什么是云原生？

修改于 2021-08-13 18:03:23 阅读 6.7K

近年来，随着云计算概念和技术的普及，云原生一词也越来越热门，无论是应用还是安全，凡是和云相关的，都要在云后面加上原生二字，好像不提云原生，在技术上就落后了一大截。

那到底什么是云原生？云原生是怎么产生的？云原生能带来什么好处？如何实现云原生？本文将就这些问题做一个总体的概述。

1. 云原生产生背景

随着云计算技术的发展，企业上云已成为趋势，越来越多的企业都已将应用部署到了云上。但是应用上云并不意味着就能充分利用云平台的优势。目前，大部分云化的应用，都是基于传统的软件架构来搭建的，然后再移植到云上去运行，和云平台的整合度非常低，主要表现在以下几个方面：

操作系统依赖强

传统应用程序和底层操作系统、硬件、存储和后备服务之间存在紧密的依赖关系，这些依赖关系使得应用程序在跨越云基础设施进行迁移和扩展时非常复杂且有风险。

系统紧耦合

传统的企业应用多采用单体架构，将许多不同的功能模块捆绑在一个部署包中，导致功能模块之间产生不必要的依赖，并导致开发和部署过程中丧失敏捷性，无法独立的部署、发布更新、重启。

手动化扩展

通过手工管理基础设施，包括手工编写管理服务器、网络和存储的配置脚本。在大规模复杂的操作中，操作人员在诊断问题时会很慢，而且无法大规模地实施。手工制作的自动化脚本还有可能将人为错误硬编码到基础设施中。

恢复缓慢

基于虚拟机的基础设施相对于基于微服务的应用程序来说，是缓慢而低效的。因为单个虚拟机启动/关闭的速度很慢，并且在部署应用程序代码之前就会带来巨大的开销。

瀑布开发

传统应用的开发模式，IT 团队定期发布软件，通常间隔几周或几个月。尽管发布的许多组件已经提前准备好了，并且没有依赖关系，也必须等待版本中的其他组件。客户想要的功能被延迟，企业失去赢得客户和增加收入的机会。

总体来说，提供方便的基础设施，只是对云计算最初级的利用（提升利用率，按需使用，不够了随时扩容），无法充分发挥云计算的优势，要想充分发挥云计算的优势（弹性、高可用性、易扩展性），就必须进行真正的云化，不仅仅是基础设施和平台的变化，应用也需要做出改变，这就需要摈弃传统的方法，在架构设计、开发方式、部署维护等各个阶段和方面都基于云的特点重新设计，从而建设全新的云化的应用，也就是云原生的应用。

2. 云原生的定义

关于什么是云原生，不同的人定义不同，目前比较权威的定义主要来自 Pivotal 公司和云原生计算基金会（Cloud Native Computing Foundation，简称 CNCF）。

2.1. Pivotal 的定义

Pivotal 公司是云原生应用的提出者，并推出了 Cloud Foundry 和 Spring 系列开发框架，因此，也是云原生的先驱者和探路者。

早在 2015 年，Pivotal 公司的 Matt Stine 写了一本名为《迁移到云原生应用架构》的小册子，其中探讨了云原生应用架构的几个主要特征：符合 12 因素的应用、面向微服务架构、自服务敏捷架构、基于 API 的协作以及抗脆弱性。

在 2017 年 10 月，接受 InfoQ 采访时，Matt Stine 对云原生的定义做了小幅调整，将云原生架构定义为具有以下六个特质：模块化(Modularity) 、可观测性(Observability)、可部署性(Deployability)、可测试性(Testability)、可处理性(Disposability)以及可替换性(Replaceability)。而 Pivotal 官网对云原生概括为 4 个要点：DevOps、持续交付、微服务以及容器化。

图 1：Pivotal 云原生思想

Matt Stine 认为云原生是一个思想的集合，云原生既包含技术（微服务，敏捷基础设施），也包含管理（DevOps、持续交付、康威定律以及重组等），云原生也可以说是一系列云技术、企业管理方法的集合。

2.2. CNCF 的定义

CNCF 是在 2015 年由 Google 联合 Linux 基金会成立的，它是一个非盈利组织， 主要宗旨是统一云计算接口和相关标准，通过技术优势和用户价值创造一套新的通用容器技术，推动云计算和服务的发展。起初 CNCF 对云原生（Cloud Native）的定义包含以下三个方面：

