美团点评技术团队

美团点评Docker容器管理平台

郑坤 ·2017-01-23 19:19

本文是郑坤根据第14期美团点评技术沙龙“你不知道的美团云”(http://www.huodongxing.com/event/1357797128200)演讲内容整理而成，已发表在《程序员》杂志2017年1月刊。

美团点评容器平台简介

本文介绍美团点评的Docker容器集群管理平台（以下简称“容器平台”）。该平台始于2015年，是基于美团云的基础架构和组件而开发的Docker容器集群管理平台。目前该平台为美团点评的外卖、酒店、到店、猫眼等十几个事业部提供容器计算服务，承载线上业务数百个，日均线上请求超过45亿次，业务类型涵盖Web、数据库、缓存、消息队列等。

为什么要开发容器管理平台

作为国内大型的O2O互联网公司，美团点评业务发展极为迅速，每天线上发生海量的搜索、推广和在线交易。在容器平台实施之前，美团点评的所有业务都是运行在美团私有云提供的虚拟机之上。随着业务的扩张，除了对线上业务提供极高的稳定性之外，私有云还需要有很高的弹性能力，能够在某个业务高峰时快速创建大量的虚拟机，在业务低峰期将资源回收，分配给其他的业务使用。美团点评大部分的线上业务都是面向消费者和商家的，业务类型多样，弹性的时间、频度也不尽相同，这些都对弹性服务提出了很高的要求。在这一点上，虚拟机已经难以满足需求，主要体现以下两点。

第一，虚拟机弹性能力较弱。使用虚拟机部署业务，在弹性扩容时，需要经过申请虚拟机、创建和部署虚拟机、配置业务环境、启动业务实例这几个步骤。前面的几个步骤属于私有云平台，后面的步骤属于业务工程师。一次扩容需要多部门配合完成，扩容时间以小时计，过程难以实现自动化。如果可以实现自动化“一键快速扩容”，将极大地提高业务弹性效率，释放更多的人力，同时也消除了人工操作导致事故的隐患。

第二，IT成本高。由于虚拟机弹性能力较弱，业务部门为了应对流量高峰和突发流量，普遍采用预留大量机器和服务实例的做法。即先部署好大量的虚拟机或物理机，按照业务高峰时所需资源做预留，一般是非高峰时段资源需求的两倍。资源预留的办法带来非常高的IT成本，在非高峰时段，这些机器资源处于空闲状态，也是巨大的浪费。

段，这些优质资源处于空闲状态，也是巨大的浪费。

由于上述原因，美团点评从2015年开始引入Docker，构建容器集群管理平台，为业务提供高性能的弹性伸缩能力。业界很多公司的做法是采用Docker生态圈的开源组件，例如Kubernetes、Docker Swarm等。我们结合自身的业务需求，基于美团云现有架构和组件，实践出一条自研Docker容器管理平台之路。我们之所以选择自研容器平台，主要出于以下考虑。

快速满足美团点评的多种业务需求

美团点评的业务类型非常广泛，几乎涵盖了互联网公司所有业务类型。每种业务的需求和痛点也不尽相同。例如一些无状态业务（例如Web），对弹性扩容的延迟要求很高；数据库，业务的master节点，需要极高的可用性，而且还有在线调整CPU，内存和磁盘等配置的需求。很多业务需要SSH登陆访问容器以便调优或者快速定位故障原因，这需要容器管理平台提供便捷的调试功能。为了满足不同业务部门的多种需求，容器平台需要大量的迭代开发工作。基于我们所熟悉的现有平台和工具，可以做到“多快好省”地实现开发目标，满足业务的多种需求。

从容器平台稳定性出发，需要对平台和Docker底层技术有更高的把控能力

容器平台承载美团点评大量的线上业务，线上业务对SLA可用性要求非常高，一般要达到99.99%，因此容器平台的稳定性和可靠性是最重要的指标。如果直接引入外界开源组件，我们将面临3个难题：1. 我们需要摸熟开源组件，掌握其接口、评估其性能，至少要达到源码级的理解；2. 构建容器平台，需要对这些开源组件做拼接，从系统层面不断地调优性能瓶颈，消除单点隐患等；3. 在监控、服务治理等方面要和美团点评现有的基础设施整合。这些工作都需要极大的工作量，更重要的是，这样搭建的平台，在短时间内其稳定性和可用性都难以保障。

避免重复建设私有云

美团私有云承载着美团点评所有的在线业务，是国内规模最大的私有云平台之一。经过几年的经营，可靠性经过了公司海量业务的考验。我们不能因为要支持容器，就将成熟稳定的私有云搁置一旁，另起炉灶再重新开发一个新的容器平台。因此从稳定性、成本考虑，基于现有的私有云来建设容器管理平台，对我们来说是最经济的方案。

