**Kubernetes**

可移植，可扩展，开源的平台，用于管理容器化的工作和服务，便于声明配置和自动化。具有大型快速的成长生态，其服务，支持和工具大范围可用。

来源于希腊，意味着舵手或飞行员。Google于2014年开源该项目，是建立在google大规模运行负载的15年经验上，结合了社区中的最佳创建和实践。

**部署的演变**

**传统部署时代**：早期的应用是运行在物理服务器上，没有途径来定义资源边界，会导致资源分配问题。假设多个应用在一台物理机上运行，会有一个应用占用大部分资源，而其他的应用则无法得到正常的表现。这类场景的一种解决方案是让每个应用独立运行于一个物理机。但由于资源未充分利用，也没有扩大规模，且也是十分昂贵的。

**虚拟部署时代**：虚拟化作为一种解决方案出现了，允许多个虚拟机运行在单个物理机CPU上。虚拟化允许应用在虚拟机之间隔离，并提供信息无法自由访问的安全级别。

虚拟化提供了更好的资源利用和扩展性，应用能轻松的添加或升级，降低硬件消耗等好处。

每个虚拟机是一个完整的机器并运行所有组件，包括在虚拟硬件上的操作系统。

**容器部署时代**：容器与VM相似，但有更宽松的隔离属性，应用间共享操作系统。因此，容器更加轻量。但容器依然有自己的文件系统，CPU，内存，处理空间等等。因为其从底层设施解耦，可以移植到云或其他操作系统上。

容器之所以流行是因为有多个好处，部分如下：

* 敏捷应用的创建与部署：相比较于VM镜像，容器镜像更加简单效率。
* 持续性开发，集成，部署：提供可靠和频繁的容器镜像构建和部署，快速且简易回滚（基于镜像不变性）。
* Dev与Ops分离：创建容器镜像要优于部署的时间，能从底层设备解耦。
* 不仅仅观察到操作系统级信息和指标，同时可以观察应用健康和其他信号。
* 环境一致性贯穿开发、测试和生产：单机和云上的表现一致。
* 云和OS分发的可移植性：任意的平台均可运行。
* 应用中心管理：从虚拟硬件到逻辑资源，提高了抽象级别。
* 松散耦合，分布式，弹性，释放的微服务：应用被分解为更小的，独立的组件，可以被动态部署和管理。
* 资源隔离：可预测的应用表现
* 资源利用：高效且紧密。

**为什么使用Kubernetes**

容器是很好的捆绑和运行应用的方式，在生产环境，你需要管理容器让应用运行并确保没有宕机，比如一个容器失败了，会需要启动另一个容器。若让一个系统来处理这个行为是不是会更容易？

这就是Kubernetes做的事，提供一个框架能使分布式系统能弹性运行。满足你扩展性要求，故障转移，开发模式等等。

* 服务发现和负载均衡：Kubernetes可以通过DNS名称或各自的IP来暴露容器；如果容器访问过高，Kubernetes能平衡负载并分发来保证部署稳定。
* 存储编排：Kubernetes允许你自动安装你选择的存储系统，比如本地存储，云服务等。
* 自动部署与回滚：可以描述一个理想状态，Kubernetes可以将实际状态以一个可控的速率向理想状态转变。比如让Kubernetes创建新的容器，将存在的容器移除并将它们的资源分给新容器。
* 自动装箱：Kubernetes允许你指定每个容器需要的CPU和内存，当容器指定了资源，Kubernetes能更好的管理容器资源。
* 自我修复：Kubernetes重启失败的容器，替换/杀死不回应健康检查的容器，且在完成前不会通知客户端。
* 密钥和配置管理：Kubernetes允许你存储和管理敏感信息，比如密码，令牌，和ssh key。你可以部署和更新密钥及应用配置而无需重建你的容器，也不会在你的配置栈中暴露密钥。

**Kubernetes有些事不做**

Kubernetes不是独立的，许多解决方案称为了选项和插件形式的。

* **不会限制应用类型**：Kubernetes目标是支持各种工作负载，包括无状态，有状态，和数据处理工作负载。如果应用能在容器中很好的运行，那在Kubernetes中也可以。
* **不部署源码，不构建应用**：持续性的集成，传递和部署工作流是有组织文化、偏好和技术要求来决定的。
* **不提供应用级的服务**，比如中间件（消息总线），数据处理框架（Spark），数据库（Mysql），缓存，也没有内置的存储系统服务（Ceph）。许多组件能在Kubernetes上运行，或通过移植机制（Open Service Broker）可以访问应用。
* **不会指挥日志，监督，或警告**：提供集成或机制来收集和导出指标。
* **不提供也不要求配置语言或系统**：提供一个声明性的API，可通过任意形式的声明规范来实现。
* **不提供也不适应任何机器配置，维护，管理，或自我修复系统**。

此外，Kubernetes不仅仅是编排系统。事实上，它消除了编排的需要。编排的技术定义是工作流的执行。相比之下，Kubernetes由一组独立的，可组合的控制过程组成，这些过程不断地将当前状态驱动到所提供的期望状态。 如何从A到C无关紧要。集中控制也不是必需的。 这使得系统更易于使用，功能更强大，更强大，更具弹性，并且可扩展