Please visit website: http://cxyroad.com

K8s 的最后一片拼图：dbPaaS

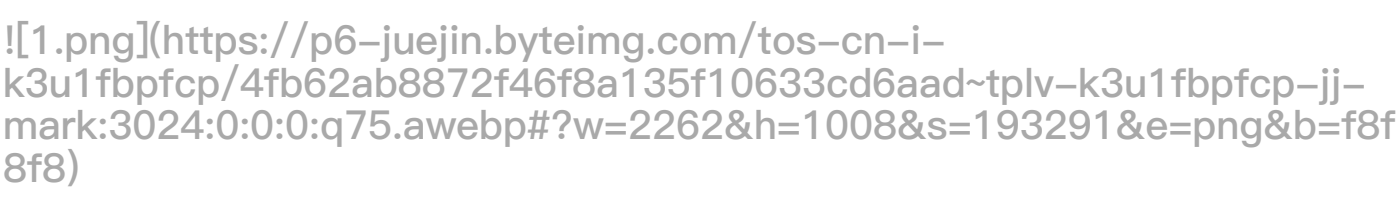
K8s 的发展使得私有云跟公共云之间的技术差不断的缩小，不管是在私有云还是公共云，大家今天都在基于 K8s 去开发 PaaS 系统。而 K8s 作为构建 PaaS 的基础，其全景图里还缺最后一块“拼图”——dbPaaS。作为一个云数据库行业干了十几年的资深从业者，以及一个云数据库初创公司的创始人，在本文中，我将结合近年来数据库和云计算的发展方向，以及我们在技术和工程上的实践，分享一些看法。

私有化部署，数据库发展的方向

近年来, 数据库整个领域主要在以下三个方向上发展: 公共云全托管，Serverless 和私有化部署。

公共云全托管，包括各家云厂商的 RDS，以及每家云厂商自研的数据库，如 PolarDB、Aurora 等等，都属于这个板块。大部分公共云用户都是用这个服务，这个板块比较成熟稳定，跟着公共云的规模一起缓慢增长。公共云的主要的优势是弹性，有更多弹性需求的用户都在被吸引到公有云的 Serverless 产品上。而海外很多初创的公司，像 CockroachDB、Neon、 PlanetScale，包括中国的初创公司 TiDB 都在往 Serverless，无服务器数据库这个方向发展。

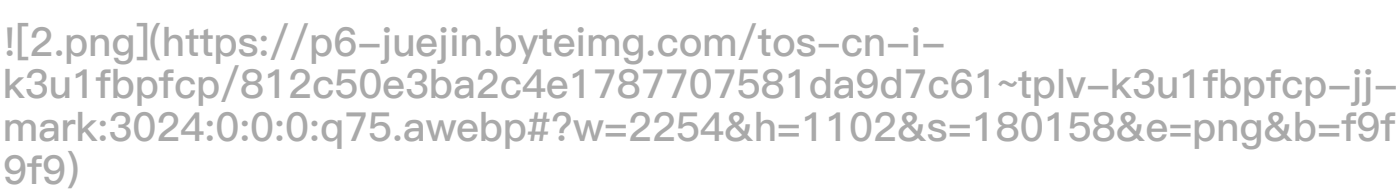
另外一个方向就是私有化部署。有报告说中国公共云的服务器大概只占服务器总数的 5% ，线下部署的比例是非常高的。大型的互联网公司、央国企、银行都是用私有云。国内的信创数据库基本上都是在做这个方向。**本质上来讲，企业选择什么数据库是由企业的业务场景来决定的，很少会因为选择一个数据库而倒推上面的云和架构。**



公有云还是私有云？K8s 正在统一云的操作系统

**企业对云的选择和使用是一直在变化的。选择私有云有很多原因，内因比如

说合规、成本控制、自主可控，外因主要是 K8s。**



第一个变化是中国的私有云正在从 OpenStack 全面进化到 K8s。过去的私有云，是用 OpenStack 管理物理机生产虚拟机。如果客户要用容器和 K8s 再在虚拟机上去搭上面的容器。但是最近，大家都开始用 K8s 去管理物理机，业务最好是能够全部容器化、云原生化。如果你要用虚拟机，可以在 K8s 上管理，例如用 kubevirt 这样的技术在 Pod 里面跑虚拟机，底层是 K8s，VM 在上层，这是一个挺大的变化。

