58集团云平台架构实践与演进

转载

Docker_ 于 2019-06-06 07:45:00 发布 392 收藏

在17年底，我们分享了《 高可用Docker容器云在58集团的实践》这篇文章，对整个容器云基础环境搭建与应用选型进行了详细介绍，本文是在该文章基础之上的进阶篇，是针对具体业务场景的落地解决方案。如果对基础环境选型比较感兴趣，可以查看上篇文章，在本文的最后会附上相关文章的链接。对于上篇文章讨论过的内容，本文将不再进行详细讨论。后续每个月，云团队都会选择平台中某一具体领域的相关工作进行详细讨论与分享，欢迎大家关注。大家想了解哪方面的实现方案与细节，可进行相应留言。

背景

通过容器化技术，58云计算平台主要解决以下几个问题：

资源利用率低：通过云化技术可以将资源利用率提升至原有的3-4倍，甚至更高。

服务扩容效率低：将传统扩容的时间从小时级别降低为分钟级别。

上线流程不规范：基于同一的镜像模板，约束整个上线过程。

为了解决上述问题，云团队通过技术选型与反复论证最终决定基于Docker与Kubernetes体系构建整个容器云环境。
云计算平台的发展历程如下：

整体架构

58云计算平台的整体架构如下：

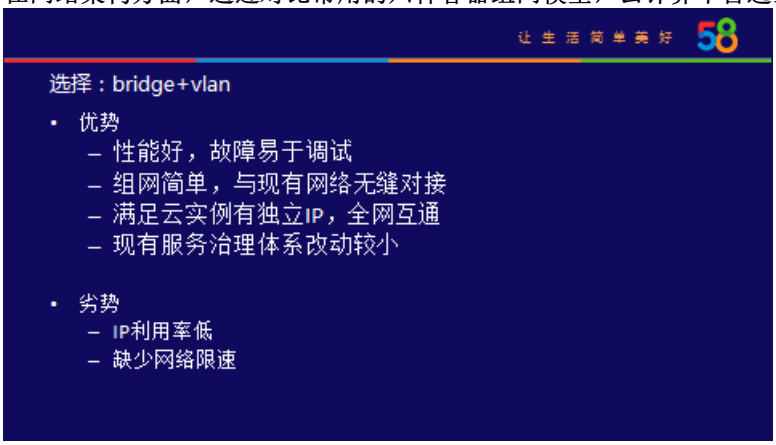
所有容器云的架构都是相似的，这里不做赘述，具体可查看上篇文章。
云计算平台承载了集团90%以上的业务流量，作为核心的服务管理与上线系统，它并不是独立运作的，为了保证整个上线流程的一致性与流畅度，满足业务日常管理与维护的通用需求，云平台与集团内部多个核心系统与组件进行了相应的对接与联动。

在项目管理方面，云平台与代码管理系统、项目管理等系统进行了内部对接，实现了代码从编译到生成镜像，再到环境部署的完全自动化。
在运维方面，云平台与CMDB、服务树、监控系统等多个运维系统进行了打通与整合，保证整个运维体系的完整性与用户体验的一致性。
在基础组件方面，集团内部的服务治理体系与常用的中间件系统都针对云平台进行了相应的改造，以适配云平台的工作模式，同时云平台也集成了现有的数据收集组件，保证数据流与用户习惯在云化前后是一致的。
云平台的使用方面，平台定义了四套环境，四套环境基于唯一的镜像仓库，这使得同一个代码版本可以在不同的环境之间进行流转，在保证代码唯一性的同时也使得相关环境的变更不会传递到下一个环境。

构建一个适配多种业务场景的容器云有很多细节需要考虑，云团队做了很多工作，由于篇幅关系，这里主要和大家分享58云计算平台在“网络架构”和“服务发现”两个核心组件上的架构实践与演进。

