容器平台建设

-- 需求和设计篇

目录

1 银行建设容器平台的背景

随着互联网金融的兴起，互联网企业依托互联网，特别是移动互联网为公众提供越来越多方便快捷、稳定高效的金融类服务，对传统的银行业务带来了很大冲击。作为应对，传统银行也在业务上不断创新，带来对 IT 基础设施和应用架构方面进行转型升级的要求。例如为了支撑电商促销活动对银行带来的高峰期海量支付请求，某银行很早就对支付渠道相关业务应用进行微服务架构改造，由此带来了容器技术的研究和运用。此银行的多年实践证明，采用容器技术平台很好地支撑了新的业务模式和业务容量。

基于业务发展的需要，和快速进步的金融科技技术，越来越多的传统银行在思考自身的互联网金融战略、金融云规划等。其中重要内容之一，是希望从技术层面更有效地支持业务创新，如微服务架构、更好的灵活性、扩展性、高可用性、更高效的业务上线效率等，因此跟上云计算技术发展的趋势，建设并推广适合自身的基于容器技术的云平台是关键任务。

2 需求分析

银行建设容器平台，不仅需要为基于微服务架构的新业务提供容器化运行和管控平台之外，还必须非常重视满足金融行业严苛的监管和安全要求。这样的定位决定了在银行建设容器平台除了要具备市场上大多数容器平台产品的能力，还应该为银行的特殊监管需求进行定制。

因此制定银行容器平台的需求时，建议考虑包括的方面有：

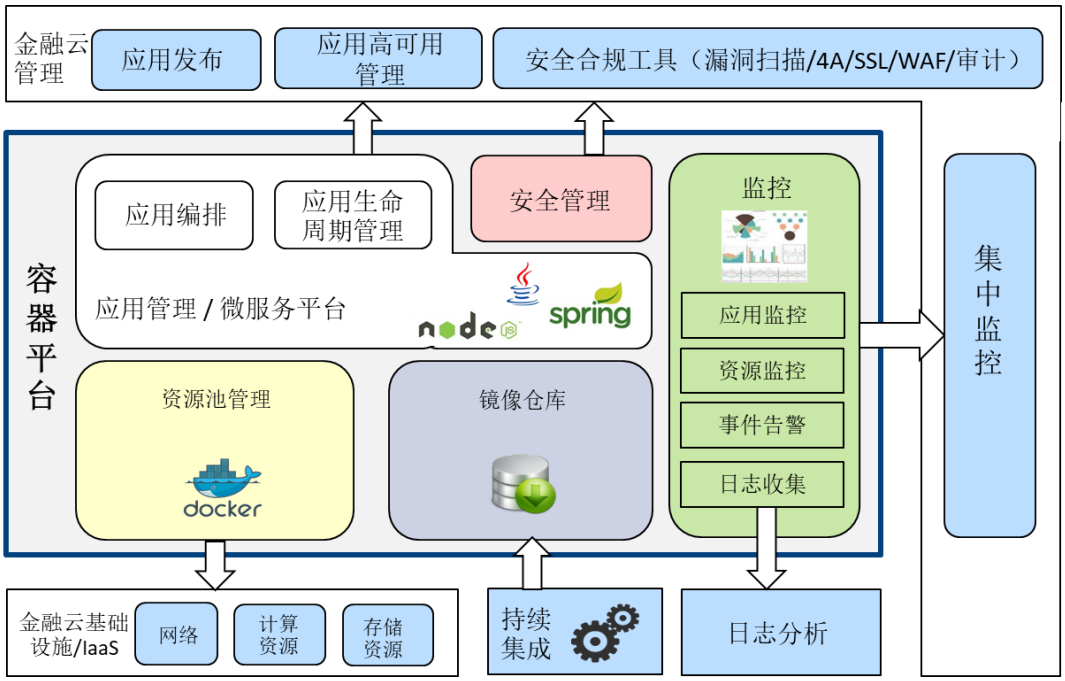
- 管理大规模容器集群能力，包括：提供容器所需的高可用集群、资源池管理、网络通信方案、存储方案、编排调度引擎、微服务运行框架、镜像管理、事件告警、集群监控和日志收集等
- 为满足金融业务的监管和安全要求，平台需要考虑应用的高可用性和业务连续性、多租户安全隔离、不同等级业务隔离、防火墙策略、安全漏洞扫描、镜像安全、后台运维的 4A 纳管、审计日志；如果容器平台还对公网提供访问，那么还需要考虑访问链路加密、安全证书等

还有一个重要方面是，银行的金融云是一个范围更大的复杂云环境，容器平台通常是这个复杂系统中的一部分，因此容器平台还要遵从银行已有 IT 技术规范和运维要求，例如可能还需要考虑：

