试用例，如果测试不通过，则执行退出container操作并删除container，显示相关driver错误。

解决方案四：插件+动态检测法，鉴于目前docker的支持的插件类别有一定的要求，而且volume

2.2.3 可行性分析

考虑到Docker的局限性和特殊，以及

nvidia 同类产品分析

Nvdia针对Docker Container推出了自己的解决方案—nvidia docker + nvidia docker plugin的形式，nvidia docker是在原生docker上封装了一层，对docker命令行接口进行一些替换和处理，如create和run命令；nvidia docker plugin则是充当一个daemon进程，用于发现host的驱动文件和GPU硬件，并对docker daemon发起的存储挂载请求做出响应。下面是关于nvidia docker对于驱动版本检查机制的分析，整体分析如图2-1所示。

1. 整体架构分析



图2-1 nvidia docker整体架构

Nvidia docker作为在原生docker之上的一层，对于create和run参数有着其自定义的处理步骤，之后才将处理结果传给原生docker进行处理。Nvidia docker与原生的docker交互发生在cudaSupported函数，其具体实现如图2-3所示。在nvidia docker获取image驱动版本的实现中考虑了remote和local两种情况，remote模式下采用json格式的文本传输获取CUDA版本；local模式下采用GetCUDAVersion函数实现CUDA版本的获取，该函数具体分析如图2-2所示。

1. 获取CUDA版本



图2-2 local模式获取cuda版本

在local模式下获取cuda版本，比较特殊的一步是driverGetVersion的本质是采用了go语言调用c语言是形式，引入cuda\_runtime\_api头文件，用go语言调用其cuda api里面的cudaDriverGetVersion函数来获取cuda版本。这里所获取到的版本将会用于和容器image里的lable版本进行比较得出驱动是否匹配的信息，以此决定是否创建一个容器。与Image里面的lable所进行比较的是该驱动所能支持的最大的CUDA版本。其具体流程如图2-3所示。

1. 获取lable版本



图2-3 获取image版本标签进行匹配

在编译CUDA image的Dockerfile里我们可以看见关于com.nvidia.cuda.

-version的定义，通过Lable可以指定image的版本，说明等；nvidia docker本质上还是调用了原生的docker的inspect命令，来获取image里面所定义的lable信息，docker inspect –-format 命令可以用于获取image或container里面的元数据，即这里所需要的com.nvidia.cuda.version标签信息。

2.2.4 结论

## 3. 前期任务完成度与后续实施计划

（前期任务完成度自我评价及后续工程环节的实施计划安排。除针对复杂工程问题的实施方案执行计划，还包括针对个人或职业发展需求的知识技能学习计划。）

### 3.1 前期任务完成度

对于前期的导师交给的任务都基本完成了，除了个别因为系统环境过于单薄，安装过程不详造成的问题，但这些均通过替换方案的方式得到了部分或间接的解决。

### 3.2 后续实施计划

3.2.1 实施方案计划

后续将着重研究采用插件式检查驱动版本的方案，根据目前的信息，Docker存储插件更多的表现出来的是在数据存储和共享方面的功能，至于如何让其通过运行某些标准测试用例来判断驱动版本还值得深思。特别是检查操作是在创建一个container之前需要完成的话，那么怎么将镜像里的驱动挂载到宿主机上

3.2.2 知识技能学习计划

剩下的时间还需要加强linux 内核学习，特别是namespace和cgroup的学习；此外，还需要复习一下操作系统的知识；为了加强对计算机体系结构的认识，需要

## 参考文献

[1] 王浩刚，聂在平.三维矢量散射积分方程中奇异性分析[J]. 电子学报，1999, 27(12): 68-71

[2] X. F. Liu, B. Z. Wang, W. Shao. A marching-on-in-order scheme for exact attenuation constant extraction of lossy transmission lines[C]. China-Japan Joint Microwave Conference Proceedings, Chengdu, 2006, 527-529

[3] 竺可桢.物理学[M]. 北京：科学出版社，1973, 56-60