腾讯游戏容器云平台的技术演进之路

InfoQ 2017-08-21 09:40

作者丨郭蕾

嘉宾丨尹烨

从 2014 年开始，腾讯游戏就逐步在生产环境中使用容器相关技术，到现在，代号为 TenC 的容器平台支撑了近 200 款游戏的运营。在这 3 年的时间里，整个平台经历了从最开始的“轻量级虚拟机”使用 Docker 的方式，到现在的原生容器云方式的迭代，接入的业务也由原来的在线服务扩展到现在的微服务、大数据、机器学习等类型。

2015 年时，InfoQ 记者曾对腾讯游戏进行过一次专访，在提到容器对于游戏业务的价值时，腾讯游戏高级工程师尹烨这样说道：“相比其他行业，游戏业务更为复杂多样，有端游、手游、页游之分，同时，还要分区和分服，甚至还要区分自研和代理。这些特性给运维和部署带来了诸多不便，而 Docker 统一的镜像分发方式，可以标准化程序的分发部署，提高交付效率。另外，游戏业务的生命周期长短不一，这也就需要弹性的资源管理和交付，而容器更加轻量，资源的交付和销毁更快。”

而在当时，Kubernetes 刚刚发布 1.0 版本，尹烨也表示他们对于 Kubernetes 的应用还没有完全发挥其优势，接下来一段时间，他们也会探索容器平台与业务开发、运维的结合方式。那在时过境迁的两年之后，腾讯游戏在容器平台的实践方面又有哪些经验和心得？InfoQ 记者再一次采访了尹烨。另外，尹烨也将会在 CNUTCon 全球运维技术大会 上与参会者分享他们的经验教训。

InfoQ：2014 年的时候腾讯游戏就开始在生产环境中使用了 Docker，到现在已有 3 年时间。能否谈谈你们从最初的简单使用 Docker 到现在的容器云平台，这中间你们大致经历了哪几个阶段？

尹烨：现在我们的 Docker 容器平台（内部代号：TenC）主要支撑了如下三种类型的业务场景，也基本上对应了平台发展的三个阶段，每个阶段都是紧随着业务的需求在推进。

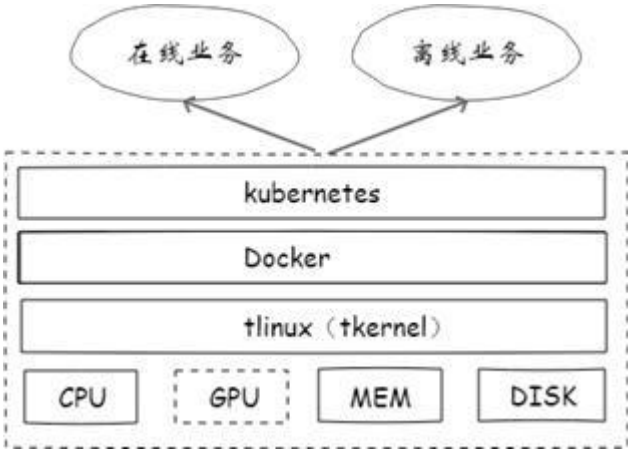
第一阶段：我们把容器当作轻量虚拟机来给业务使用，让业务程序不需要特殊修改，就可以在容器中正常运行起来，我们花了很多精力来兼容原有业务架构和使用习惯，以及保证底层运行环境的稳定运行；

第二阶段：后来在代理游戏中，出现一些微服务架构的业务，这些业务完全采用原生的 Docker 方式，基于 Docker 镜像来做分发和部署，弹性伸缩；

第三阶段： 再后来随着大数据和 AI 兴起，这类业务计算量非常大，资源需求要求高弹性，使用 Docker 容器来交付资源相对于原来物理机、虚拟机方式效率高很多。

InfoQ：目前腾讯游戏有多少的业务跑在容器云上？可否谈下你们的容器云技术栈？

尹烨：目前有过百款游戏业务都运行在容器上。在游戏业务接入容器平台前，会有一些评估要素。评估通过的话，新业务基本上会首选放到容器里。我们的平台主要基于 Docker、Kubernetes（后面简称 K8s）等开源组件开发的。当前 Docker 主要使用了 1.12 的版本，Kubernetes 主要使用 1.2/ 1.5 版本。在实际的应用过程中，我们会基于我们自身的基础环境做一些二次定制开发，包括资源调度和网络部分，例如我们自己开发的 SR-IOV CNI 插件等。 整体结构大致如下：