- 支持银行自身的应用发布体系、持续集成系统、应用建模规范、高可用管理策略
- 对接金融云底层资源池（例如 IaaS），遵从云计算资源的统一管理和分配
- 对接或改造容器平台的网络，以满足容器平台中应用与传统虚拟机、物理机中旧业务系统的相互通信，避免或尽可能减少对银行现有网络管理模式的冲击
- 对接统一身份验证、和整个金融云其它系统采用统一的租户定义、角色定义、资源配额定义等
- 对接漏洞扫描、集中监控系统、日志分析系统等已有周边系统

3 业务架构

基于对容器平台的需求分析，可以用如下的业务架构图描述容器平台应提供的业务能力、以及容器平台在银行可能和周边系统的对接关系：



各业务模块提供的功能，以及可能需要集成、对接的情况是：

业务模块	能力描述	需要集成或对接
资源池管理	负责容器运行所需的计算、网络和存储资源申请、分配、容量管理，以及适合的网络通信模式	与金融云基础设施或 IAAS 对接，获取 IAAS 提供的计算节点（虚拟机和物理机）、共享存储资源，作为容器运行的资源池；同时确保容器平台采用适合的网络方案，满足网络管理和连通性要求
镜像仓库	上传、存储、拉取应用和服务的执行镜像，可以对镜像操作做权限管理	与持续集成流水线对接
应用管理 / 微服务平台	负责运行基于容器镜像的轻量级应用或微服务，提供应用的微服务编排能力、应用全生命周期管理（上架、部署、运行管理、高可用切换、升级、下架）、支持运行时动态策略调整、服务注册发现、微服务框架等功能的运行环境	对接金融云应用发布，应用编排兼容预定义的应用建模，提供金融云应用的统一发布规范； 对接金融云的应用高可用管理，提供金融云应用统一高可用部署、跨数据中心切换支持，满足金融监管的高可用性要求

安全管理	确保容器平台及应用的运行安全，包括加入 4A 权限纳管、不同安全等级应用的隔离、镜像安全、系统漏洞检测、链路访问加密、应用防火墙、操作审计等	对接金融云安全合规管理的各类工具系统
监控管理	负责容器平台中的应用日志集中收集/导出分析；同时负责容器平台运行过程中的应用监控、资源监控、事件告警等	对接日志分析系统，导出集中收集的应用日志，实现应用日志的统一收集和分析； 对接金融云集中监控系统，上报应用运行和资源使用情况、告警事件等，用于集中展现

4 关键设计

以下讨论业务架构中各个业务模块的关键设计思路。

4.1 资源池管理

容器平台资源池管理负责容器运行所需的计算、存储资源申请、分配、容量管理，以及适合的容器网络通信方案。容器网络方案比较复杂，稍后专门论述，这里先探讨关于计算和存储资源的管理。

对于计算和存储资源的申请、分配、容量管理，可能的两种做法是：

- 按照容量预估，预先为容器平台分配预测的计算节点、存储容量的资源，在容器平台中将这些资源注册到容器集群中使用。当需要扩容或删除某些资源时，重复相应的动作。
- 对接外部的资源管理和供给系统，通常是 IaaS 系统或者具备资源供给能力的自动化系统，通过调用外部系统的接口，容器平台按需获取所需的计算和存储资源

第一种做法的优势在于系统简单，不需要对接外部资源管理系统，适合技术验证平台，或容量不需要频繁变化的情况。对于生产系统或复杂的测试系统，基本上都应该考虑第二种做法，虽然带来了系统集成的复杂性，但容器平台可以借助外部的 IaaS 等系统，确保所申请的资源的高可用性，例如可以借助底层 IaaS 系统的功能，确保容器平台获得的宿主机均匀分布在不同的可用区 AZ，这些不同的可用区 AZ 可能具备不同的供电、网络连接设备等，通常对应于数据中心不同的物理区域、楼层或楼宇，由此减少某一故障导致大部分甚至全部容器宿主机瘫痪的可能性；同时，第二种做法让资源申请和获得无需人工介入，因此能做到容器平台资源的按需自动申请、弹性扩容。

目前市场上的容器平台开源技术方案，或商业容器平台，大多具备了和 IaaS 系统的集成能力，或者需要开发的相关集成逻辑也并不复杂，例如只需通过 IaaS 的接口获取虚拟机，并用自动化任务在虚拟机中执行容器平台添加计算节点的命令，即可完成从资源申请到自动加入容器平台的完整过程。

4.2 网络设计

4.2.1 技术方案选择

在资源管理中，网络的管理是比较复杂的。对于容器平台可能的网络方案，基本上分为以下几类：

- 原生 NAT 方案
- 隧道方案（Overlay），代表性的方案有 Flannel、Docker Overlay、OVS 等
- 路由方案，代表性的方案有 Calico、MacVlan
- 自定义网络方案