美团点评容器管理平台架构设计

我们将容器管理平台视作一种云计算模式，云计算的架构同样适用于容器。如前所述，容器平台的架构依托于美团私有云现有架构，其中私有云的大部分组件可以直接复用或者经过少量扩展开发。容器平台架构如下图所示。

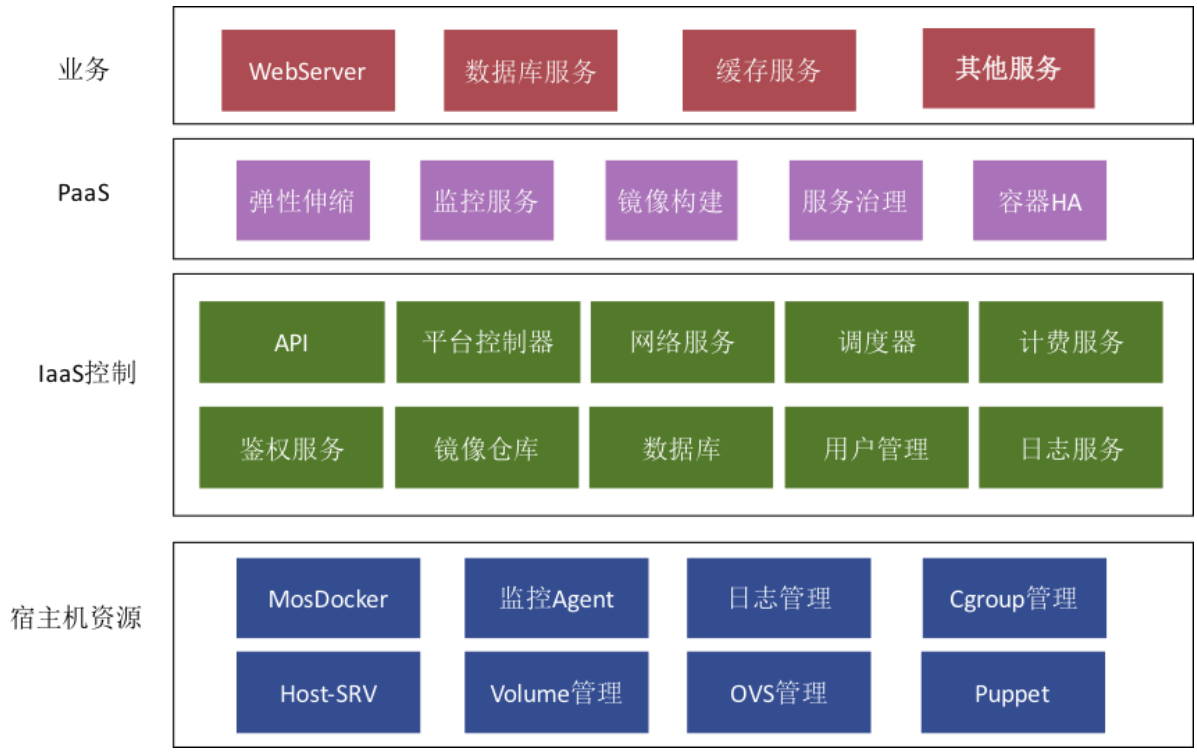

图 1. 美团点评容器管理平台架构

可以看出，容器平台整体架构自上而下分为业务层、PaaS层、IaaS控制层及宿主机资源层，这与美团云架构基本一致。

业务层：代表美团点评使用容器的业务线，他们是容器平台的最终用户。
PaaS层：使用容器平台的HTTP API，完成容器的编排、部署、弹性伸缩，监控、服务治理等功能，对上面的业务层通过HTTP API或者Web的方式提供服务。
IaaS控制层：提供容器平台的API处理、调度、网络、用户鉴权、镜像仓库等管理功能，对PaaS提供HTTP API接口。
宿主机资源层：Docker宿主机集群，由多个机房，数百个节点组成。每个节点部署HostServer、Docker、监控数据采集模块，Volume管理模块，OVS网络管理模块，Cgroup管理模块等。
容器平台中的绝大部分组件是基于美团私有云已有组件扩展开发的，例如API，镜像仓库、平台控制器、HostServer、网络管理模块，下面将分别介绍。

API

API是容器平台对外提供服务的接口，PaaS层通过API来创建、部署云主机。我们将容器和虚拟机看作两种不同的虚拟化计算模型，可以用统一的API来管理。即虚拟机等同于set（后面将详细介绍），磁盘等同于容器。这个思路有两点好处：1. 业务用户不需要改变云主机的使用逻辑，原来

给了，磁盘等同于容器。这个思路有两点好处：1. 业务用户不需要改变云主机的使用逻辑，原来基于虚拟机的业务管理流程同样适用于容器，因此可以无缝地将业务从虚拟机迁移到容器之上；