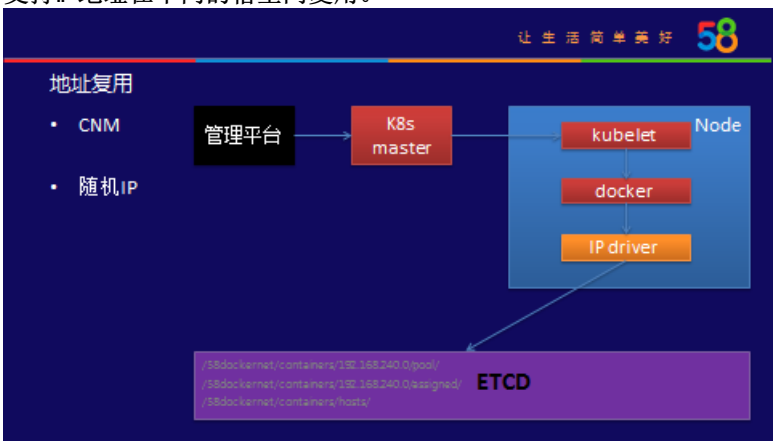
网络架构

在网络架构方面，通过对比常用的六种容器组网模型，云计算平台选择“bridge+vlan”的方式作为基础网络模型。



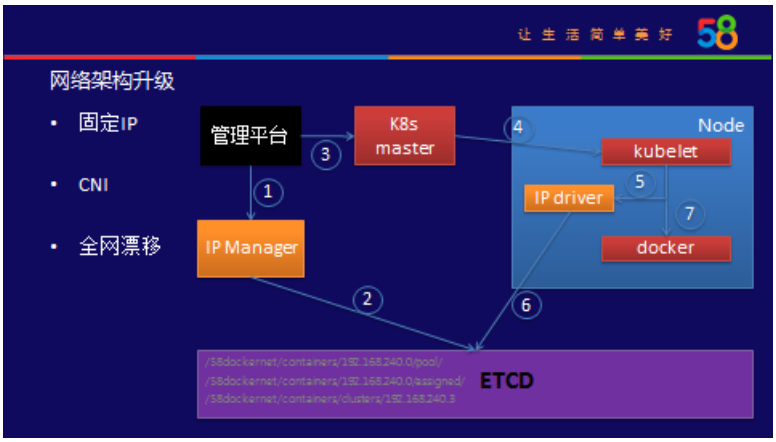
原生bridge网络模型存在两个明显的缺点：IP利用率低、缺少网络限速。

为了解决IP地址利用率低的问题，云平台基于docker的CNM接口开发了网络插件，支持多个宿主间共享同一个容器网段，也支持IP地址在不同的宿主间复用。



这种网络模式下，分配给业务实例的IP地址是随机的，实例每次重启IP地址都可能会发生变更。在推进业务云化的早期，这种模式被业务所接受，但是随着云化进程的不断深入，业务方面提出了更高的要求：固定IP。

业务需求来源于真实的案例：有些服务要求IP不能发生变化，特别是某些依赖于第三方外部接口的服务。同时集团内部很多系统都是以IP固定为前提，如果IP随机分配，这些现有系统将不可用或很难用，极大的影响用户体验。如果IP固定的需求不能被满足，业务云化将很难推进下去。所以在18年4月份，云平台对网络架构进行了升级，支持了固定IP模式。网络架构的升级很好的保证了业务云化的进程。



固定IP的网络架构基于Kubernetes的CNI接口实现，增加了IP控制器模块，业务每次扩缩容时，都会与IP控制器交互，进行IP的变更。在业务正常升级流程中，归属于业务的IP将不会发生变化。依托于腾讯机房的网络支撑，平台将容器网段的路由规则下发到交换机，实现了容器的全网漂移，而不仅仅局限于固定的交换机。（如果你想深入快速学习Kubernetes，**可以报名参加我们组织的为期3天的Kubernetes实战培训**，一线资深讲师带你从0开始上手Kubernetes。）
针对网络限速的需求，云平台结合容器网络虚拟化的特性，基于自研监控与tc工具实现了容器网络限速的能力。

