# 深度解析Docker和OpenStack系统集成

2015 OpenStack技术大会-OpenStack与Docker集成-刘光亚 PPT详述

## Docker和OpenStack

OpenStack和Docker之间是很好的互补关系。Docker的出现能让IaaS层的资源使用得更加充分，因为Docker相对虚拟机来说更轻量，对资源的利用率会更加充分。如图1所示。

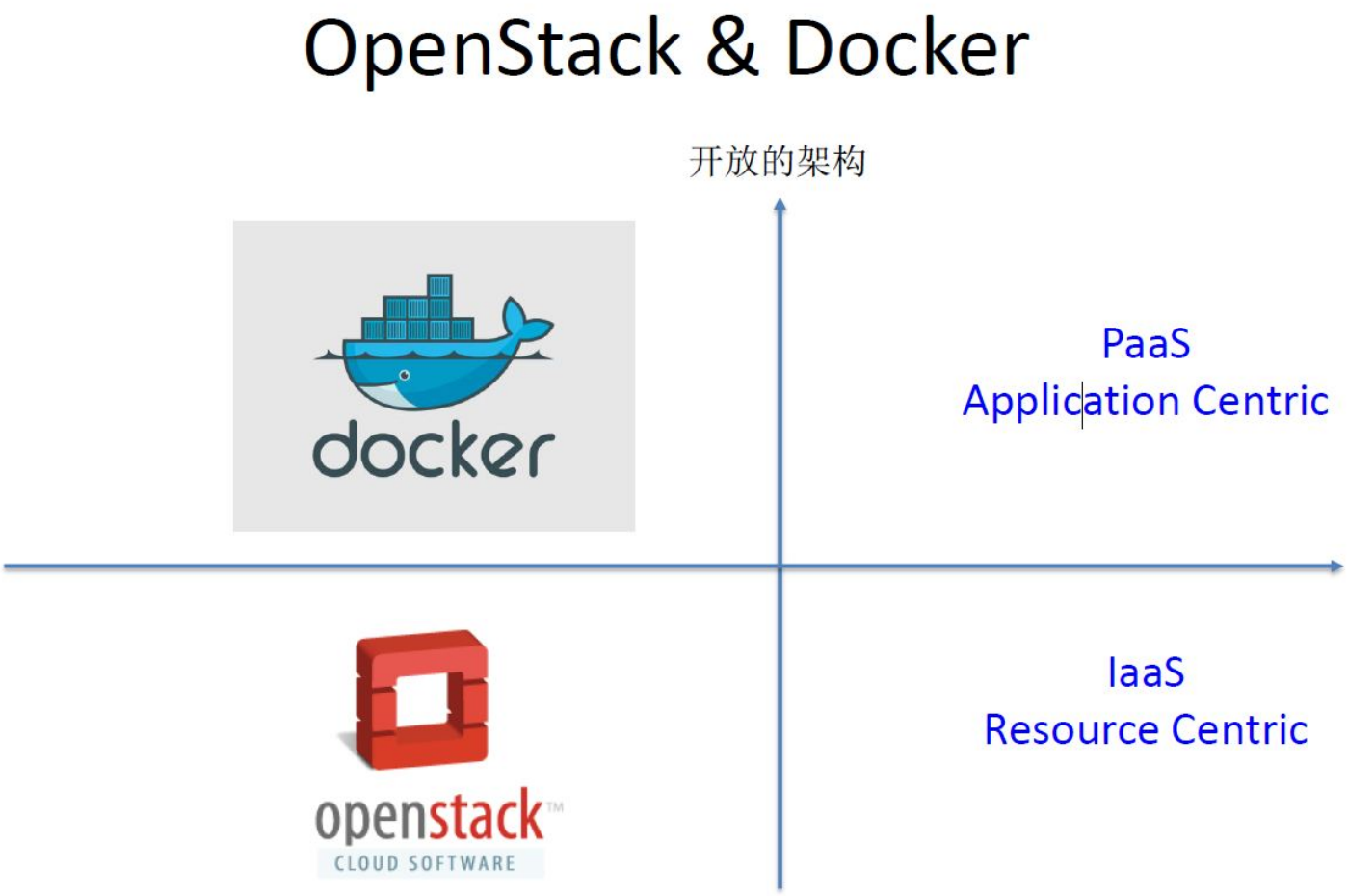



图1 OpenStack和Docker的关系

从图1可以看出，Docker主要针对Paas平台，是以应用为中心。OpenStack主要针对Iaas平台，以资源为中心，可以为上层的PaaS平台提供存储、网络、计算等资源。

## OpenStack层级

OpenStack中按照层来分级的一些项目，如图2所示。图2从下往上看：第一层是基础设施层，这一层主要包含Nova、Glance和Keystone，如果我们要想得到最基本的基础设施的服务，必须安装部署这三个项目。