其中 TLinux 是腾讯操作系统团队自研的服务器操作系统，操作系统团队针对 Docker 平台需求，进行了一系列内核特性研发和针对性改进。TLinux 为游戏在 Docker 容器上的稳定运行奠定了基础。例如，

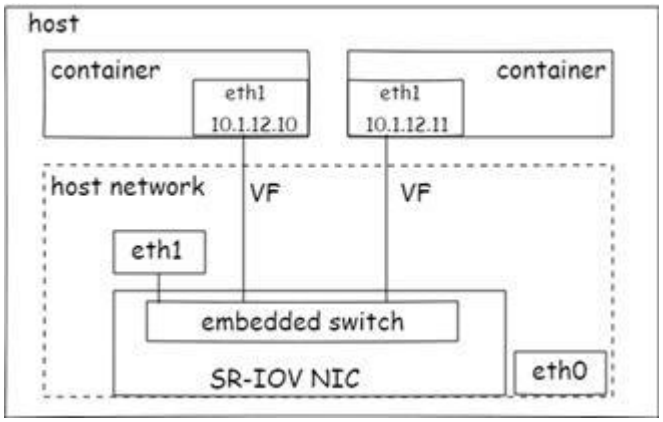
- cgroup 高版本特性移植，如 cgroup namespace, pids cgroup, cgroup 支持 IO 隔离等；
- aufs、overlayfs 特性移植；网络 sysctl 隔离；
- 其他诸如 cgroup、overlayfs、xfs 等等的 bug 修复。

InfoQ：记得 2015 年的时候你们是基于 Kubernetes 0.4 版本做的改造，现在 Kubernetes 也跟着升级了？升级之后之前哪些改动是怎么处理的？现在还有基于主干版本做定制吗？

尹烨：2014 年我们开始使用 Docker 时，Google 刚刚开源了 K8s，K8s 的设计非常精巧，我们就开始做了一些定制，来管理容器。我们现在还在使用那个古老的深度定制版本来管理我们的轻量级虚拟机容器，因为这类业务对编排没什么需求，为了保证平台的稳定，一直没有升级，将来也不太会。但针对微服务和

离线计算这部分业务，我们没有太多侵入式的定制修改 K8s，主要通过插件方式来扩展 K8s，我们会根据自身的需求来跟随社区升级 K8s。

在功能定制方面， 先大概介绍几点吧。比如，我们结合业务需求，和自身的基础环境，主要在调度和网络等方面进行了一些扩展。首先，在网络方面，我们开发了 SRIOV CNI 插件。



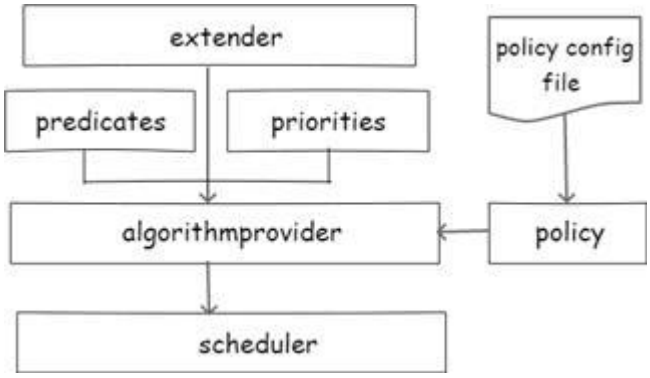
（1）母机上开启 SRIOV 功能，同时向网管系统申请子机 IP 资源，每个 VF 对应一个子机 IP。

（2）Kubernetes 在调度时，为每个 Pod 分配一个 VF 与子机 IP。

（3）在 Pod 拿到 VF 与 IP 资源，进行绑定设置后，就可以像物理网卡一样使用。

同时我们也做了一些优化：比如将 VF 中断绑定到容器分配的 CPU，并开启 RPS，把网卡的软中断分到各个 CPU 处理，来提升网络性能。

其次，在调度方面，为了支持 SRIOV 插件，在容器调度中，除了 K8S 原生提供的 CPU、Memory、GPU 之外，我们还把网络（物理 IP）也作为一种资源来调度，同时结合 K8S 提供的 extender scheduler 接口，我们定制了符合我们需求的调度程序（cr-arbitrator）。其结构如下：



cr-arbitrator 做为 extender scheduler，集成到 K8S 中，包括两部分内容：

（1）预选算法

在完成 Kuernetes 的 predicates 调度后，会进入到 cr-arbitrator 的预选调度算法，我们以网络资源为例，会根据创建的容器是否需要物理 IP，从而计算符合条件的 node（母机）。