原生 NAT 方案中，容器借助宿主机端口映射、以及在宿主机上配置的 iptables 规则，对容器的网络数据包进行 NAT 转换，再通过宿主机的路由转发实现不同容器间跨主机的网络通信。这种方式的优势是原生支持、简单、容器实例不需要额外消耗骨干网络 IP 地址、也不会增加在宿主机间传递数据包的长度；但是缺陷也是明显的：

- 同一宿主机上不同容器在宿主机上的映射端口必须区分开以避免端口冲突
- 容器迁移到不同宿主机时，很可能需要改变所映射的宿主机端口，控制比较麻烦
- 通过 NAT 通信使得容器网络数据包在骨干网上使用的不是自身的 IP，给防火墙策略带来不便
- 端口映射带来的网络性能损失，笔者自己的环境下测试结果是，使用 NAT 方式的容器在进行跨宿主机通信是，吞吐率只能达到宿主机间吞吐率的 1/3

因此，原生的 NAT 网络比较适合小规模的功能验证和试验环境，网络性能不是重要的考虑因素，测试的场景中也不涉及很多容器迁移、防火墙安全等问题。很显然，在银行正式的测试环境、生产环境下，采用原生 NAT 方案不足以满足功能、性能和安全监管要求。

隧道方案，也就是 Overlay 方案，借助容器宿主机网络，构建出一个对于容器来说三层路由可达虚拟网络。具体做法是通过把容器网络数据包整体封装放进宿主机网络报文，由宿主机网络转发到目标容器所在的宿主机，再由目标容器所在的宿主机对报文进行拆解得到容器网络数据包，交给容器网桥，由容器网桥再转发到目标容器。隧道方案的好处是没有 NAT 方案的端口冲突问题、不消耗额外的骨干网络 IP、与现有网络技术不产生冲突因此灵活性大、通过构建不同的 VLAN 虚拟网络很容易实现网络隔离、网络性能损失比原生 NAT 方案小，在满足银行业务功能和安全监管要求的同时，对性能也有一定的照顾。因此隧道（Overlay）方案目前是应用比较多的选择，在技术上的可选择性也最大，方案成熟度也比较高，所以相对其他的方案，隧道方案的实施、定制化、维护的成本比较低。不过事情都有两面，如果选择隧道方案，您还是有一些不可回避问题需要考虑解决：

- 如果容器平台中运行业务与其它平台上运行业务需要通信，则需要配置从容器外部访问容器的路由，否则容器的地址从容器平台外部不能直接路由访问。由于容器动态变化和跨主机迁移的特点，配置从外部访问容器的路由是一个比较复杂的问题，不仅需要在外部路由器、宿主机路由表中进行配置，还需要将这些配置动作和容器的启停联动起来，这需要复杂的 SDN 能力
- 由于容器网络数据包在底层网络上传输时被封装在宿主机网络报文中，因此对普通防火墙来说，容器网络数据包的地址不可见。如果需要对容器数据包进行精确的防火墙规则设置，代价很大，几乎不可行；变通的方法可以考虑把需要进行网

络隔离的容器，在启动时指定所在的 VLAN，通过不同的 VLAN 来实现隔离

- 由于容器网络数据包需要被封装在底层宿主机网络报文中，因此会增加底层网络数据包的长度，当您的底层网络也是 Overlay 网络，或者 Overlay 的层数较多时，会影响网络数据包承载数据的效率，并且封装和拆解数据包的次数也会显著影响网络的传输效率，对于性能关键的场景，这个问题也需要引起重视

路由方案通过路由技术从三层或者二层实现跨主机容器互通，没有 NAT，没有 Overlay 方案需要的数据包封装和拆解，每一个容器具有路由可达的 IP 地址，且可以做到从容器平台外部路由可达。这种网络方案的好处是性能损失小、外部路由可达、传统的防火墙仍然能正常发挥作用等。但缺点是：

- IP 地址消耗大，如果容器数量规模大，特别是采用微服务架构后，大量的容器 IP 资源被消耗，且可能出现大量 IP 冲击到路由表里，导致路由效率降低等问题
- 容器平台外部对容器网络中网络实体的变化仍不能感知，例如新的容器创建或发生跨主机迁移时，以 Calico 方案为例，Felix 和 BGP client 模块可以保证容器网络内的路由策略更新，但由于容器平台外界不能感知容器的变化，外部到此新创建容器的路由则没办法被自动创建，仍然需要额外的路由更新工作补充

再来探讨自定义网络方案，本文所提的自定义网络方案泛指针对自身特定需求，而在既有网络技术基础之上进行改造，或者将不同的网络技术进行整合、打通而形成的特殊方案。例如在某银行，容器平台作为整个金融云平台的一部分，在设计容器网络时，需要考虑整个金融云网络管理上的一致性，由此要求容器网络具备和底层宿主机网络相同的网络层级、IP 地址规划、子网划分规则，并能够实现容器实例和容器平台以外的直接路由互通，以便能够对容器网络复用既有的 SDN 控制器管理、防火墙、安全漏扫、网络抓包分析工具等的能力。因此该银行在建设自己容器平台网络时，对接了 IAAS 的 Neutron+SDN 进行统一网络管理，工作原理是：