- 应用容器化
- 面向微服务架构
- 应用支持容器的编排调度

到了 2018 年，随着云原生生态的不断壮大，所有主流云计算供应商都加入了该基金会，且从云原生的全景图中可以看出云原生正在蚕食原先非云原生应用的部分。

CNCF 基金会中的会员以及容纳的项目越来越多，该定义已经限制了云原生生态的发展，CNCF 为云原生进行了重新定位，并于 2018 年 6 月 11 日明确了云原生定义 1.0 版本：云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明式 API。

图 2：云原生代表技术

这些技术能够构建容错性好、易于管理和便于观察的松耦合系统。结合可靠的自动化手段，云原生技术使工程师能够轻松地对系统作出频繁和可预测的重大变更。

3. 云原生相关技术

依据 CNCF 发布的云原生 1.0 版本的定义，云原生技术主要包括容器、微服务、服务网格、不可变基础设施以及声明式 API，下面就这几类技术做个概述：

3.1. 容器技术

容器技术和云原生好比一对螺旋体，容器技术催生了云原生思潮，云原生生态推动了容器技术发展。从 2013 年 Docker 技术诞生，到 2015 年 CNCF 这个云原生领域重量级联盟成立，这不是历史的巧合而是历史的必然。作为云原生关键技术之一的容器，从 2013 年诞生以来一直是行业关注的焦点之一。

2013 年之前，云计算行业一直在为云原生的正确打开姿势而操心。Platform as a Service（PaaS）看起来是个有前途的方向。2006 年 Fotango 公司发布的 Zimi 服务，可以说是 PaaS 行业的鼻祖，具有按使用付费、免运维（Serverless）、API 化配置和服务等典型云原生的特征；2008 年 Google 推出 Google App Engine（GAE）；2011 年 Pivotal 发布 Cloud Foundry。

这些早期的 PaaS 平台在云原生领域进行了非常有益的探索，推动了云原生生态的健康发展，但是这些早期探索技术并没有形成大的行业趋势，而是局限在一些的特定的领域。直到 Docker 开源，大家才如梦方醒，原来不是方向不对，而是应用分发和交付的手段不行。

Docker 真正核心的创新是容器镜像（docker image），一种新型的应用打包、分发和运行机制。容器镜像将应用运行环境，包括代码、依赖库、工具、资源文件和元信息等，打包成一种操作系统发行版无关的不可变更软件包。

容器镜像打包了整个容器运行依赖的环境，以避免依赖运行容器的服务器的操作系统，从而实现“build once， run anywhere”。容器镜像一旦构建完成，就变成 read only，成为不可变基础设施的一份子。

3.2. 微服务

微服务架构是相对于单体架构来说的，两者分属不同的架构风格。在微服务架构中，服务是一个单一的、可独立部署的软件组件，它实现了一些有用的功能，服务的 API 封装了其内部实现，与单体架构不同，开发人员无法绕过服务的 API 直接访问服务内部的方法和数据，因此，微服务架构强制实现了应用程序的模块化。

微服务架构的最核心特性是服务之间的松耦合性。服务之间的交互采用 API 完成，这样做就封装了服务的实现细节，从而实现了在不影响客户端的情况下，对实现方式做出修改。

微服务架构通过将大的系统按照业务服务的粒度进行拆分，每个服务可独立开发、测试、验证和部署，这样分解后，带来的好处有如下几点：

使大型的复杂应用程序可以持续交付和持续部署 每个服务都相对较小并容易维护 服务可以独立部署 服务可以独立扩展 微服务架构可以实现团队的自治 更容易实验和采纳新的技术 更好的容错性

3.3. 服务网格

服务网格是用于处理服务间通信的专用基础设施层，负责在微服务间进行可靠地请求传递。服务网格通常通过一组轻量级网络代理来实现，这些代理与应用程序代码一起部署，而不需要感知应用程序本身。

随着规模和复杂性的增长，服务网格包含的实现的功能越来越多，它的需求包括服务发现、负载均衡、故障恢复、指标收集和监控以及通常更加复杂的运维需求，例如 A/B 测试、金丝雀发布、限流、访问控制和端到端认证等。其部署结构如下图所示：