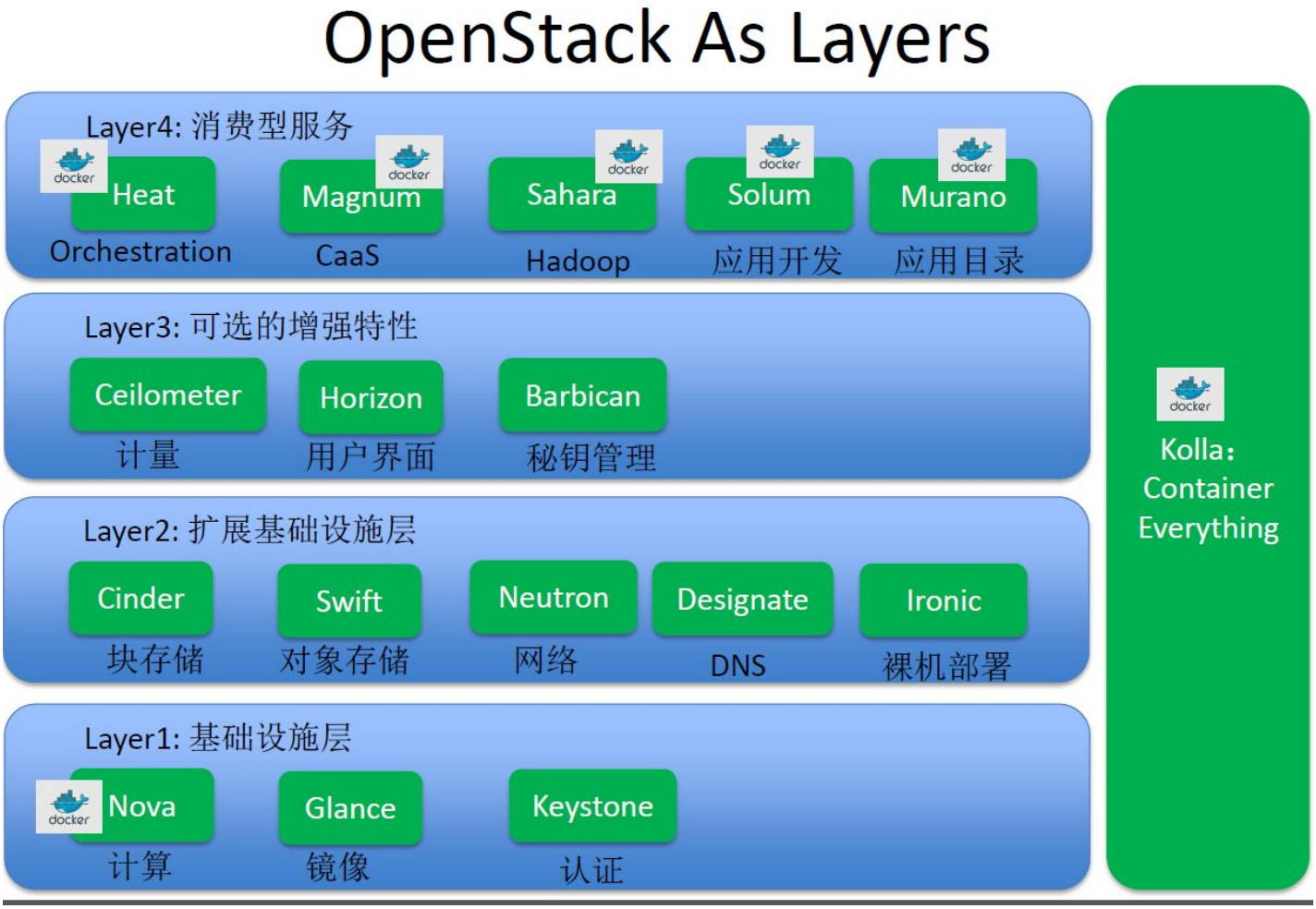



图2 OpenStack层级 &&& 图3 和Docker有关系的OpenStack项目

第二层是扩展基础设施层，这一层可以让我们得到更多跟基础设施相关的高级服务，主要包含Cinder、Swift、Neutron、Designate和Ironic等，其中Cinder提供块存储，Swift提供对象存储，Neutron提供网络服务，Designate提供DNS服务，Ironic提供裸机服务。

第三层是可选的增强特性，帮用户提供一些更加高级的功能，主要包含Ceilometer、Horizon和Barbican，其中Ceilometer提供监控、计量服务，Horizon提供用户界面，Barbican提供秘钥管理服务。

第四层主要是消费型服务，所谓的消费型服务，主要是指第四层的服务都需要通过使用前三层的服务来工作。第四层主要有Heat、Magnum、Sahara、Solum和Murano等，其中Heat主要提供orchestration服务，Magnum主要提供容器服务，Sahara主要提供大数据服务，我们可以通过Sahara很方便地部署Hadoop、Spark集群。Solum主要提供应用开发的服务，并且可以提供一些类似于CI/CD的功能。Muarno主要提供应用目录的服务，类似于App Store，就是用户可以把一些常用的应用发布出来供其他用户去使用。最右边是Kolla，Kolla的主要功能是容器化所有的OpenStack服务，便于OpenStack安装部署和升级。

OpenStack中和Docker有关系的项目如图3所示。

主要包括Nova、Heat、Magnum、Sahara、Solum、Murano和Kolla等。由图3得知，和Docker相关的大部分项目都在PaaS和SaaS层。以下主要分别说明Nova、Heat、Magnum、Murano和Kolla。

## Nova Docker Driver

如图4所示，这个Driver是OpenStack和Docker的第一次集成，主要是把Docker作为一种新的Hypervisor来处理，把所有的Container当成VM来处理。提供了一个Docker的Novacompute Driver，集成很简单，通过Docker REST API来操作Container。