- 当容器平台需要为新的租户分配网络资源时，通知 Neutron，由 Neutron 对接的 SDN 控制器按照预先定义的规则为容器平台分配所需的子网和网络地址空间
- 开发网络驱动，实现 CNI 接口。当容器平台创建新的容器实例时，网络驱动调用 Neutron 接口创建 port，为容器实例在子网中分配 MAC 和 IP，并把 IP 绑定到容器网卡（类似 nova-compute 的工作），然后通知 Neutron 容器 IP 配置成功
- 容器平台记录容器的 IP 地址，作为进行服务注册、服务发现、监控服务的基础
- Neutron 和 SDN 联动，完成为容器实例设置相关的路由策略、防火墙策略等

这种方案的效果即保证网络效率、也能完全融入现有网络管理体系、还能做到完全的 SDN 联动，但复杂程度高，定制的成本也比较大，且由于完全基于路由而没有 Overlay，也存在 IP 地址消耗比较大的问题。

需要声明以上只是以某个特定银行的定制方案为例，读者所在的银行和企业可能有自己对容器网络的需求，以及自己的技术考虑，因此自定义网络方案的可能性多种多样，最终实现的能力和代价也差别很大。

如何选择容器平台的网络技术方案会相当复杂，涉及技术、场景、人员技能、成本等多方面。容器平台网络方案直接关系到平台的容量、效率、实施和运维成本，因此选择需要充分论证，考虑容器平台的定位、规模发展、承载的业务场景、对现有网络的影响、对与集中监控系统、安全合规检查系统集成的影响等，需要认真评估需求、平衡收益和成本。

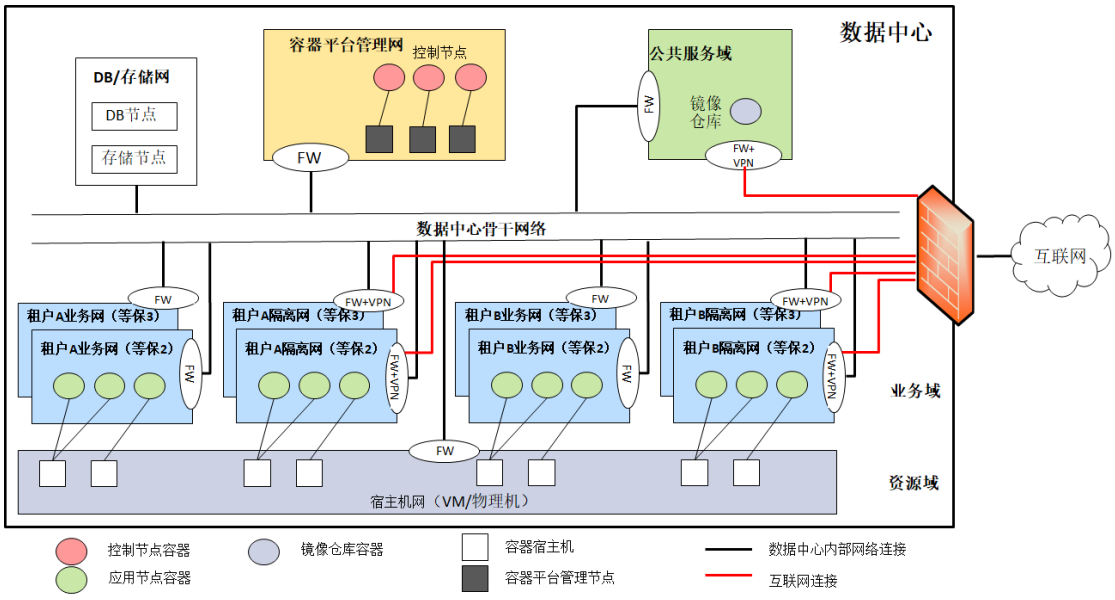
4.2.2 网络拓扑规划

除了技术方案，网络拓扑规划是网络设计的另一个重要方面，不仅涉及网络管理复杂度，还直接关系到安全合规。传统上银行科技部门会为不同安全等级的应用划分不同的网络区，分别提供不同的安全等级保护；也可能会根据运行业务的特点，分为可直接对外提供服务的网络隔离区，和只在内部运行业务处理和数据处理的业务区、数据库区等。在规划容器平台的网络拓扑时，建议保留这些已经成熟并实践多年的网络区域划分方法，保持遵守对安全合规的监管要求。