图 3：服务网格部署图

服务网格有如下几个特点：

- 应用程序间通讯的中间层
- 轻量级网络代理
- 应用程序无感知
- 解耦应用程序的重试/超时、监控、追踪和服务发现

如果用一句话来解释什么是服务网格，可以将它比作是应用程序或者说微服务间的 TCP/IP，负责服务之间的网络访问、限流、熔断和监控。对于编写应用程序来说一般无须关心 TCP/IP 这一层（比如通过 HTTP 协议的 RESTful 应用），同样使用服务网格也就无须关系服务之间的那些原来是通过应用程序或者其他框架实现的事情，比如 Spring Cloud、OSS，现在只要交给服务网格就可以了，从而极大地方便了微服务应用的开发。

3.4. 不可变基础设施

一个工作负载（比如容器、虚拟机等）一旦部署以后就不会被修改。当需要更新，修复或修改某些内容的时候，只需要将新的、经过验证的工作负载替换旧的即可。

不可变基础设施的作用主要体现在系统的稳定性方面。传统的应用程序一旦部署到用户特定的服务器上以后，服务器系统是会不断变化的，不是操作系统升级，就是安装了新的应用，可能引起冲突，导致应用程序需要跟着用户系统环境的改变而不断升级，这中间就会不断地出现新的问题。而不可变基础设施就规避了所有的这些问题，因为云原生应用是部署在不可变的基础设施上的，因此就不存在变化的问题。

3.5. 声明式 API

声明式 API 是一种比命令式 API 更高级的接口设计方式，简单来说，命令式 API 提供给用户怎么做的能力，而声明式 API 给用户提供了做什么的能力。

声明式 API 是比命令式 API 更高级的一种接口，举个例子，假如我们有一个炒菜机，如果炒菜机提供的接口是放油、放调料、放食材、大火、小火等操作，那就是命令式 API。

如果炒菜机提供的接口是来盘宫保鸡丁、来盘鱼香肉丝之类的，那就是声明式 API 了。声明式 API 比较典型的例子就是数据库提供的 SQL 接口，只需要告诉数据库你需要什么数据即可，至于怎么去获取这些数据，数据库自己会去按步骤操作。

4. 云原生的意义

传统的软件开发模式，在使用云计算平台时和使用物理机时没有什么大的区别，那么就没有将云平台的能力利用充分，在一定程度上导致了资源的浪费。云原生就是用来解决这一类的问题，将云计算平台的优势发挥到极致。

让企业应用能够利用云平台实现资源的按需分配和弹性伸缩，是云原生应用重点关注的地方。它要求云原生应用具备可用性和伸缩性，以及自动化部署和管理能力，可随处运行，并且能够通过持续集成、持续交付提升研发、测试与发布的效率。云原生应用并未完全颠覆传统的应用，采用云原生的设计模式可以优化和改进传统应用模式，使应用更加适合在云平台上运行。

云原生存在的意义是解放开发和运维，而不是让开发和运维的工作变得更加复杂和繁重。云原生还关注规模，分布式系统应该具备将节点进行水平扩展的能力，能轻易地扩展到成千上万的规模，并且这些节点具备多租户和自愈能力。云原生使得应用本身具备“柔性”，即面对强大压力的缓解能力以及压力过后的恢复能力。

5. 云原生当前生态

作为云原生领域最具权威的组织，CNCF 在每年的年度报告中都会提及 CNCF Landscape 项目，该项目开始于 2016 年 11 月，旨在为云原生应用者提供一个资源地图，帮助企业和开发人员快速了解云原生体系的全貌。CNCF Landscape 项目在 Github 上已经获得超过 7000 颗星，表明广大开发者和使用者对该项目的关注和重视。该项目给出当前云原生生态

的全景图（如下图所示），通过该全景图我们可以了解云原生相关的各种类型的项目和产品：

图 4：CNCF 生态全景图

这张全景图从云原生的层次结构，以及不同的功能组成上，展现了云原生体系的全貌，可以帮助用户在不同组件层次去选择恰当的软件和工具进行支持。总体来看，云原生生态分为以下几层：