2. 容器平台API不必重新开发，可以复用美团私有云的API处理流程
创建虚拟机流程较多，一般需要经历调度、准备磁盘、部署配置、启动等多个阶段，平台控制器和Host-SRV之间需要很多的交互过程，带来了一定量的延迟。容器相对简单许多，只需要调度、部署启动两个阶段。因此我们对容器的API做了简化，将准备磁盘、部署配置和启动整合成一步完成，经简化后容器的创建和启动延迟不到3秒钟，基本达到了Docker的启动性能。

Host-SRV

Host-SRV是宿主机上的容器进程管理器，负责容器镜像拉取、容器磁盘空间管理、以及容器创建、销毁等运行时的管理工作。

镜像拉取：Host-SRV接到控制器下发的创建请求后，从镜像仓库下载镜像、缓存，然后通过Docker Load接口加载到Docker里。

容器运行时管理：Host-SRV通过本地Unix Socker接口与Docker Daemon通信，对容器生命周期的控制，并支持容器Logs、exec等功能。

容器磁盘空间管理：同时管理容器Rootfs和Volume的磁盘空间，并向控制器上报磁盘使用量，调度器可依据使用量决定容器的调度策略。

Host-SRV和Docker Daemon通过Unix Socket通信，容器进程由Docker-Containerd托管，所以Host-SRV的升级发布不会影响本地容器的运行。

镜像仓库

容器平台有两个镜像仓库：

- **Docker Registry:** 提供Docker Hub的Mirror功能，加速镜像下载，便于业务团队快速构建业务镜像；
- **Glance:** 基于Openstack组件Glance扩展开发的Docker镜像仓库，用以托管业务部门制作的Docker镜像。

镜像仓库不仅是容器平台的必要组件，也是私有云的必要组件。美团私有云使用Glance作为镜像仓库，在建设容器平台之前，Glance只用来托管虚拟机镜像。每个镜像有一个UUID，使用Glance API和镜像UUID，可以上传、下载虚拟机镜像。Docker镜像实际上是由一组子镜像组成，每个子镜像有独立的ID，并带有一个Parent ID属性，指向其父镜像。我们稍加改造了一下Glance，对每个Glance镜像增加Parent ID的属性，修改了镜像上传和下载的逻辑。经过简单扩展，使Glance具有托管Docker镜像的能力。通过Glance扩展来支持Docker镜像有以下优点：

- 可以使用同一个镜像仓库来托管Docker和虚拟机的镜像，降低运维管理成本；
- Glance已经十分成熟稳定，使用Glance可以减少在镜像管理上踩坑；
- 使用Glance可以使Docker镜像仓库和美团私有云“无缝”对接，使用同一套镜像API，可以同时支持虚拟机

和Docker镜像上传、下载，支持分布式的存储后端和多租户隔离等特性；

- Glance UUID和Docker Image ID是一一对应的关系，利用这个特性我们实现了Docker镜像在仓库中的唯一性，避免冗余存储。

可能有人疑问，用Glance做镜像仓库是“重新造轮子”。事实上我们对Glance的改造只有200行左右的代码。Glance简单可靠，我们在很短的时间就完成了镜像仓库的开发上线，目前美团点评已经托管超过16,000多个业务方的Docker镜像，平均上传和下载镜像的延迟都是秒级的。

高性能、高弹性的容器网络

网络是十分重要的，又有技术挑战性的领域。一个好的网络架构，需要有高网络传输性能、高弹性、多租户隔离、支持软件定义网络配置等多方面的能力。早期Docker提供的网络方案比较简单，只有None、Bridge、Container和Host这四种网络模式，也没有用户开发接口。2015年Docker在1.9版本集成了Libnetwork作为其网络的解决方案，支持用户根据自身需求，开发相应的网络驱动，实现网络功能自定义的功能，极大地增强了Docker的网络扩展能力。

从容器集群系统来看，只有单宿主机的网络接入是远远不够的，网络还需要提供跨宿主机、机架和机房的能力。从这个需求来看，Docker和虚拟机来说是共通的，没有明显的差异，从理论上也可以用同一套网络架构来满足Docker和虚拟机的网络需求。基于这种理念，容器平台在网络方面复用了美团云网络基础架构和组件。