第二个变化是 K8s 的发展使得私有云跟公共云之间的技术差逐步变窄。公共云和私有云上的 K8s 技术差别并不大。过去云厂商主要的技术壁垒都在于我怎么把几百万台虚拟机运维好，用规模优势来降低成本，但是这样的技术在 K8s 面前就像马奇诺防线一样被绕过了。公共云和私有云今天都得站在怎么把容器做好，怎么把 K8s 生态做好这同一条起跑线上。

第三个趋势是，不管运行在私有云还是公共云，大家都在基于 K8s 去开发 PaaS 系统。K8s 作为一个多云的，可以一统公共云、私有云的容器操作系统，可以屏蔽底层 IaaS 的差异，让上层通过 Pod、Service、PVC 等抽象来操作 IaaS 资源。调度器、服务发现、配置管理、API Server 等等这些开发一个 PaaS 所需要依赖的基础能力，K8s 都提供了，不仅可以节约很多开发成本，而且 K8s 在设计和工程上都做的更好。比如说，K8s 的 API 是声明式 API，这个就是个挺先进的设计。公共云厂商做的比较早，在设计 API 的时候还没有这个理念，所以 API 都是过程式的。但后来的系统，比方说 Terraform，就用的声明式 API，所以开发者使用起来更简单，书写起来更容易理解，也更不容易出错。在 K8s 上做 PaaS 就可以利用好这些后发优势。

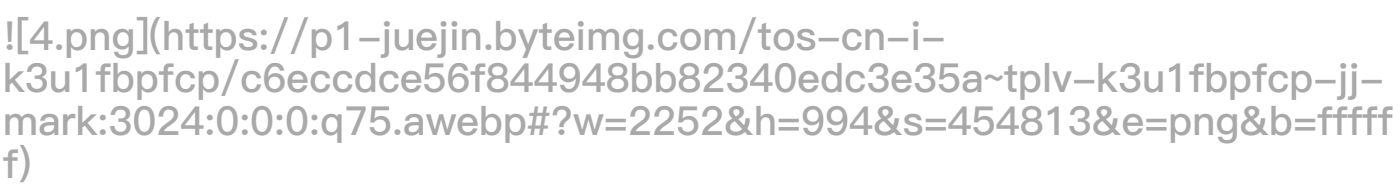
K8s 的最后一片拼图 —— dbPass

K8s 作为构建 PaaS 的基础，其全景图里还缺最后一块“拼图”——dbPaaS。在 K8s 上构建 dbPaaS 是大势所趋。所有的 PaaS 都在用 K8s 做自己的底座，dbPaaS，就是数据库的 PaaS 也不例外。如果企业的私有云都用 K8s 来构建，再搞一套虚拟机或者物理机的管理平台，然后在这个基础上发明一套 K8s 已经有了的能力，再做数据库的管理，其实是一个重复建设，不是特别合理。

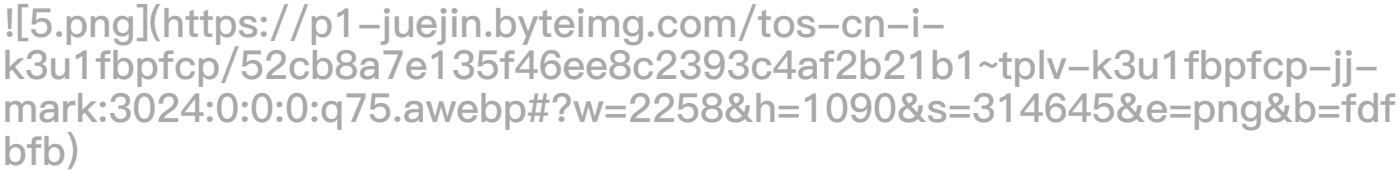
k3u1fbpfcp/f8ba3f594d7d4e62a02126a776783b27~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=2242&h=1112&s=303454&e=png&b=ffffff)

dbPaaS 有哪些挑战？我过去在阿里云做 RDS，最深有体会的一件事情是数据库的类目太多了，版本也太多了。大家使用的组合里面至少几十种数据库的不同版本。

每一种数据库的运维操作都也很复杂，因为数据库是存储企业关键数据的基础设施，运维操作精细而复杂。与之对应的就是人不够，与之对应的痛苦就是人手不够，挺多事情都是可以靠堆人去做的，但是作为云厂商，要考虑向做一件事情投入人力的 ROI（投资回报率），也就是人效。