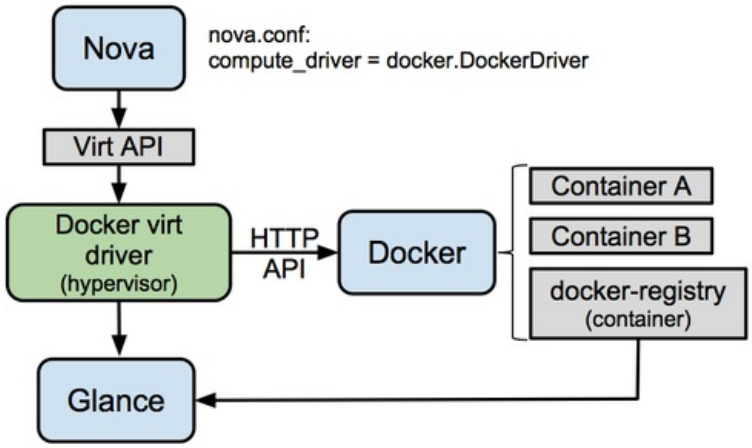


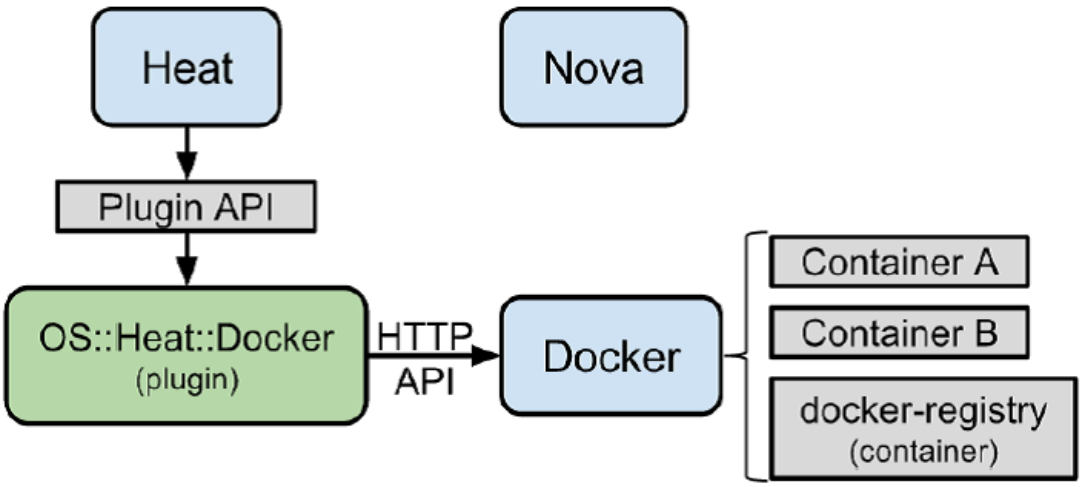

图4 Nova Docker Driver

这个Driver的优点是实现比较简单，只需要把Novacompute中的一些对虚拟机操作的常用接口实现就可以，现在主要支持创建、启动、停止、Pause、Unpause等虚拟机的基本操作。因为Nova Docker Driver也是一个Nova的Compute Driver，所以它可以像其他的Compute Driver一样使用OpenStack中的所有服务，包括使用Novascheduler来做资源调度，用Heat来做应用部署、服务发现、扩容缩容等，同时也可以通过和Neutron集成来管理Docker网络。也支持多租户，为不同的租户设置不同的Quota，做资源隔离。

它的缺点也很明显，因为Docker和虚拟机差别也挺大的，Docker还有一些很高级的功能是VM所没有的，像容器关联，就是使不同容器之间能够共享一些环境变量，来实现一些服务发现的功能，像K8S的Pod就是通过容器关联来实现的。另外一个是端口映射，K8S的Pod也使用了端口映射的功能，可以把一个Pod中的所有Container的Port都通过Net Container Export出去，便于和外界通信。还有一个是不同网络模式的配置，因为Docker的网络模式很多，包括Host模式、Container模式等等，以上的所有功能都是Nova Docker Driver不能实现的。

## Heat Docker Driver

因为Nova Docker Driver不能使用Docker的一些高级功能，所以社区就想了另一个方法，和Heat去集成，如图5所示。



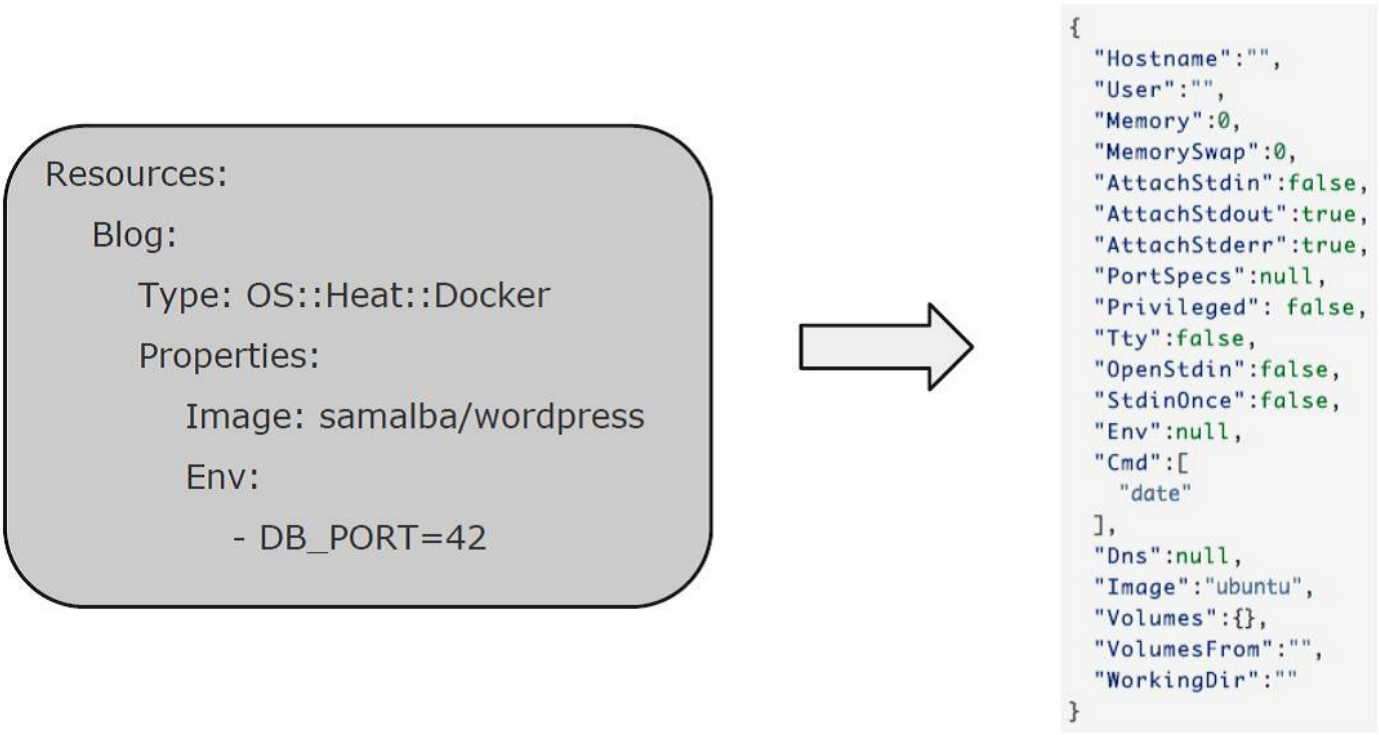


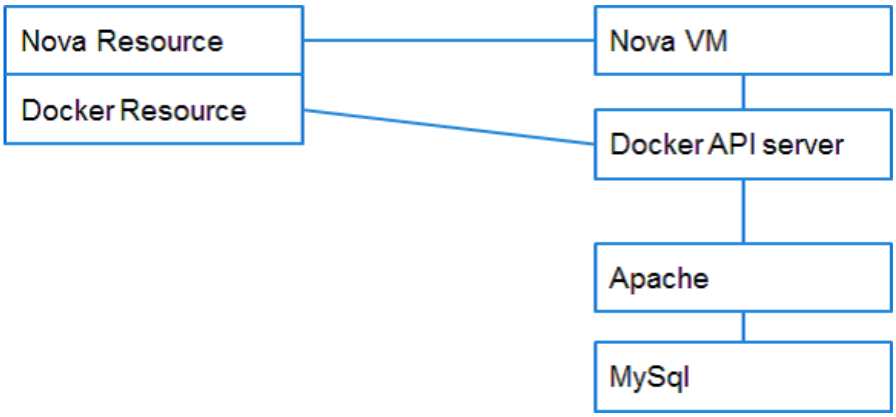

图5 Heat Docker Driver

因为Heat采用的也是插件，所以就在Heat实现了一个新的Resource，专门来和Docker集成。这个Heat插件是直接通过REST API和Docker交互的，不需要和Nova、Cinder和Neutron等来进行交互。