同时，根据对容器平台的定位和管理策略，容器平台可能需要在传统的网络拓扑上做相应的扩充，例如：

- 如果容器平台是金融云的一部分，网络拓扑必须支持多租户的隔离
- 容器平台中的容器和宿主机都运行在网络中，容器运行应用属于业务，而宿主机运行容器属于资源，建议把容器所在的业务域和宿主机所在的资源域划分到不同的网络区，分别使用不同的管理和访问策略，保留足够的灵活性满足不同的用户需求
- 容器平台自身运行所需的管理节点、镜像仓库、计算节点可以考虑放到不同的网络区，以满足它们各自不同的运行要求。例如镜像仓库可能需要提供对公网的服务，以便用户从公网浏览和管理镜像、管理节点可能需要运行在支持带外管理的网络区等

用下图总结以上探讨的银行如何规划容器平台网络拓扑的内容：



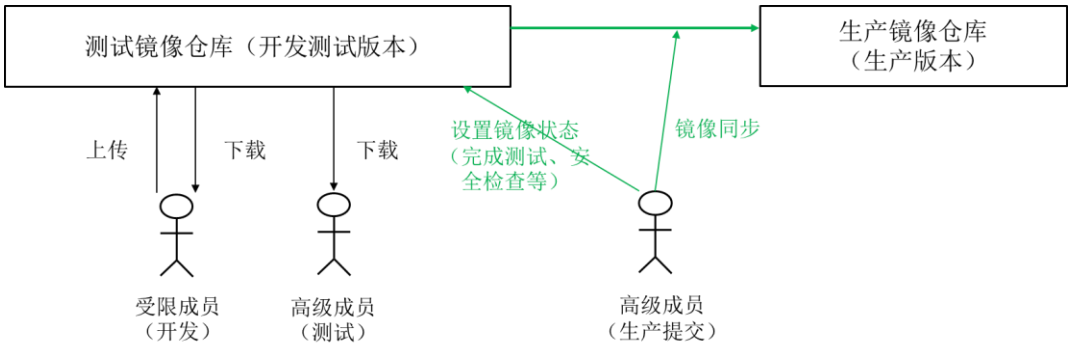
4.3 镜像仓库

镜像仓库负责存储和发布应用的镜像部署版本，在功能上并不复杂，但由于监管要求和业务的特殊性，银行高度关切生产环境的安全性，都要求用于生产发布的镜像版本必须通过严格的测试阶段，以及严密的安全检查步骤，因此建议对生产环境运行专用的生产镜

像仓库；同时，在持续集成越来越普遍的情况下，为了保证开发和测试的方便，我们需要测试镜像仓库。建议生产镜像库和测试镜像库在物理上分开、网络上的连通通过防火墙策略做限制（只开放必须的端口用于镜像同步）。

在使用规则上，测试镜像仓库允许随时的镜像上传和更新，通常都会对接持续集成系统；而对于生产镜像仓库，为了保证镜像来源的安全、可控，建议限制为只能从测试镜像同步，规定只有在测试镜像仓库中标记为完成测试、经过安全检查的镜像，由有相应权限的账号，在经过必要的审批或者满足一定规则的情况下，从测试镜像仓库中把镜像同步到生产镜像仓库。一旦镜像进入生产镜像仓库，就被当做正式的生产发布版本，接下来就按照银行现有的生产发布和变更流程，在指定的变更窗口，从生产镜像库中拉取镜像进行部署，这样做也很好地满足了银行的安全监管要求。

下图总结建议的镜像仓库体系、和相关工作流程：



4.4 应用管理

应用管理负责运行基于容器镜像的轻量级应用或微服务，提供应用的微服务编排能力、应用全生命周期管理。

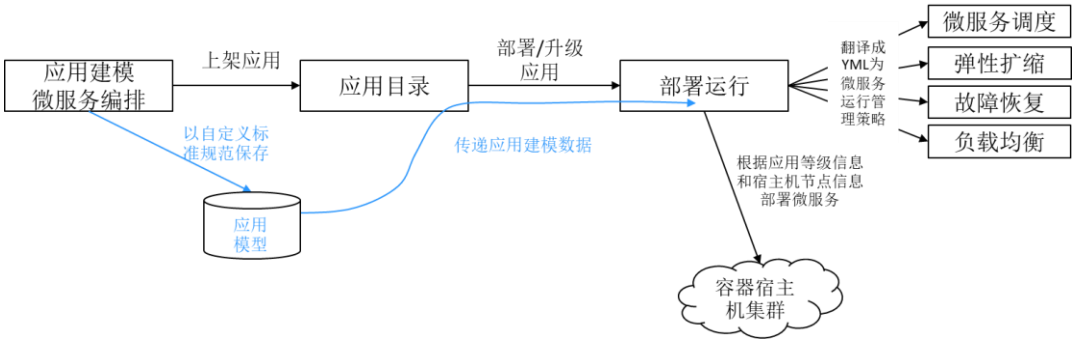
4.4.1 应用编排

应用编排的目的是为了给容器平台上运行的应用进行建模标准化，描述应用运行的资源需求、部署模式、部署参数、运行时动态规则（弹性伸缩、故障迁移等）。目前开源和商用容器平台都已支持自己的应用编排，例如 Kubernetes 的 yaml 文件方式，但对银行来说，可能还存在一些不足：