说到人效，这里就要说到烟囱式架构这个陷阱。企业在构建 dbPaaS 的时候，方法通常是对于每一种要支持的数据库，就搞一个独立的小团队，然后写一套独立的代码，甚至有的时候连运维人员都是独立的。这种做法人效低，因为支撑每根烟囱的人力都是一个小池子，人员很难在烟囱之间流动，因此用烟囱式架构去支持 dbPaaS 这种长尾市场的业务，是一个错误的选择。拉长时间看，人员变动引起故障的概率在烟囱式架构里更高。还有更多的问题，比如资源不能做到跨引擎共享和混部，会增加部署一套 dbPaaS 的起步成本。



像搭积木一样搭建数据库

用一种新的方法去实现 dbPaaS。今天各种各样的数据库都会发布自己的容器，容器本身就是一个标准的东西，那我们能不能像搭积木一样的，在 K8s 上把它们给组装起来部署提供服务，然后把它们的运维操作，也以一种标准地去组装？

乐高积木能够拼搭成各种各样的蓝图，其本质在于乐高是高度标准化的。它的

凸起凹槽、厚度，都是有标准的。如果想让数据库像乐高积木一样的能够搭起来，也要解决一个标准问题，然后把各种各样的数据库容器适配到这个标准上。

在计算机科学当中，有几个比较著名的标准，比如说 POSIX，POSIX 是对文件系统操作接口的抽象。K8s 里面有容器运行时标准 CRI，容器存储标准 CSI，容器网络标准 CNI，等等。通过这些标准，各种各样的容器网络、存储、分布式存储的项目都可以对接到 K8s 生态中去。除了标准和抽象之外，还有一个我们值得借鉴的方法叫分层，比如说 OSI 的 7 层网络协议和 TCP/IP 4 层协议。通过分层，各家厂商的网络产品软硬件、各种协议，都能找到合适的一个层次，通过层与层之间的标准接口，适配起来组成一个完整的系统。

我们通过抽象和分层这两种方法来定义容器化数据库的标准，设计了 API，本质上它类似于 POSIX API，是一个标准。我们先看下 POSIX API，有了 POSIX API 之后，上层的应用就可以用一个很标准的 API 去操作各种文件，不用管底层的文件系统是什么，是 ext4，xfs 还是 nfs。类似的，我们设计的 API 是一个描述多个数据库容器之间的关系和拓扑结构，以及每个数据库容器的组成、可以提供的服务以及如何响应各种事件的行为的一个描绘，它是抽象的，跟任何一种具体的数据库是无关的，能够表达各种各样的数据库。此外，在设计这个 API 的时候我们也做了分层。我们分了五层，最底下一层是 K8s 的 API，它代表的 IaaS 的对象，倒数第二层是 Instance，描绘怎么把 IaaS 资源组装起来，构成一个单节点的数据库的副本。InstanceSet 就是把多个 Instance 组装成了一个多副本的数据库。Component 在多副本的基础上，加了更多数据库的行为，比如说成员管理，备份恢复等等。再之上是组装成 Cluster， Cluster 就是一个完整的数据库集群，包含多个组件。我们把不同数据库的能力、特性都映射到这个 5 层当中去，通过抽象和分层提出了一个数据库容器组运行在 K8s 上的标准。

dbPaaS Operator 的核心实现就是通过这个抽象的 API 来管理数据库的生命周期，它不知道操作的数据库是 MySQL 还是 PostgreSQL 还是 Redis，它只知道操作的是一个 Cluster 对象，一个 Component 对象，操作的是抽象的对象。通过这种方法就能做到 DBPaaS 最核心的控制软件，跟被操作的数据库引擎无关。**它能做到一套代码适配到多种的数据库引擎上去，这是我们最核心的

创新，是吸收了阿里云 RDS 设计、开发、维护经验教训后的创新，全世界范围目前没有第二个 dbPaaS 采用了这种设计**。用户去操作数据库也会通过 Cluster，Component 这种标准 API，不管对数据库做任何运维操作，体验会非常的接近，学习成本会比较低。

再比如 Redis 高可用部署。Redis 主从架构会包含 Redis Server 和 Sentinel，其中 Redis Server 是一主一从两个副本，而 Sentinel 是三节点。Redis 官方原生的集群架构 Redis Cluster 则是一个水平分片的集群架构，比如说有 5 个分片，可以用 5 个 Component 来表达，每个 Component 又是一主一从两个节点。

接下来一个问题是我们怎么把 30 多种数据库给集成到一套 dbPaaS Operator 代码里。我们再回顾下 POSIX API，如果一个文件系统想兼容 POSIX 标准，让使用 POSIX API 访问文件系统的应用程序都能操作它，就得先实现 POSIX 接口。类似的，一个数据库要想接入这个 dbPaaS Operator，就得让这个引擎也实现一组 dbPaaS Addon API。