这个Driver的一个优点首先是它完全兼容Docker的API，因为我们可以在Heat Template里边去定义我们关心的参数，可以实现Docker的所有高级功能。另外因为和Heat集成了，所以默认就有了Multi-Tenant的功能，可以实现不同Docker应用之间的隔离。

但它的缺点也非常明显，因为它是Heat直接通过REST API和Docker交互的，所以Heat Docker Driver没有资源调度，用户需要在Template中指定需要在哪一台Docker服务器上去部署，所以这个Driver不适合大规模应用，因为没有资源调度。另外因为没有和Neutron去交互，所以网络管理也只能用Docker本身的网络管理功能来实现。

图6是使用Heat Docker Driver的一个很典型的应用场景。主要是通过Heat在虚拟机部署一个小规模的Docker Container应用，这个例子主要是一个Web Server的应用。



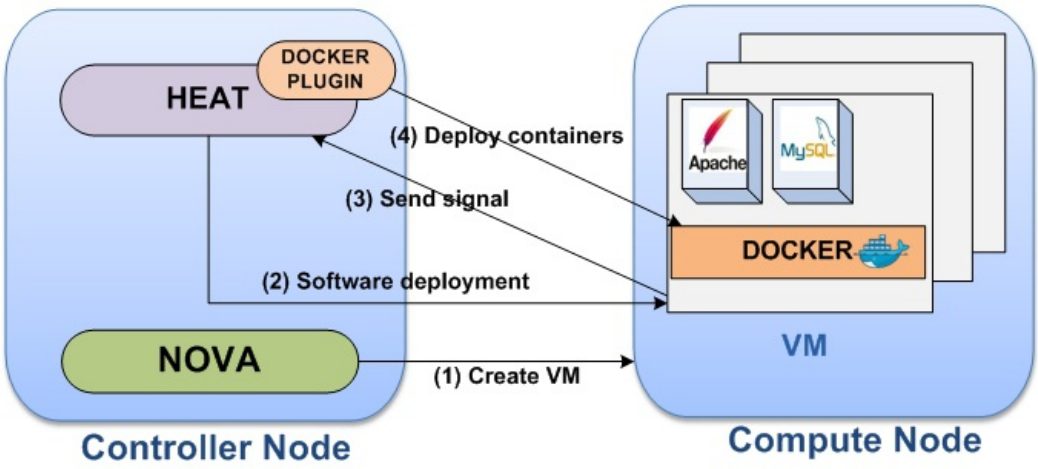



图6 使用Heat Docker driver一个典型应用场景

它的方法是在Heat Tempalte定义两种类型的Resource，一种是Nova Server，一种是Docker Container，Docker Container需要依赖Nova Server，就是必须Nova Server创建完后才能开始创建Docker Container。当用户在创建Nova Server的时候，需要在Nova Server上通过User Data安装Docker，Docker Container需要等Nova Server启动后，在Nova Server上创建Docker Container。这里有一个详细的例子，大家可以参考：http://techs.eNovance.com/7104/multi-tenant-Docker-with-openstack-heat

# Magnum

在OpenStack和Docker集成的过程中，我们发现从OpenStack现有的项目中，找不到一个很好的集成点，虽然和Nova、Heat都做了集成的尝试，但缺点很明显，所以社区就开始了一个新的专门针对Docker和OpenStack集成的项目Magnum，用来提供容器服务。Magnum是2014年在巴黎峰会后从11月开始做的，到现在5个月时间有27个Contributor，有700多个Commit，发展速度还可以。

Mangum的主要目的是提供Container服务的，它同时还可以和多个Docker集群管理系统集成，包括K8S、Swarm、CoreOS等。和这几个平台集成的主要原因是能让用户可以很方便地通过OpenStack云平台来集成K8S、CoreOS、Swarm这些已经很成型的Docker集群管理系统，促进Docker和OpenStack生态系统的融合。

