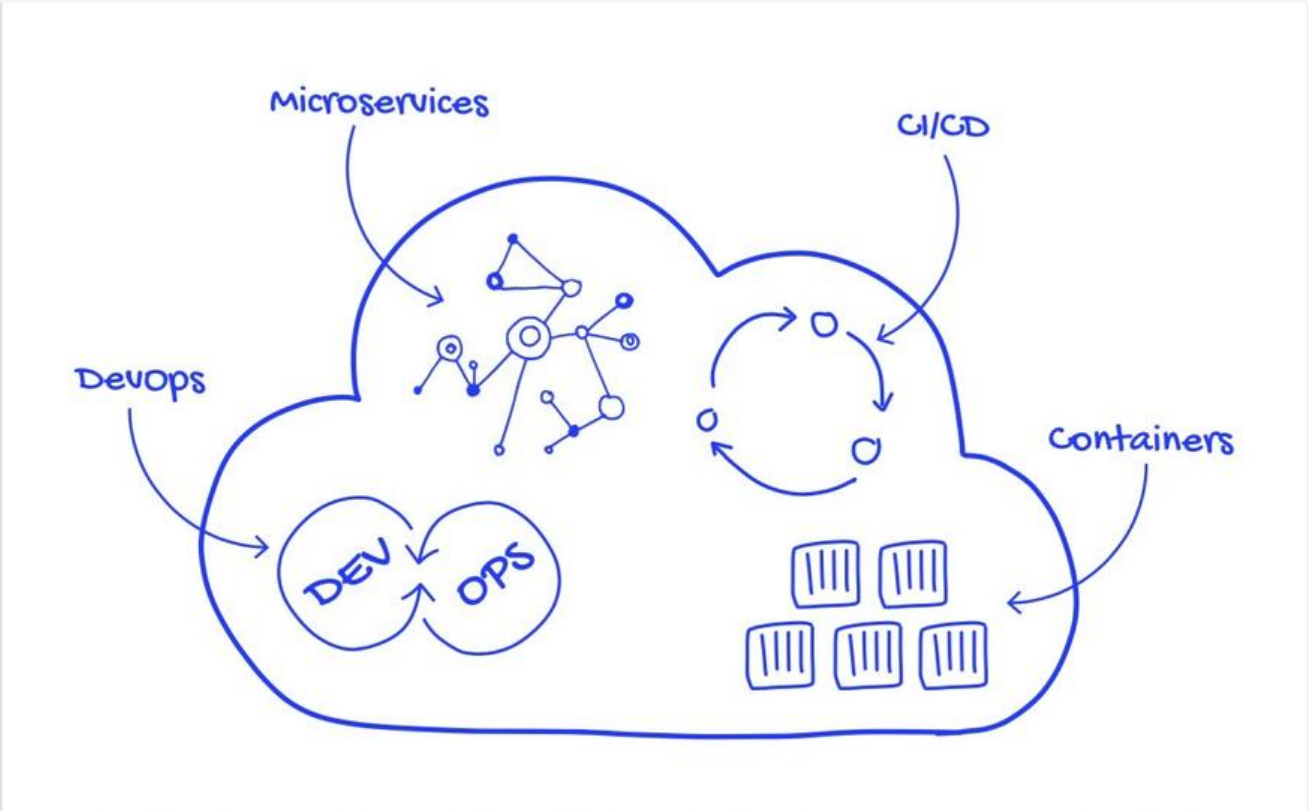
【译】如何设计一个云原生应用的架构?