给大家举个例子，Redis 是怎么接入到 dbPaaS 当中？Redis 有很多种部署形态，单节点、主从、Sharding、还有 Redis cluster，形态有很多种。按照烟囱式的做法，每种部署形态都会是一个烟囱，一套独立的代码去管理。而在我们的 dbPaaS 里，不同的部署形态可以像积木一样去组装，不同的部署形态就是几行 YAML 的区别。

首先我们要实现一个个“积木块”。“积木块”要把 Redis，Sentinel、Redis proxy 这些数据库镜像进行一次包装。对这些 Docker 镜像，我们用 dbPaaS 的 API，用 YAML 给它们增加一些扩展信息，里面会包含配置文件模板、服务和网络的配置、以及存储的配置等，还会扩展一些被事件触发时会执行的脚本（Action），以及版本镜像列表以及不同版本兼容性的描述信息等等。写完 Redis Server 的定义，我们就可以它看作是一个“积木块”。Redis Sentinel 和 Redis proxy 也是类似的，我们用 YAML 把它们都定义成 dbPaaS 的“积木块”。

定义好这些“积木块”之后，我们在另一个 YAML 当中定义它的拓扑结构，告诉 KubeBlocks 系统刚才那些“积木块”该怎么组装，就能组装成一个集群结构。开发一个 Addon 的成本就是 写 YAML 加上一些脚本，配置文件的模板，以及监控的模板等等，不需要写 Go 代码、Java 代码，大概就是写几千行的非代码文件就能接进来，比从头写一个管控烟囱的开发成本要低多了。

容器化到底会不会影响数据库的性能？K8s 适不适合有状态的服务？

关于数据库的容器化，我常常遇到两个问题。第一个就是容器化会不会影响数据库的性能？第二个是 K8s 适不适合管理有状态服务？

首先回答第一个问题。**容器化不会影响数据库的性能**。容器本身其实就是 Linux 操作系统当中的一个普通的进程，只是这个进程设置了 namespace，设置了 group，使得它能够完成一些叫做“隔离”的障眼法。重要的事情说三遍，容器不是虚拟化，容器不是虚拟化，容器不是虚拟化。

上图大概画了 4 种常见的隔离方案，有虚拟化的，有 gVisor、有容器，在这四种虚拟化的方案当中，前面三种 VM、 microVM、gVisor 都有一层虚拟化层，这层虚拟化层有的放在 hypervisor 里面，有的放在用户态，唯独容器 runC 是没有虚拟化这层的，它就是一个普通的进程。所以性能上来说，在容器基础上做优化，它的底子其实是最好的。为什么要隔离呢？因为现在 CPU 的核数太多了。前两天有一个新闻说明年 AMD、英特尔还有 ARM 的核数都要达到 200 核了，大家都在往一个服务器上塞更多的核，这样密度更高能效更好，但这么多核数单体应用和单体数据库都无法消费，大家用的最多的数据库一般都是 8C、16C，这时候一定是要用隔离技术，而在隔离技术里容器的性能是最好的。

基于容器，我们做了一些优化，把容器做到跟物理机上面的性能一样。最显著的，我们优化容器存储、容器网络、以及数据库的一些参数，可以达到在物理机上同样的甚至更好的性能。比如说 PostgreSQL 的吞吐，我们对 PG 容器做了一些优化之后，吞吐的峰值和公共云上 RDS 一个水平，而且在低并发量下比 RDS 的吞吐更高，在高并发量下吞吐会更稳。用容器跑 Redis 的用户经常担心，延迟会不会增长，如果不优化肯定有影响，所以我们把容器网络换成 eBPF 之后，就跟物理网络的延迟一样低了。

第二个问题是 K8s 适不适合管理有状态的服务？首先 K8s 原生用来管理有状态服务的控制器，叫 StatefulSet。StatefulSet 确实不太好用，它有挺多限制，比如不支持 PVC 存储在线扩容的，另外变更 Pod 的时候，必须要严格按顺序去变更，这使得我没办法去指定一个 Pod 下线，它也不支持异构的各种各样的 Pod 配置。所以**我们在 KubeBlocks 当中把 StatefulSet 换成了我们自己