首先来看Magnum的主要概念，如图7所示。

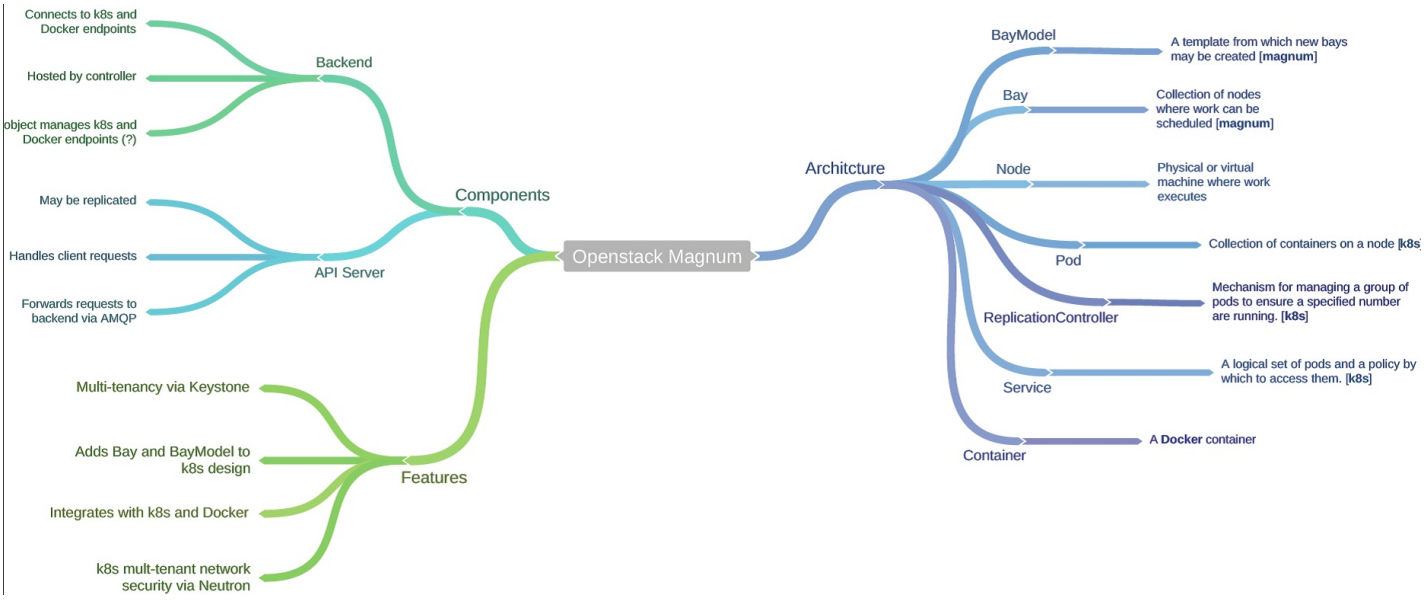



图7 Magnum的主要概念

在这个图的右边，主要有：Bay、Baymodel、Node、Pod、Service、Replication Controller和Container。

1. Bay在Magnum主要表示一个集群，现在通过Magnum可以创建K8S和Swarm的bay，也就是K8S和Swarm的集群。
2. Baymodel是Flavor的一个扩展，Flavor主要是定义虚拟机的规格，Baymodel主要是定义一个Docker集群的一些规格，例如这个集群的管理节点的Flavor、计算节点的Flavor、集群使用的Image等。

3. Node主要是指Bay中的某个节点。

Pod、Replication Controller和Service的意思和在K8S的意思是一样的，用户可以通过Magnum来和K8S集成，通过Magnum API来操作K8S集群。

1. Pod是Kubernetes最基本的部署调度单元，可以包含多个Container，逻辑上表示某种应用的一个实例。比如一个Web站点应用由前端、后端及数据库构建而成，这三个组件将运行在各自的容器中，那么我们可以创建三Pod，每个Pod运行一个服务。或者也可以将三个服务创建在一个Pod，这取决于用户的应用需求。

2. Service是Pod的路由代理抽象，用于解决Pod的高可用的问题。Service因为Pod的运行状态可动态变化(比如Pod的IP变了或者某个Pod被删除了等)，所以访问端不能以写死IP的方式去访问该Pod提供的服务。Service的引入旨在保证Pod的动态变化对访问端透明，访问端只需要知道Service的地址，由Service来提供代理。

3. Replication Controller是Pod的复制抽象，用于解决Pod的扩容缩容问题。通常，分布式应用为了性能或高可用性的考虑，需要复制多份资源，并且根据负载情况动态伸缩。通过Replication Controller，我们可以指定一个应用需要几份复制，Kubernetes将为每份复制创建一个Pod，并且保证实际运行Pod数量总是与该复制数量相等（例如，当前某个Pod宕机时，自动创建新的Pod来替换）。

4. Container就是某个Docker容器。

Magnum中的主要服务有两个，一个是Magnum API，一个是Magnum Conductor。

Magnum API和其他项目的API的功能是一样的，主要是处理Client的请求，将请求通过消息队列发送到backend，Magnum现在支持的backend有K8S、CoreOS、Swarm、Docker等，未来还会支持Rocket等Docker集群管理工具。

在Magnum，后台处理主要是通过Magnum Conductor来做的。Magnum Conductor的主要作用是将Client的请求转发到对应的Backend，通过对用户请求的解析，帮用户找到最合适的Backend，所以Magnum Conductor需要为每个API的请求找到合适的Backend处理Client的Request。

Magnum其实和Nova的架构很像，Nova主要提供IaaS服务，Magnum主要提供CaaS，Nova可以管理不同类型的Hypervisor，包括VMware、KVM、PowerVM等，Magnum也一样，它可以管理不同的Docker集群管理工具，包括K8S、Swarm、CoreOS等。最终目标是期望和Nova一样，能够让用户不用关心后台的Docker集群管理工具到底是什么，只管提请求，最终通过Magnum获得自己需要的Container。

从图7来看，Magnum很简单，主要是做了些集成工作，但其实Magnum对开发人员的要求也很高，因为Magnum需要和Heat、Ironic、Nova、K8S、CoreOS和Swarm等集成，所以需要开发人员对这些项目都有一定的了解，至少需要对这些项目的概念和主要特性都很清楚。

Magnum现在的一些主要特性，包括K8S as a Service、CoreOS as a Service、Swarm as a Service等。所以有这些功能，主要是因为有的客户已经有OpenStack集群了，现在想在OpenStack上运行一些以应用为中心的系统，像K8S、Swarm、Mesos等，通过这些系统为用户提供应用服务，所以用户就可以通过Magnum来实现它的需求，通过OpenStack提供Iaas服务，上层的K8S、Swarm等来提供Container服务。

另外，Magnum还实现了多租户的功能，可以将不同租户之间的Resouce隔离，每个租户可以用自己的Baymodel、Bay等对象，实现安全隔离。Magnum准备和K8S的开发人员合作，打造一个最好的OpenStack和K8S结合的项目。

Magnum Roadmap如图8所示。

Details in PPT

图8 Magnum Roadmap

第一个是Magnum Conductor的水平扩展，Magnum服务也是无状态的，所以可以水平扩展，一旦发现Conductor性能存在瓶颈时，可以通过启动一个新的Conductor来分担系统负载。

第二个是原生Docker集群的调度问题，Magnum现在只能管理单个的Docker的服务器，因为还没有一个原生的调度器能够让Magnum管理一个Docker集群，但可以和Swarm、Gantt或Mesos集成实现Docker集群资源调度的功能。现在经过讨论，可能会使用Swarm来做调度，因为通过Magnum部署完Swarm集群后，默认可以通过Swarm来管理Docker集群。