- 对银行的特定需求支持不足，例如银行应用的安全等级、部署的网路区等这些特殊信息的描述
- 不同的容器编排系统、甚至同一编排系统的不同版本，可能存在编排语法不同、不兼容的问题。银行的应用建模是重要的资产，不能允许由于版本升级、技术改造而导致众多应用的建模不兼容

因此建议容器平台自定义应用编排规范，如果容器平台定位为银行整体金融云的一部分，那么容器平台的应用编排应兼容整体金融云的应用建模规范，确保金融云上所有应用建模的一致性。

在用自定义的编排规范对应用进行标准化描述后，需要对底层的容器平台进行能力扩充定制，对应用编排信息进行翻译，变成容器平台可以理解的信息，再根据这些信息对应用进行部署、升级和运行管理。下面的图描述了应用建模、以及使用应用建模进行部署、升级和运行管理的过程。



4.4.2 生命周期管理

应用全生命周期管理负责应用的上架、部署、升级、下架、支持运行时动态管理策略，还可支持双活部署、同城灾备切换等金融云高级能力。这部分功能可能需要对接金融云的应用发布、高可用部署和切换模块，提供整个金融云所有应用统一的部署、高可用体验。在上一节应用编排，我们讨论了有关上架、部署、升级、运行管理等，这里来看应用的高可用部署和切换。

容器平台可以从实例、服务、应用三个层级，分别实现应用的高可用，分别是：

- 实例级，即容器故障自动恢复
- 服务级，即服务/微服务的多个实例的跨不同可用区部署
- 应用级，即应用跨数据中心切换

从单个运行实例级别，可以采用容器故障自动恢复机制，对发生故障的容器实例进行快速的故障发现、自动迁移和恢复。基本上开源和商业的容器集群管理系统都支持按照用户预定义的规则，进行故障检测和实例恢复。

从单个服务级别，可以把一个服务或微服务的多个容器运行实例，在跨多个可用区中进行分布部署。在容器平台在进行资源池管理时，借助底层 IaaS 平台，确保容器宿主机均匀分布在不同的可用区 AZ 中。当在容器平台部署一个服务或微服务的多个容器实例时，尽量把多个容器实例调度运行在处于不同可用区 AZ 的宿主机上，这可以借助容器调度策略、以及事先对宿主机加标签以方便识别所在的可用区来配合完成。

绝大多数情况下，我们都有机会从实例级别、服务级别进行努力，提高应用的高可用性。

现在越来越多的银行都在建设自己的多地或多中心系统，或者即有本地私有数据中心，也同时具备云端的数据中心。如果您有这样的基础设施，或有相关的建设目标，那么您还可以从更高的级别，即从整个应用的级别来考虑高可用性。做法是在多个数据中心，对应用进行多活、或者主备的部署，把业务流量按照合适的比例分发到不同的数据中心进行交易处理。多数据中心部署的一个关键问题是交易数据的一致性，由于分布式数据库在数据一致性上目前并不成熟，分库分表又会带来额外的关联复杂性，因此大多数银行当前阶段选择起来还很谨慎，而会继续采用单个集中的主数据库，加上位于不同数据中心的备库，之间通过数据库自身的同步技术、加上底层存储系统的快速同步，确保备库和主库的

数据保持一致，在进行数据中心切换时，尽可能地减小数据丢失，即减小 RPO。采用主备库同步的方案，会对网络低延迟有比较高的要求，这限制了不同数据中心间的最远距离，通常位于同一城市区域、相距数十公里以内是比较可行和普遍的。

4.5 安全管理

安全管理是满足行业监管要求必须考虑的问题，是银行建设容器平台的特殊要求。

安全管理的难点在于涉及面广，包括系统漏洞、病毒威胁、链路加密、攻击防范、系统访问权限上收、操作审计等，此外安全管理面对的安全威胁不断地发展变化，也增加了防范的技术难度和持续的工作量。同时金融云和容器自身的特点，在传统银行安全管理的基础上，还增加了多租户隔离、角色管理、镜像安全检测等新问题。

4.5.1 对接安全合规体系

鉴于安全管理的复杂性，如果在容器平台中单独进行安全管理，代价很高；而且安全管理也十分依赖长时间的积累，容器平台单独进行安全管理，也难免在一段时间内出现各种安全问题纰漏。因此建议容器平台在安全管理上直接对接银行现有的安全管理防范体系，充分利用现有的各类安全工具、手段，在现有安全管理手段的基础上，按需增加功能应对容器平台带来的新需求新问题。这应该是见效快、成本低、风险也比较低的方式。