作者：panjf2000
原文链接：https://ld246.com/article/1591588094418
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)



本文原作者 Siddharth Patnaik 曾任华为的 Lead Architect，现在是沃尔玛旗下 Walmart Labs 的 Prncipal Architect，本文是他在云原生领域多年架构经验的总结与归纳。

导言

云原生 (Cloud Natvie) 是一种将应用程序以微服务的形式构建并使之运行在容器化和动态编排平台上的方式，这些平台充分利用了云计算模型的优势。**云原生关注的是应用如何构建和部署，而非运行哪里。**这些技术能够赋能组织/企业在公有云、私有云和混合云等现代和动态的开发环境里构建和运可扩展的应用程序。这些应用程序是从头开始构建的，以松散耦合原则进行系统设计，并且专门针对级别的规模和性能进行了优化，基于托管服务并利用持续交付来实现可靠性和更快的上市速度。总体标是提高速度、可扩展性，以及最终提高利润率。

速度 -- 各类规模的公司现在都发现了一种能够快速行动并迅速将创意推向市场的战略优势。通过这方式，意味着以前那种数以月计才能将一个想法落地成一个实际产品的周期变成了现在的几天甚至几时。实现这一目标的核心是企业内部的文化转变，从以前那种『大爆炸』式项目一步到位的方式过渡更多的渐进式改进。从本质上讲，云原生 (Cloud Native) 策略是关于如何解决技术风险的。过去，们规避风险的标准方法是保持缓慢而谨慎地行动。云原生 (Cloud Native) 的新方法则是依靠采取小、可逆和低风险的步骤来实现迅速前进。

可扩展性 -- 随着业务的发展，在更多的地方为更多的用户提供更广泛的设备支持，同时又要维持灵响应和管理成本，而不会导致失败，成为了战略需要。

利润 -- 在云基础设施的新世界中，战略目标是只在新客户上线时购买所需的额外资源。支出从前期的 CAPEX（购买新机器以预期成功）变成了 OPEX（按需支付额外的服务器）。

CNCF 的角色

云原生计算基金会是一个归属于 Linux 基金会的开源软件基金会，其中包括 Google (谷歌)、IBM、In

el (英特尔)、Box、Cisco (思科)和 VMware 等大公司都在致力于使云原生计算具有普遍性和可持续。云原生计算利用开源软件技术栈将应用程序以微服务的形式进行部署，将每个部分打包进自己的容中，并动态编排这些容器以优化资源利用率。

为什么需要 CNCF？根据他们自己的 FAQ，答案是：

- 公司们正在逐渐意识到它们需要成为一家软件公司，即便它们其实并没有软件业务。例如，因为 Airnb 这家公司对酒店住宿行业的革命性改变，导致更多传统的酒店正在挣扎着与之竞争。
- 云原生允许 IT 和软件更快速地迭代。
- 采用云原生技术和实践使公司能够在内部创建软件，使业务人员能够与 IT 人员密切协作，不落后竞争对手，并为客户提供更好的服务。

云原生设计原则

设计为松散耦合的微服务

微服务是一种将单体应用程序拆解成一套小型服务的方法，每个服务都在自己的进程中运行，并使用像 HTTP 等轻量级协议进行通信。这些服务是围绕业务功能构建的，可以通过完全自动化的部署机制独进行部署。

使用最佳的语言和框架组合进行开发

每个云原生应用程序的服务都应该使用最适合该功能的语言和框架来开发。云原生应用程序是多语言，服务使用各种编程语言、运行时和框架。例如，开发人员可以使用 Node.js 来开发基于 WebSocket 的实时流服务，同时选择 Python 来构建机器学习基础服务，并选择 spring-boot 来提供 REST APIs 接口。这种细粒度的微服务开发方式使得开发者能够选择最合适的语言和框架来开发去开发特定的任。

以互动和协作为中心的 APIs

云原生服务使用轻量级 APIs，这些 APIs 基于表述性状态转移（REST）等协议来暴露其功能。内部务使用 Thrift、Protobuf、gRPC 等二进制协议进行通信，以获得更好的性能。

无状态且可大规模扩展

云原生应用将其状态存储在数据库或其他外部实体中，因此实例可以随意上下线，任一实例都可以处请求。实例不依赖于底层基础设施，底层基础设施允许应用程序以高度分布式的方式运行，并且仍然持其状态独立于底层基础设施的弹性机制。从可伸缩性的角度来看，架构非常简单，只需向集群添加通的商用服务器节点就可以扩展应用程序。