第三个是原生Docker集群的网络管理，现在大家的一致想法是在Docker server上通过Host模式来部署一个Container的l2 agent来管理Docker Server的网络。

第四个是Notification，其主要作用是便于追踪Magnum中所有对象的状态，尤其是当第三方和Magnum集成的时候，可以通过Notification来监控Magnum中的操作的对象。

最后一个是K8S深度集成，现在Magnum和K8S的集成是Magnum通过调用K8S命令行kubectl和K8S交互，这样做比较简单，但受到的限制很大，因为Magnum需要通过解析Kubectl的输出来判断每个调用是不是成功，但有时Kubectl并不能输出某个API的所有Output，所以Magnum关心的一些值通过Kubectl可能拿不到。现在有个BluePrint就是想通过K8Sclient使用REST API来和K8S交互，这样就可以拿到每个调用的全部输出，Magnum可以很方便地取得自己关心的输出，来做相应的操作。图8下面的Link是Magnum现在的所有的BluePrint。

# Murano

Murano是Mirantis贡献的，并且也进了OpenStack Namespace。也和K8S集成了，用户可以通过Murano使用K8S的功能，可以通过Murano部署Pod、Service、Replication Controller等。Murano主要是在OpenStack基础上提供应用目录服务。Muarno和Solum之间其实是有关系的，Solum主要是用来开发应用的，Solum把应用开发完后，可以通过Murano来发布。用户可以通过Murano挑选自己需要的应用服务，通过应用服务组合构建自己的应用。

Murano也是通过Heat部署应用，Kubernetes和Murano集成之后，现在K8S也已成为Murano的一个应用服务。

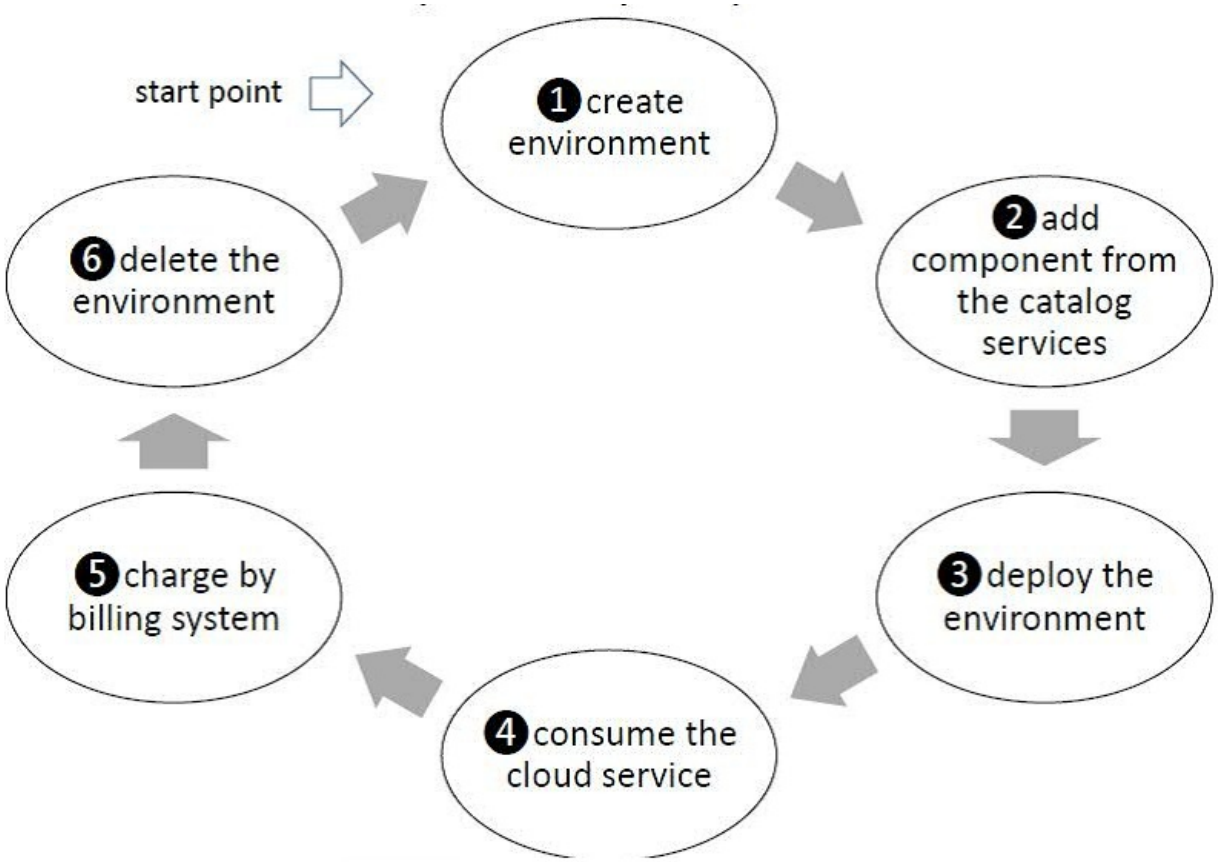



图9 Murano工作流

图9所示是Murano的一个工作流，可以看到它的使用很简单。首先创建用户的应用环境，然后为应用环境添加应用服务，接下来部署应用环境，应用环境会通过OpenStack的Heat来部署。