（2）优选算法

在整个集群中，需要物理 IP 的容器与 Overlay 网络的容器并未严格的划分，而是采用混合部署方式，所以在调度 Overlay 网络的容器时，需要优化分配到没有开启 SRIOV 的 node 上，只有在资源紧张的情况下，才会分配到开启 SRIOV 的 node 上。

除了 cr-arbitrator 实现的调度策略外，我们还实现了 CPU 核绑定。容器在其生命周期内使用固定的 CPU 核，一方面是避免不同业务 CPU 抢占问题；另一方面在稳定性、性能上（结合 NUMA）得到保障及提升，同时在游戏业务资源核算方面会更加的清晰。

InfoQ：内部有游戏开始使用微服务架构了？你们的容器云平台是如何支撑微服务架构的？

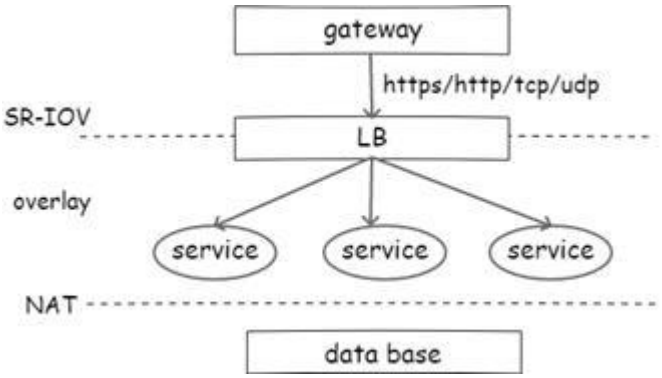
尹烨：腾讯游戏主要分两部分，一部分是代理的，另外一部分是自研的。对于自研的业务，经历那么多年的沉淀，有一套非常成熟的框架和运营体系。微服务架构带来的变化比较大，已有的业务架构转过来成本太高。

而代理游戏进行架构的转变，相比而言成本会低很多。所以，一些新的代理业务往往会采用微服务架构。今年开始，我们内部也有一些在线业务也在开始这方面的尝试，我想后面这种业务会越来越多的。

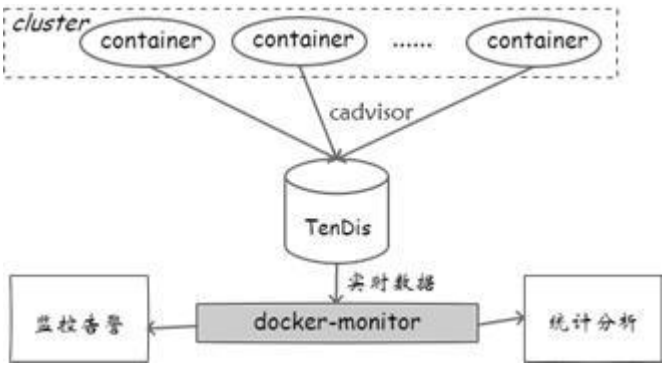
这些业务本身的架构使得业务很容易在容器上运行。另外，如果业务需要使用 K8s 的一些高级特性，比如服务发现、自动扩缩容机制等，则需要业务在架构上和实际部署时做一些简单的适配，这些适配都比较简单。

相对于轻量虚拟机，微服务的业务在网络、监控、日志等方面有较大的不同。

首先是网络部分，受限于底层物理网络，业务的容器不再每个容器一个物理 IP（这会消耗大量的内网 IP 资源，而且管理也不方便），所以，我们使用了 Overlay 的网络方案，将容器的网络与底层物理网络解耦。业务的逻辑部分的容器都会跑在 Overlay 网络上，但是业务的数据层服务，比如 MySQL，Redis 之类的，仍然跑在物理网络上，Overlay 与数据层之间通过 NAT 访问。同时，接入层通过内部 LB 对接外部网关，分离网关与业务服务。大致结构如下：