标准的tc工具只支持单向限速，即出口流量限速。单方向限速无法满足业务的实际需求。在深入研究容器网络虚拟化的原理后，我们发现，容器在做虚拟化时，会创建一个网卡对，他们是对等的网络设备，在宿主机上体现为veth，在容器内部体现为eth0。基于这个特性，我们实现了双向限速：即对两块网卡同时做出口流量限速。由于是对等网卡，所以对veth做出口流量限速就相当于对eth0做入口流量限速。
在网络限速的基础上，云平台又进行了多维度的完善，分别支持动态限速、秒级限速与弹性限速等网络应用场景，极大的实现带宽复用。

服务发现

服务发现是容器云平台中非常核心的服务组件，是流量的入口，也是服务对外暴露的接口。云平台中IP动态分配，节点弹性伸缩，需要有一套自动变更负载均衡器的方式。对于后端服务，集团有成熟的服务治理框架与体系，在云化过程中，中间件团队对服务治理体系进行了改造使其可以适配不断变化的云环境。对于前端服务，集团是基于Nginx做负载均衡，并且没有一套支持IP自动变更的架构。为了实现前端服务的云化，云平台团队与运维团队共同设计了全新的服务发现架构。
在调研阶段，云团队也调研了Kubernetes自带的服务发现机制，这一机制无法提供复杂的负载均衡策略，并且无法满足业务在特殊场景下需要对部分节点摘除流量的需求，所以最终我们否定了这一方案。

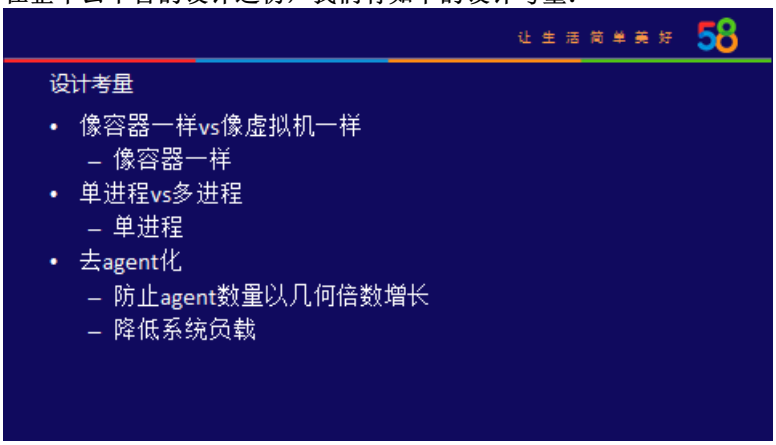
这是业界典型的服务发现架构。服务端和负载均衡器通过Consul进行解耦。服务注册在Consul中，负载均衡器通过Watch

Consul中目录的变化来实时感知节点变更。在云化的早期，为了满足快速上线的需求，云平台对集团内部的Java Web框架进行了修改，使得其以心跳的方式自动注册到Consul中。这很好的解决了负载均衡器自动感知业务变更的问题以及云化过程中的流量灰度问题。
这一架构也引入了新的问题：调试问题与多语言扩展问题。由于是Java框架代理业务注册到Consul中，这使得活跃节点无法被下掉流量从而进行调试，但实际场景中很多故障调试需要下掉流量后才能进行。对Java语言的支持是通过修改框架来完成的，而集团中还有很多其他类型语言，比如PHP、Node.js和Go等。对每种接入语言或框架都需要以修改代码的方式才能接入到云平台中来，开发成本高并且对业务不友好。Consul被直接暴露给服务方，也增加了Consul的安全风险。
基于此，在18年4月，云平台对整个服务发现架构进行了升级。