图10主要是说明怎样通过Murano部署一个K8S的Pod。

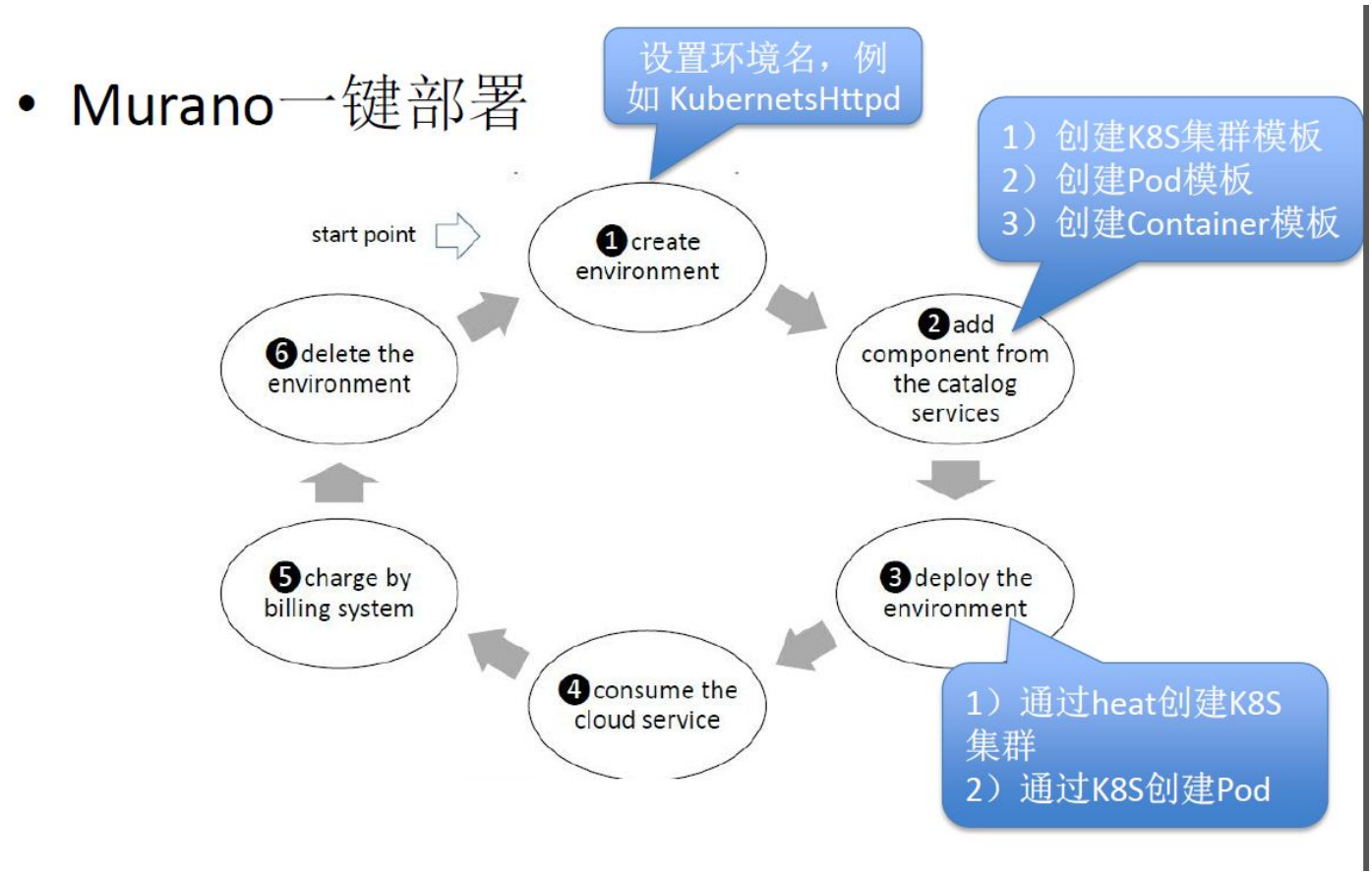



图10 怎样通过Murano部署一个K8S的Pod

Murano和K8S的集成主要在前三步，第一步是先创建一个K8S的环境；然后在第二步需要为我们创建的K8S环境添加一个应用服务，首先需要添加一个K8S的集群应用模板，然后在K8S集群的基础上添加一个Pod的应用；第三步是在第二步Pod的基础上，为Pod添加Container。这三步做完后，就可以部署环境了，最终Murano会调用Heat首先创建一个K8S的集群，然后K8S集群根据用户的需求创建Pod。所有的这些操作都可以在Murano的GUI上操作，非常简单。

## OpenStack Deployment With Docker

OpenStack现在的部署工具很多，包括RDO、Fuel、Chef、Triple-O等，但这些工具对OpenStack升级的支持不是很好。想在OpenStack升级的话，主要有两种方式：基于Image与基于Package 。

基于Package的更新方式通常不是原子的，升级过程中存在很多导致失败的原因，尤其是一些公共包依赖的问题，可能导致部分package更新失败的可能。基于Image的方式，更新是原子的。

Triple-O是一个很典型的通过镜像部署的例子，但是Triple-O升级OpenStack集群时，粒度太大，它通常情况下是同时对OpenStack多个服务去升级，这样对OpenStack集群影响比较大，会导致某些服务在一定时间段内不能被访问。

所以就有Kolla这样一个项目，来解决快速升级和回滚的问题。

## Kolla：Docker+OpenStack

Kolla的主要功能是使用Docker容器快速部署升级OpenStack服务。

Kolla的最终目标是为OpenStack的每一个服务都创建一个对应的Docker Image，通过Docker Image将升级的粒度减小到Service级别，从而使升级时，对OpenStack影响能达到最小，并且一旦升级失败，也很容易回滚。升级只需要三步：Pull新版本的容器镜像，停止老版本的容器服务，然后启动新版本容器。回滚也不需要重新安装包了，直接启动老版本容器服务就行，非常方便。

Kolla是通过Docker Compose来部署OpenStack集群的，现在主要是针对裸机部署的，所以在部署Docker Container时，默认的网络配置都是Host模式。所以Kolla的好处就非常明显了，将OpenStack升级的粒度细化到了Service级别，升级失败时，可以很容易回滚。

图11展示怎样通过Kolla去部署一个OpenStack集群。

Details in PPT

图11 通过Kolla部署OpenStack集群

可以看到首先需要启动一个管理节点，只需要通过一个命令就可以把管理节点部署完成，这个命令是调用Docker Compose来部署OpenStack的所有服务，然后我们可以在每一个计算节点上通过Docker Compose安装计算节点需要的服务，就能部署一个OpenStack集群。因为Kolla的Docker Image粒度很小，它针对每个OpenStack服务都有特定的Image，所以我们也可以通过Docker Run来操作某个具体的OpenStack服务。这个例子是通过Docker Image启动了Glance API。

图12是一个OpenStack compute的一个例子。

Details in *PPT*

图12 OpenStack Compute的一个例子

这是一个简单的YML文件，我们可以通过Docker compose使用这些YML文件部署OpenStack节点，可以看到这个YML文件包含Compute Data Image、Libvirt Image、Network Image、Nova API Image和Compute Image等。第一个Compute Data主要是为其他Container提供存储服务的，可以看到Libvirt和Novacompute的Container的存储都是从Compute Data来的，因为它们都有一个字段voluemes_from指向Compute Data这个Container。