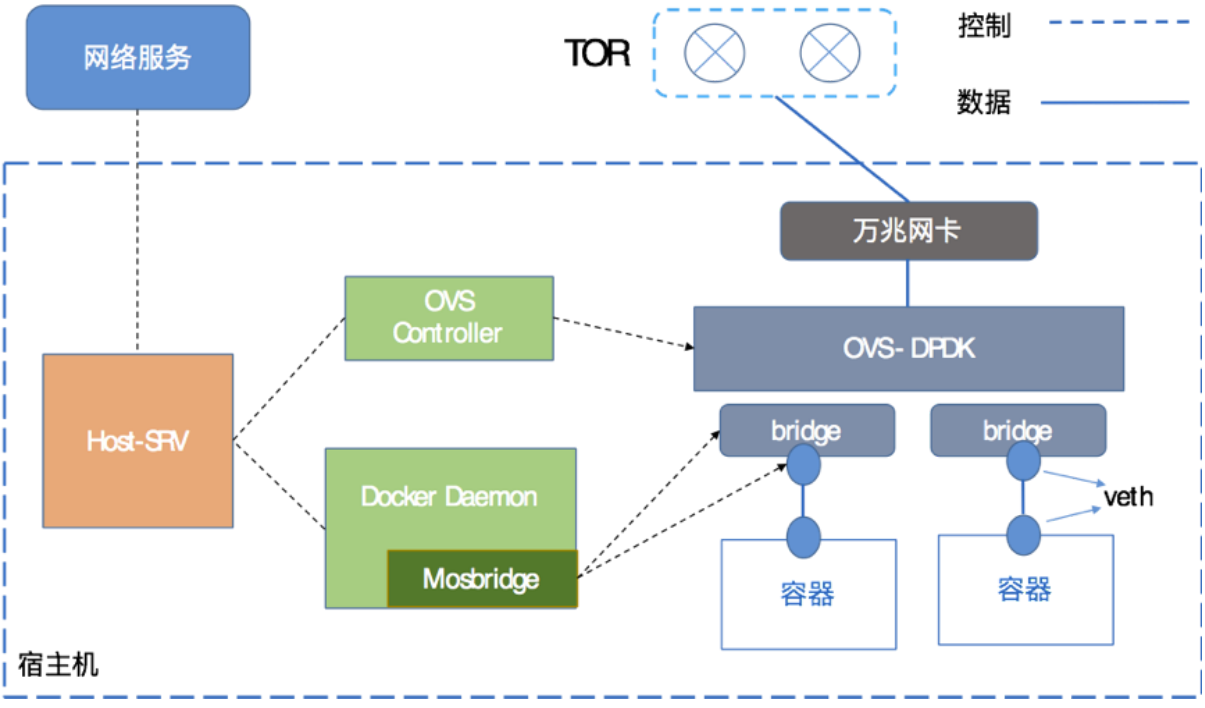

图 2. 美团点评容器平台网络架构

数据平面: 我们采用万兆网卡，结合OVS-DPDK方案，并进一步优化单流的转发性能，将几个CPU核绑定给OVS-DPDK转发使用，只需要少量的计算资源即可提供万兆的数据转发能力。OVS-DPDK和容器所使用的CPU完全隔离，因此也不影响用户的计算资源。

控制平面: 我们使用OVS方案。该方案是在每个宿主机上部署一个自研的软件Controller，动态接收网络服务下发的网络规则，并将规则进一步下发至OVS流表，决定是否对某网络流放行。

MosBridge

在MosBridge之前，我们配置容器网络使用的是None模式。所谓None模式也就是自定义网络的模式，配置网络需要如下几步：

1. 在创建容器时指定—net=None，容器创建启动后没有网络；
2. 容器启动后，创建eth-pair；
3. 将eth-pair一端连接到OVS Bridge上；
4. 使用nsenter这种Namespace工具将eth-pair另一端放到容器的网络Namespace中，然后改名、配置IP地址和路由。

然而，在实践中，我们发现None模式存在一些不足：

- 容器刚启动时是无网络的，一些业务在启动前会检查网络，导致业务启动失败；
- 网络配置与Docker脱离，容器重启后网络配置丢失；
- 网络配置由Host-SRV控制，每个网卡的配置流程都是在Host-SRV中实现的。以后网络功能的升级和扩展，例如对容器添加网卡，或者支持VPC，会使Host-SRV越来越难以维护。

为了解决这些问题，我们将眼光投向Docker Libnetwork。Libnetwork为用户提供了可以开发Docker网络的能力，允许用户基于Libnetwork实现网络驱动来自定义其网络配置的行为。就是说，用户可以编写驱动，让Docker按照指定的参数为容器配置IP、网关和路由。基于Libnetwork，我们开发了MosBridge – 适配美团云网络架构的Docker网络驱动。在创建容器时，需要指定容器创建参数—net=mosbridge，并将IP地址、网关、OVS Bridge等参数传给Docker，由MosBridge完成网络的配置过程。有了MosBridge，容器创建启动后便有了网络可以使用。容器的网络配置也持久化在MosBridge中，容器重启后网络配置也不会丢失。更重要的是，MosBridge使Host-SRV和Docker充分解耦，以后网络功能的升级也会更加方便。

解决Docker存储隔离性的问题

业界许多公司使用Docker都会面临存储隔离性的问题。就是说Docker提供的数据存储的方案是Volume，通过mount bind的方式将本地磁盘的某个目录挂载到容器中，作为容器的“数据盘”使用。这种本地磁盘Volume的方式无法做到容量限制，任何一个容器都可以不加限制地向