复原韧性作为架构核心

根据墨菲定律 - “凡是可能出错的事就一定会出错”。当我们将其应用于软件系统时，分布式系统也发生故障：硬件可能会出错、网络可能会出现暂时的故障，甚至有极低的概率整个服务或区域也可能出现中断，但即使是概率极低也必须做好预案。复原韧性是指系统从故障中恢复并继续运行的能力。不是故障规避，而是以避免停机和数据丢失为目标去处理故障。复原韧性的目标是在发生故障后将应程序恢复到能完全正常运行的状态。复原韧性提供以下内容：

- 高可用性 - 应用程序保持健康状态继续运行的能力而无需大量宕机。

● 灾难恢复 - 应用程序从罕见但重大意外事件中恢复的能力：非临时、大规模的故障，比如波及整个域的服务中断。

使应用程序更具韧性的主要方法之一是通过冗余机制。HA 和 DR 高可用方案使用多节点集群、多区部署、数据复制、无单点故障、持续监控等实现。

以下是实现复原韧性的一些策略：

● 重试临时故障 - 临时故障可能是由于网络连接短暂丢失、数据库连接突然掉线或服务繁忙导致的超造成的。一般这种临时故障能通过简单重试请求来解决。

● 跨实例的负载均衡 - 万物集群化。无状态应用程序应该能够通过向集群添加更多节点进行扩展。

● 优雅降级 - 如果一个服务故障且没有故障转移路径，应用程序应该要能优雅降级，同时持续提供可受的用户体验。

● 限流高吞吐的佃户/用户 - 有时候一小撮用户可能会产生过多的负载。这将会对其他用户产生影响从而降低应用程序的整体可用性。

● 使用断路器 - 断路器模式可以防止应用程序反复尝试一个可能失败的操作。断路器封装了对服务的用并跟踪最近的失败次数。如果失败次数超过阈值，则断路器开始返回错误代码而不再是调用服务。

● 应用补偿事务 - 补偿事务是一种撤销另一个已完成事务造成的影响的事务。在分布式系统中，实现务强一致性可能非常困难。补偿事务是一种通过使用一系列更小的单个事务来实现一致性的方法，这事务的每个步骤都可以被撤消。

测试复原韧性 -- 通常复原韧性测试不能像测试应用程序功能一样（通过运行单元测试、集成测试等。相反，你必须测试在间歇性发生故障的情况下端到端的工作负载情况。例如：通过崩溃进程注入故、过期证书、让相关服务不可用等。像 chaos monkey 这样的框架可以用于这种混沌测试。