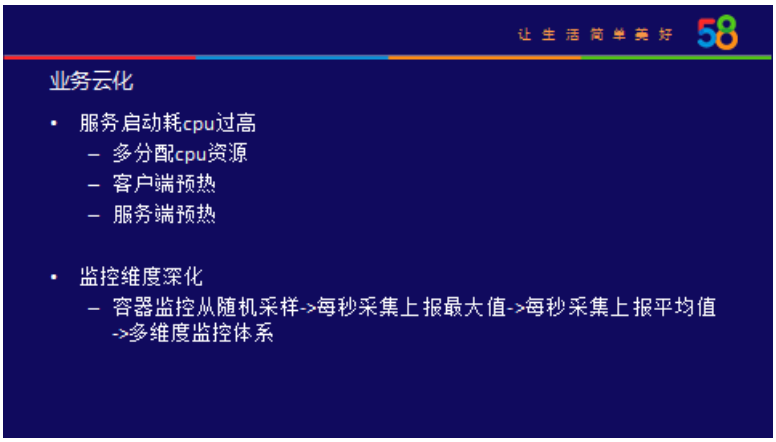
新的服务发现架构中，业务与Consul中间增加了Proxy代理层，代理层通过Watch Kubernetes事件实时感知业务节点信息的变化，配合健康检查功能可以保证业务节点在变更时流量无损。基于代理层，任意语言的程序不需要做任何修改，仅通过简单配置即可接入到云平台。服务的注册与发现托管到云平台，Consul组件对业务透明，开发人员使用更友好。

复盘与反思

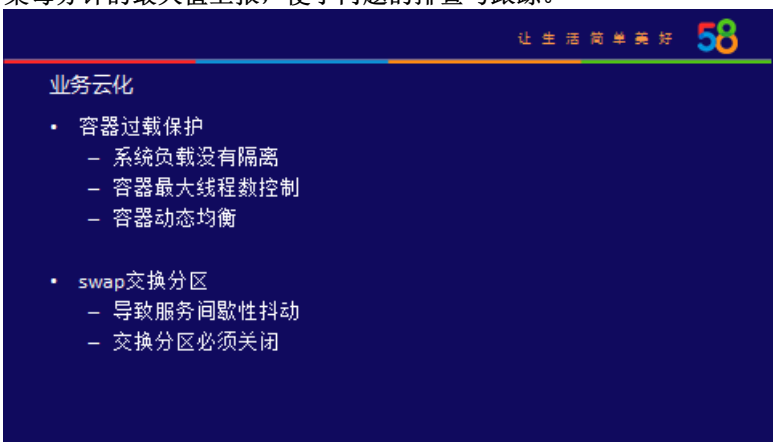
回顾这两年来的容器云架构演进过程与业务云化历程，复盘遇到的棘手问题及其解决方案，与大家共同探讨。
在整个云平台的设计之初，我们有如下的设计考量：



容器云都会面临一个问题：对于业务来说，是容器还是虚拟机？虚拟机是业务习惯的使用模式，业务更容易接受。容器是以服务为核心，服务存在即存在，服务关闭即销毁，不再是虚拟机以机器为核心的模式。思虑再三，最终决定以容器的模式来定位，全新的平台提供全新的服务模式与体验。
虽然是以容器为核心，但容器中可运行单进程也可运行多进程。我们规范容器中只运行一个业务进程，防止由于运行多进程而导致业务之间相互干扰情况的发生，这类问题很难进行排查与定位。
去Agent化：在传统物理机模式下，一台物理机上可能会有多个Agent：运维管控Agent、监控Agent、业务自定义Agent等。云化后面临一个现实的问题：我们需要在每个容器中都安装这些Agent么？如果每个容器中都集成这些Agent，对云平台来说是简单的，这没有任何工作量。但是一台物理机上可能会运行上百个容器，这些Agent数量也会大量膨胀，带来系统负载的增加的问题。基于此，云平台投入大量的精力来实现容器的无Agent化，即在所有功能正常运行的前提下，Agent只运行在宿主上，不会运行在容器中。
业务云化过程中云团队遇到了很多问题案例，并形成自己独特的解决方案。这里选择了几个有代表性的案例，与大家进行分享。