盘一使用。这种本地磁盘Volume的方式无法做到容量限制，任何一个容器都可以不加限制地向Volume写数据，直到占满整个磁盘空间。

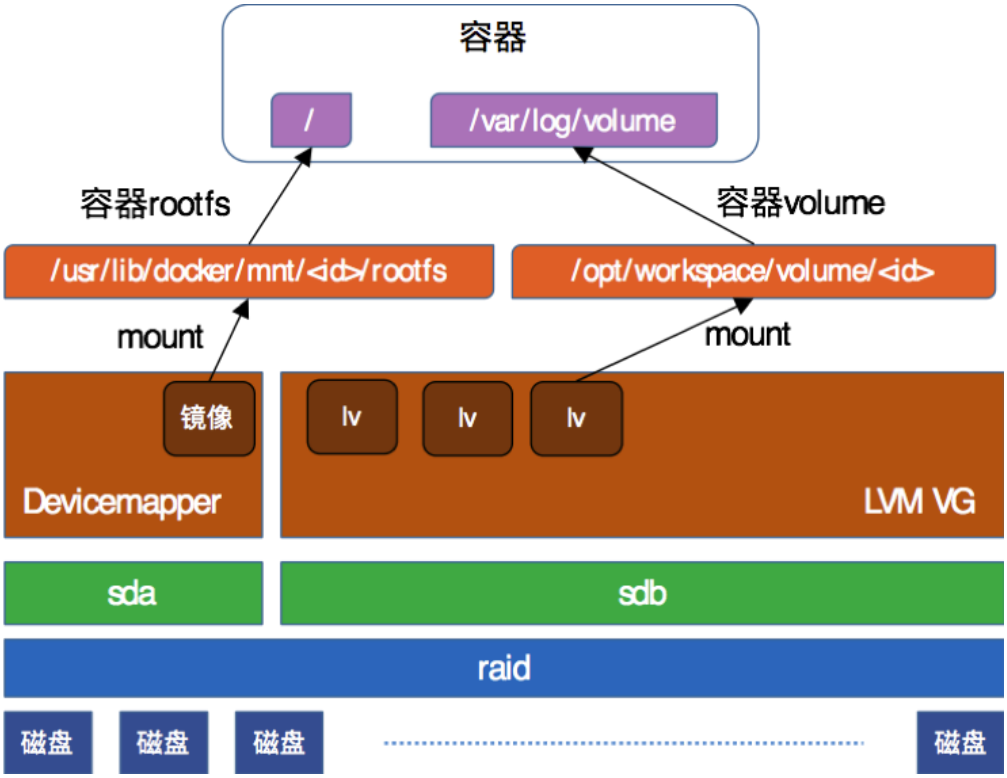

图 3. LVM-Volume方案

针对这一问题，我们开发了LVM Volume方案。该方案是在宿主机上创建一个LVM VG作为Volume的存储后端。创建容器时，从VG中创建一个LV当作一块磁盘，挂载到容器里，这样Volume的容量便由LVM加以强限制。得益于LVM机强大的管理能力，我们可以做到对Volume更精细、更高效的管理。例如，我们可以很方便地调用LVM命令查看Volume使用量，通过打标签的方式实现Volume伪删除和回收站功能，还可以使用LVM命令对Volume做在线扩容。值得一提地是，LVM是基于Linux内核Devicemapper开发的，而Devicemapper在Linux内核的历史悠久，早在内核2.6版本时就已合入，其可靠性和IO性能完全可以信赖。

适配多种监控服务的容器状态采集模块

容器监控是容器管理平台极其重要的一环，监控不仅仅要实时得到容器的运行状态，还需要获取容器所占用的资源动态变化。在设计实现容器监控之前，美团点评内部已经有了许多监控服务，例如Zabbix、Falcon和CAT。因此我们不需要重新设计实现一套完整的监控服务，更多地是考虑如何高效地采集容器运行信息，根据运行环境的配置上报到相应的监控服务上。简单来说，我们只需要考虑实现一个高效的Agent，在宿主机上可以采集容器的各种监控数据。这里需要考虑两点：