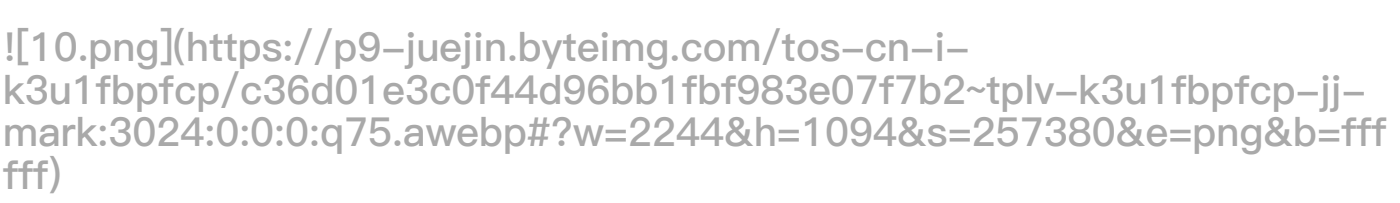
写的 InstanceSet，解决掉了 StatefulSet 的各种问题**。

另一方面，很多人误以为 在 K8s 里面必须要依赖 Pod 的迁移以及 PVC 跨机重新挂载这些 K8s 原生的机制去解决高可用、高可靠的问题，而在线下环境往往没有分布式存储，如果 Node 挂了，PVC 没办法迁移，会导致在线下部署的时候数据库的可用性、可靠性受损。所以我们在 **KubeBlocks 里面的做法是不依赖于 K8s 的检测和重调度，而是使用数据库本身的高可用和多副本技术去解决一个节点挂掉之后服务怎么恢复的问题，把数据库的稳定性和 K8s 的稳定性解耦开**。

K8s 上去构建 dbPaaS 有丰富的应用场景

我们接触到了很多的企业，发现在 K8s 上去构建 dbPaaS 是业界正在进行的趋势。公共云厂商，如阿里云的 RDS 全线产品都是跑在 K8s 上的，腾讯云的 TDSQL 是跑在 K8s 上的，移动云电子云的 RDS 跑在 K8s 上…… 海外的数据库的初创公司，像 TiDB，Cockroach、Neon、PlanteScale 也都是把自己的数据库的 dBPaaS 架在 K8s 上。

国内的互联网公司，如阿里、字节、快手也是如此。银行领域，像工行、招行，也走得很靠前。行业上最近也在牵头做数据库容器化的标准。



总之，在 K8s 上建数据库已经成为趋势。越来越多的企业选择将自己的数据库部署在 K8s 之上, 这种方式可以充分发挥容器技术的优势, 提高数据库的敏捷性、可靠性和可运维性。这种趋势在未来几年内有望进一步扩大和深化。

作者介绍

曹伟（鸣嵩），云猿生数据创始人 & CEO，前阿里云数据库总经理 / 研究员，多年深耕数据库领域云原生数据库 PolarDB 创始人。中国计算机学会数据库专委会执行专委，中国计算机学会开源专委会执行专委，获得 2020 年中国电子学会科技进步一等奖，在国际顶级学术会议发表论文 20 余篇。曹伟于 2022 年开始创业，创建了云原生数据库开源项目 KubeBlocks，并推出相应的企业服务，目前产品已经服务于互联网、金融、运营商、等多个领域的头部企

业。

End

[KubeBlocks 已发布 v0.8.0](http://cxyroad.com/”https://mp.weixin.qq.com/s/7JSBXRg88YWO1N0N8zgQ5A”)！KubeBlocks v0.8.0 推出了 Component API，让数据库引擎的组装变得更加简单。Addon 机制也有了重大改进，数据库引擎的 helm chart 从 KubeBlocks repo 中拆分出去，从此数据库引擎或者版本的变动已与 KubeBlocks 发版解绑。v0.8.0 还支持多版本的数据库引擎定义。Pika、ClickHouse、OceanBase、MySQL、PostgreSQL、Redis 等均有功能更新，快来试试看！

小猿姐诚邀各位体验 KubeBlocks，也欢迎您成为产品的使用者和项目的贡献者。跟我们一起构建云原生数据基础设施吧！

官网: [www.kubeblocks.io](http://cxyroad.com/”http://www.kubeblocks.io”)

GitHub: [github.com/apecloud/ku…](http://cxyroad.com/”https://github.com/apecloud/kubeblocks”)

Get started: [cn.kubeblocks.io/docs/previe…](http://cxyroad.com/”https://cn.kubeblocks.io/docs/preview/user-docs/try-out-on-playground/try-kubeblocks-on-local-host”)

小猿姐，一起学习更多云原生技术干货。

原文链接: https://juejin.cn/post/7386835462134775827