服务启动耗CPU过高是云化早期时遇到的棘手问题。这个问题产生的原因很多：Java的语言特性导致JVM是在运行中逐步进行优化的；内部的很多开发框架都是在流量过来时才初始化链接；某些业务资源也是在流量过来时才进行初始化。当流量分发过来时，多种资源同时初始化导致服务需要大量的CPU资源，超过了平台为其分配的CPU配额，服务出现大量超时与抛弃。这一问题的简单解法是多分配CPU资源，但从长远角度来看这会导致资源利用无法有效把控，也会影响弹性调度的效果。最终，我们从两个维度解决这一问题：在调用方增加预热策略，流量逐步分发过来，CPU资源使用更平滑；在服务方增加预热方法，默认初始化链接等资源，同时引导用户进行自定义的初始化。
在容器监控维度方面，由于默认的监控数据是基于随机采样的分钟级数据，导致当服务出现短期秒级的CPU波动时，监控系统无法捕获，从而影响问题的排查与定位。针对这一问题，云平台对监控维度进行了深化，增加了容器级别秒级监控，实时采集每分钟的最大值上报，便于问题的排查与跟踪。



由于Cgroups只对CPU和内存资源进行了隔离与限速，并没有对系统负载进行隔离，如果容器中运行进程或线程数过多，会导致宿主机整体的负载波动，进而对上面的所有服务都会造成影响。
云平台增加了两个维度的控制策略：容器级别的最大线程数控制，防止由于单个容器线程数过多会对宿主造成影响。宿主级别的过载保护，当宿主上的负载过高时，过载保护策略会自动触发宿主上的容器进行漂移，直至负载降至合理的范围。
云平台必须关闭swap交换分区，这是一个深刻的经验教训。在云化的早期，云平台的交换分区没有关闭，部分服务经常出现由于使用交换分区而导致的耗时随机抖动，这类问题很难排查，耗费了大量的精力。
58云计算平台核心软件版本变迁过程如下：

	初始版本	当前版本
centos	7.2	7.2
kernel	3.10.0-327	4.18.7-1
docker	1.10.3	1.10.3(patched)
Kubernetes	1.5.2	1.7.12(patched)
registry	2.6.0	2.6.0
Docker storage	direct lvm	direct lvm

后记

容器云在58集团的实践与探索过程中，云团队做了很多技术选型与优化工作，也进行了很多技术方案的架构演进工作，究其根本原因是58集团是发展了十多年的互联网公司，它有自己独特的业务场景和成熟的开发体系与基础组件，容器云在集团内部起步较晚，内部系统很难为云平台做适配，只能云平台通过不断演进来适配不停发展的业务场景。这也是在较有规模的互联网公司做基础架构与创业公司的区别和挑战所在。所幸在团队的共同努力下，我们用1年多的时间完成了集团流量服务的云化工作。目前，集团内部还有很多服务需要云化，未来的挑战更大。
作者：姚远，58集团云计算平台技术负责人，架构师；多年互联网开发和架构经验，长期从事并关注服务高并发与高性能等方面工作，在58集团先后负责过消息总线、调用链跟踪系统等中间件系统的设计与架构工作；当前主要负责云计算平台的架构与设计、推进业务云化等工作。
本文转载自公众号：58架构师， 点击查看原文。

Kubernetes入门与进阶实战培训

Kubernetes入门与进阶实战培训将于2019年6月14日在北京开课，3天时间带你系统掌握Kubernetes，学习效果不好可以继续学习。本次培训包括：Docker基础、容器技术、Docker镜像、数据共享与持久化、Docker三驾马车、Docker实践、Kubernetes基础、Pod基础与进阶、常用对象操作、服务发现、Helm、Kubernetes核心组件原理分析、Kubernetes服务质量保证、调度详解与应用场景、网络、基于Kubernetes的CI/CD、基于Kubernetes的配置管理等，点击下方图片或者点击阅读原文了解详情。