1. 监控指标多，数据量大，数据采集模块必须高效率；

2. 监控的低开销，同一个宿主机可以跑几十个，甚至上百个容器，大量的数据采集、整理和上报过程必须低开销。

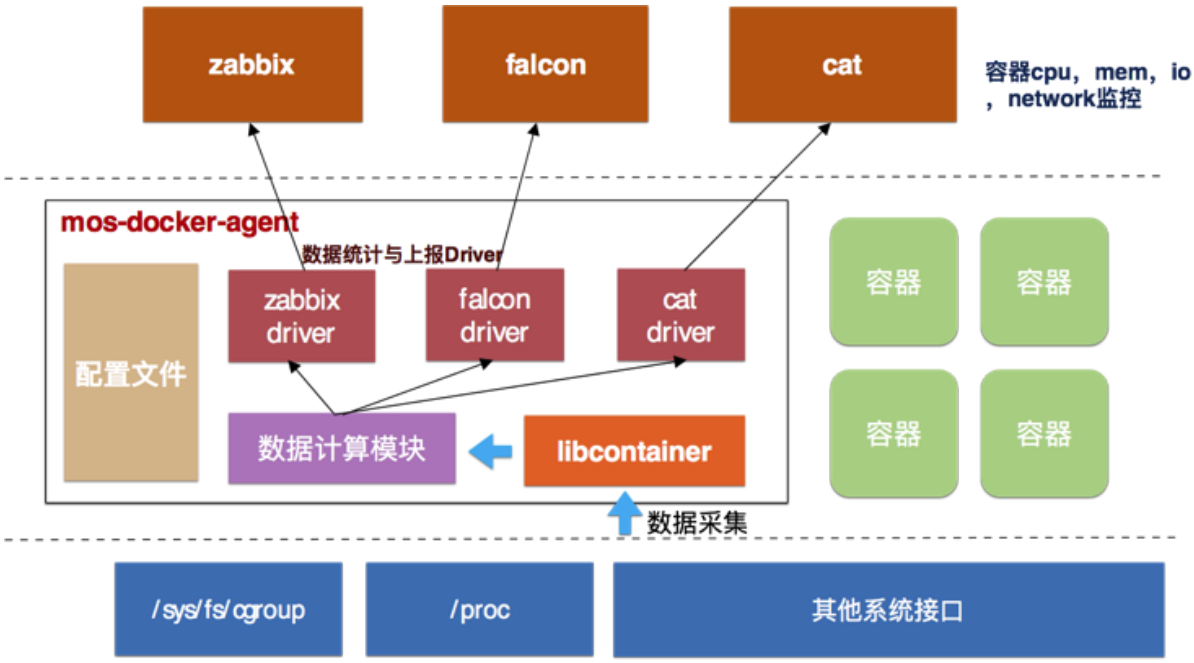

图 4. 监控数据采集方案

针对业务和运维的监控需求，我们基于Libcontainer开发了**Mos-Docker-Agent**监控模块。该模块从宿主机proc、CGroup等接口采集容器数据，经过加工换算，再通过不同的监控系统driver上报数据。该模块使用GO语言编写，既可以高效率，又可以直接使用Libcontainer。而且监控的数据采集和上报过程不经过Docker Daemon，因此不会加重Daemon的负担。

在监控配置这块，由于监控上报模块是插件式的，可以高度自定义上报的监控服务类型，监控项配置，因此可以很灵活地适应不同的监控场景的需求。

支持微服务架构的设计

近几年，微服务架构在互联网技术领域兴起。微服务利用轻量级组件，将一个大型的服务拆解为多个可以独立封装、独立部署的微服务实例，大型服务内在的复杂逻辑由服务之间的交互来实现。

美团点评的很多在线业务是微服务架构的。例如美团点评的服务治理框架，会为每一个在线服务配置一个服务监控Agent，该Agent负责收集上报在线服务的状态信息。类似的微服务还有许多。对于这种微服务架构，使用Docker可以有以下两种封装模式。

1. 将所有微服务进程封装到一个容器中。但这样使服务的更新、部署很不灵活，任何一个微服务的更新都要重新构建容器镜像，这相当于将Docker容器当作虚拟机使用，没有发挥出Docker的优势。
2. 将每个微服务封装到单独的容器中。Docker具有轻量、环境隔离的优点，很适合用来封装微服务。不过这样可能产生额外的性能问题。一个是大型服务的容器化会产生数倍的计算实例，这对分布式系统的调度和部署带来很大的压力；另一个是性能恶化问题，例如有两个关系紧密的服务，相互通信流量很大，但被部署到不同的机房，会产生相当大的网络开销。

对于支持微服务的问题，Kubernetes的解决方案是Pod。每个Pod由多个容器组成，是服务部署、编排、管理的最小单位，也是调度的最小单位。Pod内的容器相互共享资源，包括网络、Volume、IPC等。因此同一个Pod内的多个容器相互之间可以高效率地通信。