银行面对严格的行业安全监管要求，现有的安全管理防范体系，包括技术工具、工作流程和指导规范等都已经相当完备，例如系统漏洞扫描、补丁安装、防病毒系统、SSL 和证书申请、WAF、统一身份认证和 4A 访问管理、操作日志审计等。为了利用上这些能力，需要在容器平台设计，特别是网络方案的设计上，仔细考虑如何才能让这些工具和手段对容器平台发挥作用。例如某银行系统漏洞扫描的方式是由运行在网络中的扫描服务，定期地通过向被扫描目标 IP 发送端口检测报文，分析响应结果来判断是否存在系统漏洞，做出是否需要安装补丁或者关闭被扫描目标相关系统服务等决定。为了让系统漏洞扫描服务能对容器平台正常工作，那么在设计网络方案时，让漏洞扫描服务与容器平台中的容器 IP 路由可达就是必须要考虑的问题。

4.5.2 多租户隔离

如果容器平台作为金融云的一部分，并计划为不同的租户提供服务，那么根据租户对安全的要求，支持不同租户的隔离也是要考虑的内容。

在之前讨论网络拓扑规划时，建议把不同租户的容器运行在各自不同的虚拟网络 VLAN 中，并为不同的 VLAN 设置必须的防火墙规则、关闭相关的路由来保证不同租户的业务在网络上隔离。

由于容器共享宿主机内核的特点，如果把不同租户的容器运行在同一台宿主机上，租户可能面临来自其他租户容器运行带来的不利影响，例如：

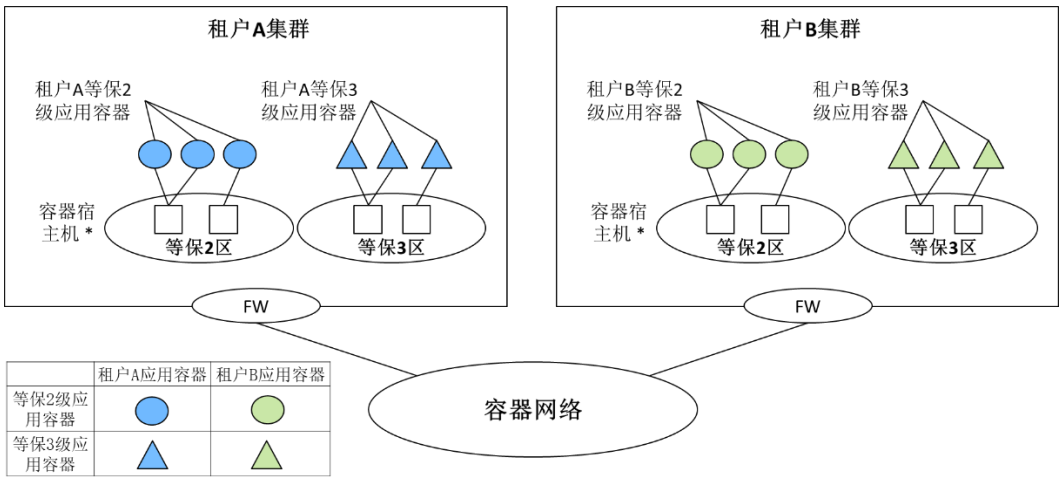
- 资源竞争导致的性能下降
- 其他租户容器应用的 bug 导致的宿主机内核运行异常，进而导致自己租户容器的运行故障
- 潜在的来自其他租户的恶意容器应用，利用共享内核进行攻击和窃密

因此，建议容器平台为不同的租户分配各自专属的、不同的资源池，租户只能在属于自己的宿主机上运行自己的容器应用。这虽然导致了资源利用率的降低，但在根本上回避了容器运行依赖共享宿主机内核、隔离性天生不如虚拟机的局限，这和主要基于虚拟机的IaaS 平台对多租户隔离的做法不同。

4.5.3 应用等级隔离

除了不同租户间的隔离，即使在同一租户下，运行不同安全等级的应用，因为容器共享系统内核的特点，应用也面临其它等级应用的资源争抢、故障影响等问题。另外，不同等级的应用，往往要求不同级别的运行环境高可用性、安全性，因此在同一租户下，也应该把不同等级的应用隔离开，分别部署到各自专属的资源池内。

下面的图以两个租户、分别有不同的安全等级的应用部署为例，描绘应用的部署状态：



4.6 监控日志

4.6.1 监控

和安全管理类似，监控体系也在银行发展多年，特别是针对生产系统的监控、告警体系，根据自身运维的需要，不断积累、优化了多年，大多已经比较完备；围绕目前的监控体系，也形成了成熟的应急方案、流程，人员技能和经验也多围绕既有生产监控系统进行培训、学习。