Kubernetes 服务提供商

图中最底层是 Kubernetes 认证的服务提供商（包括公有云提供商 AWS、Google、Azure、Ali、Baidu 以及 Tencent 等，以及私有云提供商谐云、灵雀云、博云、才云、DaoCloud 以及 Rancher 等）以及 Kubernetes 认证的培训伙伴。

Provisioning

有了物理机或虚拟机后，在运行容器化服务之前，需要为容器准备标准化的基础环境，这就是 Provisioning 这一层的作用。在 Provisioning 这一层中，分为以下几个功能组成模块：

Automation & Configuration：用于自动化部署和配置容器运行平台和环境，代表工具和厂商包括 Ansible、Chef、Puppet、VMware 以及 OpenStack。 **容器镜像库**：容器镜像库是整个 CNCF 云原生中的核心部件之一，因为基于容器的运行环境中，所有的应用都需要借助容器镜像库来进行安装和部署。容器镜像库又分为公有和私有，公有的容器镜像库包括 docker 官方的 registry，AWS 的 Elastic Container Registry，Google 的 Container Registry 等。在私有镜像库中，VMware 中国团队主导的 Harbor 得到了广泛的应用，大量的容器平台目前都基于 Harbor 构建其镜像仓库。 **Security & Compliance**：Notary 和 TUF（The Upgrade Framework）是这个领域两个主要的项目，其中 TUF 是一个开源的安全标准，Notary 是其中一个实现。Notary 软件除了确保软件的出处外，它还能保证在未经容器镜像提供者批准的情况下，不会在镜像供应链的任何地方修改镜像内的内容，从而确保从开发到运营的过程中，安全都被无缝统一地嵌入到整个工作流中。 **Key Management**：主要用于在整个容器平台中进行秘钥管理。

Runtime

Runtime 这一层可以理解为容器的整个运行环境，是云原生中最核心的部分，它包括了计算、存储、网络三大块：

Container Runtime：Docker 是最广为人知的容器运行环境，但生产环境下也有一些其他的容器环境在运行。Containerd 是满足 OCI 规范的核心容器运行时，从设计上就是为了嵌入大型系统的。它由 Docker Inc 公司启动，并且在 2017 年 3 月份捐赠给了 CNCF。此外，CoreOS 的 RKT 是一个用于在 Linux 上运行应用程序容器的 CLI，也可以作为安全、可组合和基于标准的容器虚拟化运行环境。 **Cloud-Native Storage**：起初，容器为无状

态的运行单元，容器最上一层文件系统无法保存其在运行时写入的文件或数据，容器重建或重启后，这些写入的数据将丢失。但随着数据库、消息队列等中间件逐步在容器环境中得到应用，如今用户对容器持久化存储的理解和需求也更加深入和迫切。 **Cloud-Native Network**：网络历来是虚拟化技术中最灵活多变的部分，目前，大多数客户使用的主要包括 Calico、Flannel、Open vSwitch 等方案。

Orchestration Management

这一层主要负责容器平台的编排和调度，包括服务的发现和治理，远程调用，服务代理，微服务治理等组件，包括：

计划与编排：在这个领域，Kubernetes 是当仁不让的头号玩家，目前基于 Kubernetes 的容器生态得到了迅速发展。其它的编排工具还包括 Mesos 和 Docker Swarm 等。 **协调与服务发现**：分布式计算中很重要的一点就是各个服务之间的协同以及服务发现（或节点发现的问题），从老牌的 Zookeeper 到近年来在很多互联网厂商和应用中流行的 Consul（Docker Swarm 默认使用），都可以用于分布式服务的发现和配置，Kubernetes 默认使用的则是 CoreOS 旗下的 Etcd。 **远程过程调用**：微服务间进行通信，通常有两种方式，一种是 HTTP REST-JSON 的方式，另一种为 RPC 方式，两者相比，RPC 方式效率更高。常用的包括 Google 开源的 GRPC 、apache 旗下的 thrift 框架、Netflix 开源的自带负载均衡的 ribbon 和 avra 数据序列化框架。

App Definition and Development

这一层就是容器平台上运行的具体应用和工具了，可以理解为容器平台的应用商店。根据应用的不同作用的使用场景，可以大致分为以下几种类型：数据库（例如 MySQL、MariaDB、mongoDB、PostgreSQL、Cassandra、TiDB 等）、流处理和消息队列（例如 Spark、Storm、RocketMQ、Kafka、RabbitMQ 等）、应用和镜像制作（用于将应用封装成标准镜像，使应用能在标准的容器平台上运行，例如 Helm、Docker Composer、Packer 等）、CI/CD（最常见的 Jenkins、Atlassian 公司开发的 Bamboo 等）。