我们借鉴了Pod的思想，在容器平台上开发了面向微服务的容器组，我们内部称之为**set**。一个set逻辑示意如下图所示。

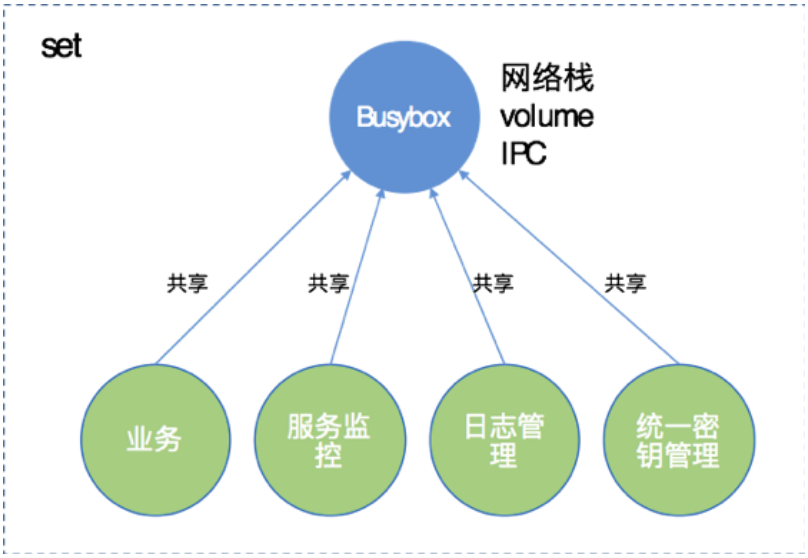

图 5. Set逻辑示意图

set是容器平台的调度、弹性扩容/缩容的基本单位。每个set由一个BusyBox容器和若干个业务容器组成， BusyBox容器不负责具体业务，只负责管理set的网络、Volume和IPC配置。

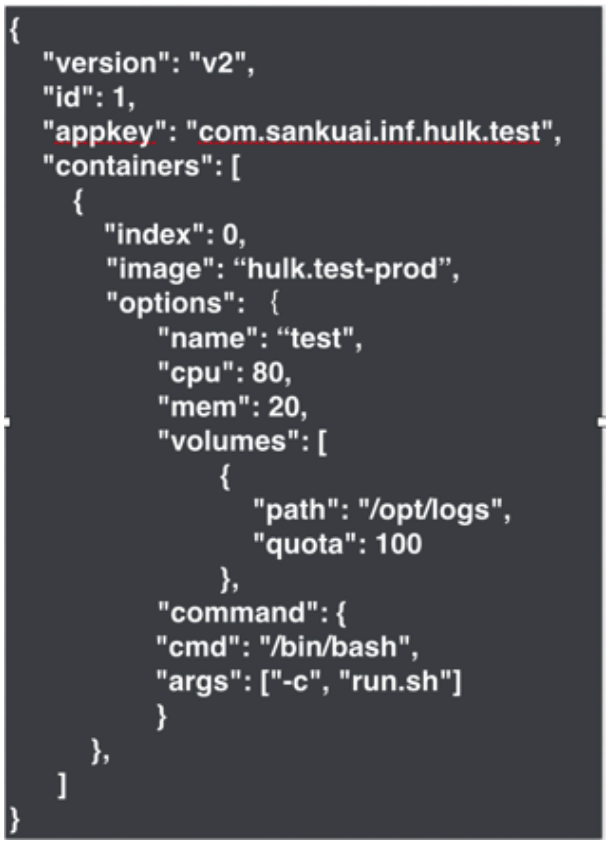

图 6. set的配置json

set内的所有容器共享网络，Volume和IPC。set配置使用一个JSON描述(如图6所示)，每一个set实例包含一个Container List，Container的字段描述了该容器运行时的配置，重要的字段有：

- **Index**，容器编号，代表容器的启动顺序；
- **Image**，Docker镜像在Glance上的name或者ID；
- **Options**，描述了容器启动时的参数配置。其中CPU和MEM都是百分比，表示这个容器相对于整个set在CPU和内存的分配情况（例如，对于一个4核的set而言，容器CPU:80，表示该容器将最多使用3.2个物理核）。

通过set，我们将美团点评的所有容器业务都做了标准化，即所有的线上业务都是用set描述，容器平台内只有set，调度、部署、启停的单位都是set。
对于set的实现上我们还做了一些特殊处理：

- Busybox具有Privileged权限，可以自定义一些sysctl内核参数，提升容器性能。
- 为了稳定性考虑，用户不允许SSH登陆Busybox，只允许登陆其他业务容器。
- 为了简化Volume管理，每一个set只有一个Volume，并挂载到Busybox下，每个容器相互共享这个Volume。

很多时候一个set内的容器来自不同的团队，镜像更新频度不一，我们在set基础上设计了一个灰度更新的功能。该功能允许业务只更新set中的部分容器镜像，通过一个灰度更新的API，即可将线上的set升级。灰度更新最大的好处是可以在线更新部分容器，并保持线上服务不间断。

Docker稳定性和特性的解决方案：MosDocker

众所周知，Docker社区非常火热，版本更新十分频繁，大概2～4个月左右会有一个大版本更新，而且每次版本更新都会伴随大量的代码重构。Docker没有一个长期维护的LTS版本，每次更新不可避免地会引入新的Bug。由于时效原因，一般情况下，某个Bug的修复要等到下一个版本。例如1.11引入的Bug，一般要到1.12版才能解决，而如果使用了1.12版，又会引入新的Bug，还要等1.13版。如此一来，Docker的稳定性很难满足生产场景的要求。因此十分有必要维护一个相对稳定的版本，如果发现Bug，可以在此版本基础上，通过自研修复，或者采用社区的BugFix来修复。