## 如何选择？

OpenStack和Docker相关的项目这么多，该怎么去选择呢？简单说明如下。

1. 如果用户只是想将以前的VM Workload迁移到Docker Container，那么它可以使用Nova Docker Driver，一个很典型的例子是Sahara通过Heat调用Nova Docker Driver来创建Hadoop集群。
2. 如果用户想使用Docker的一些高级功能来部署一个小规模集群，那就可以考虑Heat Docker Driver。
3. 如果用户想通过OpenStack集成现有的一些Docker集群管理工具像K8S、Swarm来管理大规模的Docker集群，建议使用Magnum。另外因为Magnum也是基于Heat做的，所以默认也支持混合云的功能。

Murano和Docker的集成，主要体现在它提供了一个K8S的应用，用户可以通过这个K8S应用来管理Docker集群。但Murano和Docker的焦点不一样，Magnum主要提供容器服务，Murano主要提供应用目录服务。

最后的Kolla，主要是简化OpenStack的安装部署和升级的。

## Kilo的层级多租户管理

因为Docker的出现，使得IaaS层的计算密度进一步提升，这种密集型的计算对资源调度和运维的要求很高，需要一些比较高级的资源调度策略来提高资源利用率。Docker Container和虚拟机的最大区别是它轻量的特性，对于这种轻量性的容器技术，可以通过在资源调度中加入不同租户之间资源共享的机制来提高资源利用率。所以OpenStack在Kilo新加的层级租户的管理，这个功能可以理解为一种基本的资源共享，因为这个功能允许父账户和子账户之间共享资源。

图13是一个例子。

Details in *PPT* 图13 Kilo的层级多租户管理

我们可以看到每个租户的旁边都有一些值，以最顶层的租户为例，这个租户hard*limit=1000，used=100，reserved=100，allocated=70，它们的意思分别为：hard*limit是这个租户的资源上限；used是当前这个租户已经使用的资源；reserved是被这个租户预定的资源，即使当前的租户不用，也不能分给别的租户；allocated是当前租户分给子账户的资源，它的值是直接子账户的hard_limit的总和。我们可以看到ORG这个租户的直接子账户两个部门1和部门2，两个的hard*limt分别为300和400，所以ORG的allocated值是700，这四个值还隐藏了一个值，那就是free，当前账户的空闲资源，free是hard*limit减掉used、reserved和allocated，空闲资源有两个作用，一个是可以供本账户使用，还有一个是供子账户使用。我们可以看到ORG的空闲资源为1000-100-100-700=100，就是说它现在有100个空闲的资源，可以供自己或者子账户使用，就是给Dept-1或者Dept-2去用。这样就为OpenStack增加了不同层级租户之间的资源共享的功能。

这种形式有什么缺点呢？我们可以看到账户之间的关系只局限于父子账户，同一层级的账户之间还是没有关系，它们的资源也不能互相共享。我们看team11和team12，它们之间的资源不能共享，就是即使team11的资源都用完了，team12的资源一点没用，team11也不能从team12那边借资源过来，这样反映到企业就是，同一层级的部门之间资源不能共享，这是一个很大的短板，会导致资源不能被充分利用。我们可以把当前Kilo的这种层级租户资源共享理解为独占模式。

Details in *PPT*

图14 Kilo的层级租户模式

图14是当前Kilo的层级租户模式，现在共16个资源，T1和T2各独占8个，我们可以把每个资源看成是一个Docker Container，假定所有的Container规格都是一样的。就是说T1和T2最多可以创建8个Container。

T1要4个资源，因为它有8个，所以可以拿到4个，剩下4个空闲的；T2要12个资源，因为它只有8个，所以只能拿到8个资源。最终结果是T1有4个空闲的资源，T2缺少4个资源，这种独占策略的缺点就是资源不能共享，假如能把T1的这4个空闲的资源给T2用，就能达到最优的资源使用率。

针对Kilo即将实现的层级账户的缺点，有人正在建议OpenStack社区作出改进增加同一层级租户之间的资源共享，主要有两种形式：同一层级资源借入/借出和同一层级资源资源共享。

图15是同一层级的租户之间资源共享改进的第一个模式：同一层级租户间资源借入借出。

Ｄｅｔａｉｌｓ　ｉｎ　*ＰＰＴ* 图15 同一层级租户间资源借入/借出

借入借出是基于独占策略来改进的。这个例子和刚才的独占策略很像，从这个例子我们可以看到，租户T1和T2个独占8个资源，租户T2可以从租户T1借入4个资源。现在T1要4个，我有8个，可以拿到4个用着，然后有4个是空闲的，T2来了，要12个，先拿到自己独占的那8个，还需要4个，然后发现可以从T1那借4个，正好满足T2的请求。再进一步，就是假如这时候T1又来资源请求了，T1可以通过某种策略将借给T2的那4个要回来。

图16是同一层级的租户之间资源共享改进另一种模式：同一层级租户之间资源共享。

Ｄｅｔａｉｌｓ　ｉｎ　*ＰＰＴ*

图16 同一层级租户之间资源共享

现在同一层的租户之间的资源是不能共享的，所以我们需要让同一层级租户之间的资源能够共享来提升系统的资源使用率。所以我们现在可以引入同一层级的租户之间的共享模式。

举例来说，现在有两个租户，T1和T2，都不独占任何资源，共享所有资源，并且T1和T2的共享比例为1:1，现在一共有16个资源，这16个资源都是共享的，T1和T2的共享比例为1:1，所以T1和T2都deserve 8个资源，但这8个并不是被T1或者T2独占的，而是共享的。

也就是说，现在T1和T2都有资源请求了，T1要4个，T1现在deserve 8个，那就可以直接去拿4个资源。T2来了，T2要12个，但是T2只deserve 8个，那T2就先拿到deserve的这8个，然后它发现share pool里边还有4个资源没人用，那它就可以把这4个再拿过来，这样T1和T2的请求就都满足了。

这种借入借出模式和共享模式也可以同时去使用，我们可以把一部分资源设置成独占的，一部分设置为共享的，给用户更大的空间来配置它的资源规划。