Platform

从横向上看，整个云原生还包括众多的经过认证的平台供应商。

Observability and Analysis

这个部分包含了大量用于对平台进行监控（Prometheus、Nagios、Grafana、Zabbix 等）、日志（Fluentd、ElasticSearch、Logstash）、以及追踪（Jaeger）的工具。

综上所述，CNCF Landscape 全景图中包含了 CNCF 社区成熟或使用范围较广、具有最佳实践的产品和方案供用户在实际应用中选择，而且该生态还在快速发展中。

6. 云原生实施路径

关于如何实现云原生应用，CNCF 给出了比较详细的利用开源软件和云原生技术的参考路线图，如下图所示：

图 5：CNCF 云原生技术路线图

整个路线图分成了十个步骤，每个步骤都是用户或平台开发者将云原生技术在实际环境中落地时，需要循序渐进思考和处理的问题：

容器化

目前最流行的容器化技术是 Docker，你可以将任意大小的应用程序和依赖项，甚至在模拟器上运行的一些程序，都进行容器化。随着时间的推移，你还可以对应用程序进行分割，并将未来的功能编写为微服务。

CI/CD（持续集成和持续发布）

创建 CI/CD 环境，从而使源代码上的任意修改，都能够自动通过容器进行编译和测试，并被部署到预生产甚至生产环境中。

应用编排

Kubernetes 是目前市场上应用编排领域被最广泛应用的工具，Helm Charts 可以用来帮助应用开发和发布者用于升级 Kubernetes 上运行的应用。

监控和分析

在这一步中，用户需要为平台选择监控、日志以及跟踪的相关工具，例如将 Prometheus 用于监控、Fluentd 用于日志、Jaeger 用于整个应用调用链的跟踪。

服务代理、发现和治理

CoreDNS、Envoy 和 LInkerd 可以分别用于服务发现和服务治理，提供服务的健康检查、请求路由、和负载均衡等功能。

网络

Calico、Flannel 以及 Weave Net 等软件用于提供更灵活的网络功能。

分布式数据库和存储

分布式数据库可以提供更好的弹性和伸缩性能，但同时需要专业的容器存储予以支持。

流和消息处理

当应用需要比 JSON-REST 这个模式更高的性能时，可以考虑使用 gRPC 或者 NATS。gRPC 是一个通用的 RPC（远程调用）框架（类似各种框架中的 RPC 调用），NATS 是一个发布/订阅和负载均衡的消息队列系统。

容器镜像库和运行环境

Harbor 是目前最受欢迎的容器镜像库，同时，你也可以选择使用不同的容器运行环境用于运行容器程序。

软件发布

最后可以借助 Notary 等软件用于软件的安全发布。

7. 参考资料：

1、CNCF 云原生定义 1.0，github.com/cncf/toc/blob/main/DEFINITION.md 2、迁移到云原生应用架构，Matt Stine 著，Jimmy Song 译，jimmysong.io/migrating-to-cloud-native-application-architectures 3、云原生全景图，landscape.cncf.io 4、CNCN landscape 项目，github.com/cncf/landscape 5、持续演进的 Cloud Native -云原生架构下微服务最佳实践，王启军，电子工业出版社 6、畅谈云原生，敖小剑，skyao.io/talk/201902-cloudnative-freely-talk 7、Service Mesh 发展趋势：云原生中流砥柱，敖小剑，skyao.io/talk/201905-servicemesh-development-trend 8、不可变基础设施，cloud.tencent.com/developer/news/329406 9、云原生技术详解，www.cnblogs.com/sddai/p/13726935.html 10、什么是服务网格，jimmysong.io/istio-handbook/concepts/what-is-service-mesh.html 11、什么是服务网格，www.redhat.com/zh/topics/microservices/what-is-a-service-mesh 12、运维的思维升级-不可变基础设施，wangkai1994.github.io/2019/01/12/Immutable-Infrastructure 杨文宏/中孚信息（北京）研究院