其次是监控与告警，对于微服务业务，无法使用原来的监控方案。而监控、告警是整个游戏运营过程中最为核心的功能之一，在游戏运行过程中，对其性能进行收集、统计与分析，来发现游戏模块是否存在问题，负载是否过高，是否需要扩缩容之类等等。在监控这一块，我们在 cAdvisor 基础上进行定制，其结构如下：



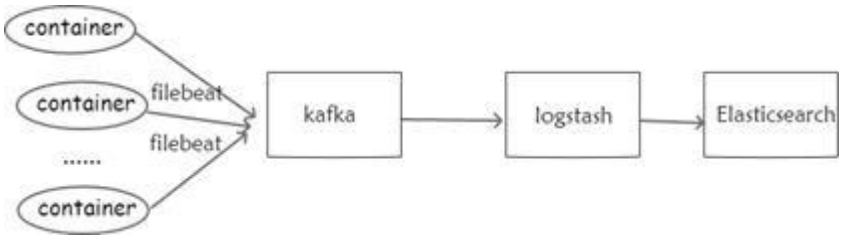
（1）每个母机部署 cAdvisor 程序，用于收集母机上容器的性能数据，比如 CPU 使用情况、memory、网络流量、TCP 连接数等。

（2）在存储方面，目前直接写入到 TenDis 中，后续如果压力太大，还可以考虑在 TenDis 前加一层消息队列，例如 Kafka 集群。

（3）Docker-monitor，是基于 cAdvisor 收集的数据而实现的一套性能统计与告警程序。在性能统计方面，除了对每个容器的性能计算外，还可以对游戏的每个服务进行综合统计分析，一方面用于前端用户展示，另一方面可以以此来对服务进行智能扩缩容。告警方面，用户可以按业务需求，配置个性化的告警规则，docker-monitor 会针对不同的告警规则进行告警。

最后，是业务日志的处理。Docker 在容器日志处理这一块，目前已很丰富，除了默认的 json-file 之外，还提供了 gcplogs、awslogs、fluentd 等 log driver。而在我们的日志系统中，还是简单的使用 json-file，一方面容器日志并非整个方案中的关键节点，不想因为日志上的问题而影响 Docker 的正常服务。

另一方面，把容器日志落地到母机上，接下来只需要把日志及时采集走即可，而采集这块方案可以根据情况灵活选择，可扩展性强。我们当前选择的方案是 Filebeat + Kafka + Logstash + ElasticSearch，其结构如下：



我们以 DaemonSet 方式部署 Filebeat 到集群中，收集容器的日志，并上报到 Kafka，最后存储到 Elasticsearch 集群，整个过程还是比较简单。而这里有个关键点，在业务混合部署的集群中，通过 Filebeat 收集日志时怎样去区分不同的业务？而这恰恰是做日志权限管理的前提条件，我们只希望用户只能查看自己业务的日志。以下是具体的处理方案与流程：

（1）首先我们在 Docker 日志中，除了记录业务程序的日志外，还会记录容器的 name 与 namespace 信息。

（2）接着我们在 Filebeat 的 Kafka 输出配置中，把 namespace 作为 topic 进行上报，最终对应到 Elasticsearch 的 index。

（3）在我们的平台中，一个 namespace 只属于一个业务，通过 namespace，可以快速的搜索到业务对应的日志，通过容器的 name，可以查看业务内每个模块的日志。

InfoQ：这么长时间的应用，有做过复盘吗？未来有什么计划？

尹烨：我们一直致力于为游戏业务提供高效的计算资源。从轻量虚拟机，到微服务，再到离线计算，我们一直紧随业务的场景往前演进。特别是弹性计算场景，

相对于原来交付物理机 / 虚拟机的方式，现在基于容器的方式更加高效。我们希望在满足业务对计算资源需求的同时，尽可能提高资源的利用率，从而降低运营成本。我们会一直朝这个目标努力。

具体来说，整个平台也还有很多需要改进和优化的地方。比如，网络方面，我们希望简化外部网关的接入，进一步提升底层 *overlay* 网络的性能。优化资源调度策略，考虑更多的一些因素，比如任务优先级、任务运行时长等，将在线业务与离线业务混合部署等。