除了稳定性的需求之外，我们还需要开发一些功能来满足美团点评的需求。美团点评业务的一些需求来自于我们自己的生产环境，而不属于业界通用的需求。对于这类需求，开源社区通常不会考虑。业界许多公司都存在类似的情况，作为公司基础服务团队就必须通过技术开发来满足这种需求。

基于以上考虑，我们从Docker 1.11版本开始，自研维护一个分支，我们称之为MosDocker。之所以选择从版本1.11开始，是因为从该版本开始，Docker做了几项重大改进：
Docker Daemon重构为Daemon、Containerd和runC这3个Binary，并解决Daemon的单点失效问题；

- 支持OCI标准，容器由统一的rootfs和spec来定义；
- 引入了Libnetwork框架，允许用户通过开发接口自定义容器网络；
- 重构了Docker镜像存储后端，镜像ID由原来的随即字符串转变为基于镜像内容的Hash，使Docker镜像安全性更高。

到目前为止，MosDocker自研的特性主要有：

1. MosBridge，支持美团云网络架构的网络驱动, 基于此特性实现容器多IP，VPC等网络功能；
2. Cgroup持久化，扩展Docker Update接口，可以使更多的CGroup配置持久化在容器中，保证容器重启后CGroup配置不丢失。
3. 支持子镜像的Docker Save，可以大幅度提高Docker镜像的上传、下载速度。

总之，维护MosDocker使我们可以将Docker稳定性逐渐控制在自己手里，并且可以按照公司业务的需求做定制开发。

在实际业务中的推广应用

在容器平台运行的一年多时间里，已经接入了美团点评多个大型业务部门的业务，业务类型也是多种多样。通过引入Docker技术，为业务部门带来诸多好处，典型的好处包括以下两点。

- 快速部署，快速应对业务突发流量。由于使用Docker，业务的机器申请、部署、业务发布一步完成，业务扩容从原来的小时级缩减为秒级，极大地提高了业务的弹性能力。
- 节省IT硬件和运维成本。Docker在计算上效率更高，加之高弹性使得业务部门不必预留大量的资源，节省大量的硬件投资。以某业务为例，之前为了应对流量波动和突发流量，预留了32台8核8G的虚拟机。使用容器弹性方案，即3台容器+弹性扩容的方案取代固定32台虚拟机，平均单机QPS提升85%， 平均资源占用率降低44-56%(如图7，8所示)。
- Docker在线扩容能力，保障服务不中断。一些有状态的业务，例如数据库和缓存，运行时调整CPU、内存和磁盘是常见的需求。之前部署在虚拟机中，调整配置需要重启虚拟机，业务的可用性不可避免地被中断了，成为业务的痛点。Docker对CPU、内存等资源管理是通过Linux的CGroup实现的，调整配置只需要修改容器的CGroup参数，不必重启容器。

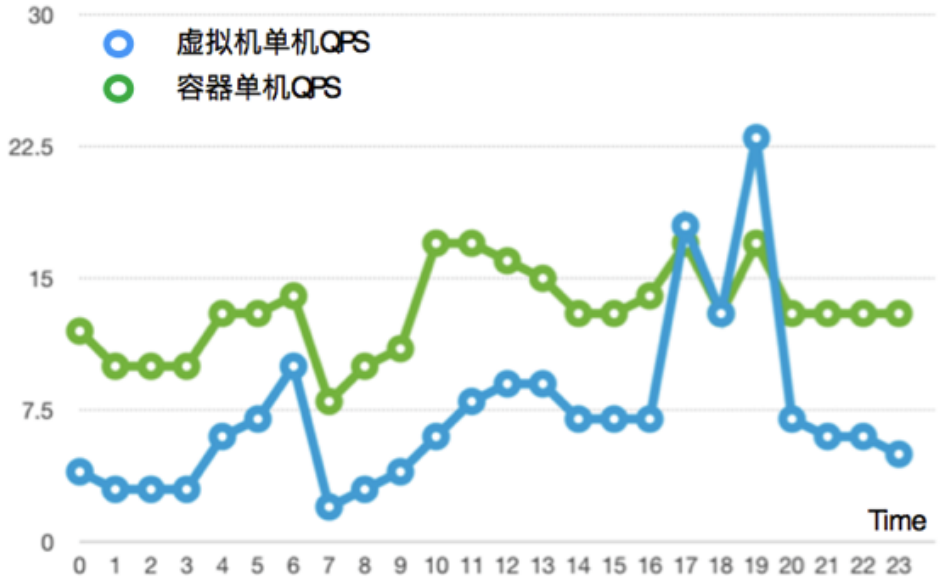

图 7. 某业务虚拟机和容器平均单机QPS