打包成轻量级容器并编排

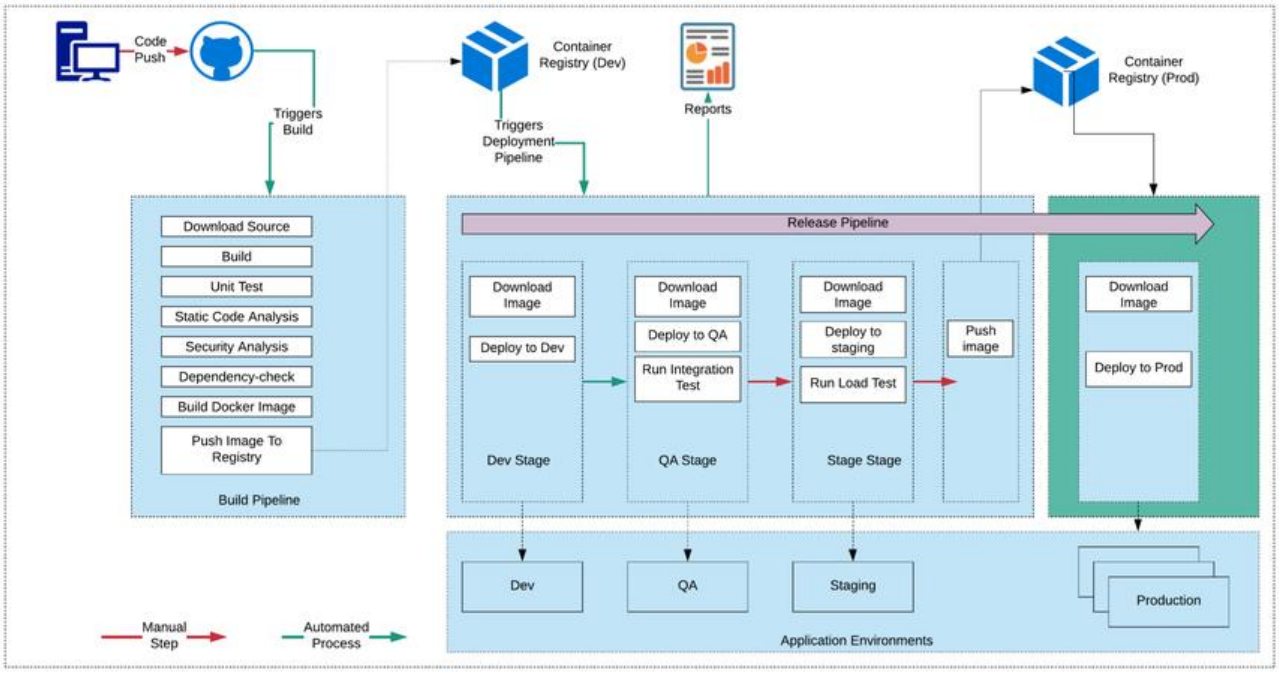
容器可以将应用程序隔离至共享操作系统内核的小型轻量级执行环境中。容器大小通常以 MB 为单位因此所使用的资源远远少于虚拟机，并且几乎是瞬时启动。Docker 现在已成为容器技术的标准，它供的最大优势是便携性。

云原生应用程序使用 Kubernetes 进行部署，Kubernetes 是一个开源平台，旨在自动部署、扩展和理容器化应用程序。Kubernetes 最初由谷歌开发，现在已经成为部署云原生应用程序的操作系统。也是首批从 CNCF 毕业的项目的其中之一。

通过 CI/CD 实施敏捷 DevOps 和自动化

DevOps，是“开发”和“运维”的融合，描述了实现快速敏捷开发和可扩展、可靠运维所需的组织构、实践和文化。

DevOps 是关于文化、协作实践和自动化的，它要求开发和运营团队保持一致，在提升客户体验、更地响应业务需求、以及确保业务创新能在安全性和运营需求之间保持平衡方面形成一个统一的思想。代的组织笃信开发和运维的人员职责的合并，可以让一个 DevOps 团队同时承担起这两项责任。通过种方式，你只需要一个团队来负责开发、部署以及在生产环境运行软件。



持续集成（CI）和持续交付（CD）是一组操作原则，它们使应用程序开发团队能够更频繁、更可靠交付代码变更。CI 的技术目标是建立一致和自动化的方法来构建、打包和测试应用程序。有了集成过中的一致性，团队能够更频繁地提交代码变更，从而实现更好的协作和软件质量。

持续交付 (CD) 负责交接持续集成 (CI)，完成后续的工作。CD 将交付应用程序到选定的基础设施环境过程自动化，它挑选出 CI 构建完成的应用包，将其部署到像 Dev、QA、Performance 等各种环境，分阶段运行各种测试，如集成测试、性能测试等，最后部署到生产环境中。通常 CD 的 pipeline 几乎不需要人工介入，鉴于持续部署 (Continuous Deployment) 是一个完全自动化的 pipeline，从码拉取到生产环境部署的整个过程都自动化了。

弹性 - 动态扩容/缩容

得益于云计算的弹性特性，云原生应用得以在运行高峰期通过增加计算资源来缓解压力。如果你部署云上的电子商务应用正在经历一个流量高峰，你可以设置额外的计算资源给应用，等到高峰消退之后下线这些资源。云原生应用可以根据需求调整增删的资源和规模。

英文原文

Cloud Native Application Architecture

原文链接：【译】如何设计一个云原生应用的架构?