因此，如果容器平台没有特别的需求，在银行的生产环境下，建议将容器平台的监控体系对接目前的集中监控系统，方便运维人员对生产环境的统一监控管理，既有的应急方案、流程、人员技能和经验都可以得到沿用。

即使容器平台有不同于传统的监控需求，例如：

- 容器运行不再关注单个容器实例的增加和消失，而只关注整体服务的可用性
- 新的业务应用通过输出标准化日志，对日志进行分析提取业务性能数据、成功率

出现这些不同的需求，也建议用集中监控系统进行展示和告警，只是需要在集中监控

系统之外先进行处理，再把处理后的结果对接到集中监控系统。例如，我们不用再在集中监控系统中配置去监控每一个容器实例的 IP，而是由容器平台检查服务所需要的容器实例数量是否还在允许的最小值以上，只有当实例数量低于接受的最小值，才给集中监控系统发送告警，其它情况下只需要汇报当前实例数量即可，不在乎实例数量的波动。

在设计具体的监控时，应把监控进行分类，分别处理。具体可分为：

- 应用和服务监控
- 资源监控
- 平台监控

应用和服务监控关心业务服务的正确工作状态，这可能需要通过调用平台 API、或通过应用日志分析、特定端口响应等方法来判断，需要开发一定的逻辑处理，再把结果对接到集中监控。

资源监控主要关注每个宿主机、以及计算节点集群的整体资源的使用情况，是否需要增加节点扩容等，这一点基本上传统的监控体系都已经能够做到，方式上以在宿主机上运行 agent，进行资源数据收集、然后上报为多。

平台的监控关注容器平台的控制节点、数据库、提供的服务等是否工作正常，这一点通常开源和商业的容器平台自身就已提供相应的管理控制台。如果不介意界面风格的差异，集中监控系统可以直接嵌入或跳转到容器平台的管理控制台；如果为了一致的监控体验，或者需要进一步的监控和告警定制，就需要开发集成逻辑，通过调用容器平台的 API，对获得的数据进行处理、封装，再对接到集中监控系统。

4.6.2 日志

在容器平台中，日志大致分为两类：

- 环境日志，包括容器运行日志、宿主机容器引擎日志、容器平台管理日志
- 应用日志，指运行在容器中的业务应用在进行业务处理中，对处理过程中的关键结果、状态所进行的记录

环境日志有各自固定的输出位置，主要用于出现故障时进行运维排查，和容器平台运行的业务并无直接关系；对于容器平台，更需要关注、处理的是应用日志。因为以容器为载体，以分布式方式运行，无论是运行位置、数量等都会随时发生变化，所以如果不对应用日志做特别处理，应用日志会散布在容器集群的任意节点上。传统的方式，即登录到某一个特定的节点上去查看日志，已经不能适用于容器平台中的应用了。

我们必须对应用的日志进行集中收集，相关的开源方案选择也比较丰富，例如 ELK、Fluentd 等，或者直接通过在容器中挂载 NFS 共享文件系统，把业务运行的日志实时写到共享文件系统中进行集中收集。

5 总结和建议

容器平台的建设要考虑场景、技术、系统对接、成本、人员技能等因素，有不小的复杂度。以上各部分从一个最精简容器平台所需考虑的各个方面，结合作者的项目实践，提出供大家参考的建议。

容器技术发展非常迅速，相关的开源社区和厂商都在努力贡献，新技术层出不穷。同时，微服务框架的功能越来越完备，随着微服务架构思想不断普及，不久的将来会有越来越多的新业务基于微服务架构的形式出现，而容器是微服务架构应用的最理想运行环境，这就意味着容器平台未来的应用场景也是不断改变、进化和越来越丰富的。

在目前阶段，容器平台的建设是一个不断改进，迭代积累的过程。一开始可能没办法在各个方面都能想得很清楚，不仅在技术方案，更在业务场景、运维方式、安全合规等方面，都需要很多的讨论。但不应该怀疑这个大的趋势，看看 Google、Netflix 等国际大厂商在容器、微服务上的成功，以及国内的互联网厂商在容器和微服务上不断的投入和积累，我们应该相信这个方向并持续地发展下去。近几年在人民银行年度的银行获奖科技项目中，有关容器技术的项目逐渐增多，也是肯定在这个方向发展的标志。

技术和市场不会停下来等任何人，如果您想跟上技术的趋势，请在您的组织内给容器平台、微服务推广等工作足够的信任，投入上从小规模试验做起，积累经验后慢慢推广，随着技术和和市场的成熟，更多的微服务架构应用运行在容器平台上，为业务带来服务能力和效率提升，也为您的投入带来应有的回报。