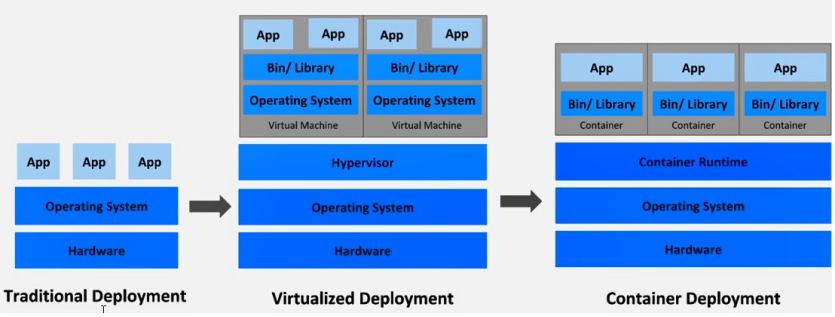
# kubernetes 工作原理

## 一．案例概述




### 传统部署时代

早期是在物理服务器上运行应用程序。无法为物理服务器中的应用程序定义资源边界，这会导致资源分配出现问题。例如：如果在物理服务器上运行多个应用程序，则可能会出现一个应用程序占用大部分资源的情况，结果可能会导致其他应用程序的性能下降。一种解决方案是在不同的物理服务器上运行每个应用程序，但是由于资源利用不足而无法扩展，并且组织维护许多物理服务器的成本很高。↵

### 虚拟化时代

虚拟化功能允许在单个物理服务器 CPU 上运行多个虚拟机（VM）。虚拟化功能允许应用程序在 VM 之间隔离，并提供安全级别，因为一个应用程序的信息不能被另一应用程序自由地访问。↵

因为虚拟化可以轻松地添加或更新应用程序、降低硬件成本等等，所以虚拟化可以更好地利用物理服务器中的资源，并可以实现更好的可伸缩性。每个 VM 是一台完整的计算机，在虚拟化硬件之上运行所有应用组件，包括其自己的操作系统。↵

### 容器部署时代

容器类似于 VM，但是它们具有轻量级的隔离属性，可以在应用程序之间共享操作系统（OS）。因此，容器被认为是轻量级的。容器与 VM 类似，具有自己的文件系统、CPU、内存、进程空间等。由于它们与基础架构分离，因此可以跨云和 OS 分发进行移植。↵

↵

随着 Docker 技术的发展和广泛流行，云原生应用和容器调度管理系统成为 IT 领域大热的话题。在 Docker 技术火爆之前，云计算技术领导者与分布式系统架构推广者就已经开始提出并广泛传播云原生应用的思想。2011 年 Heroku 工程师提出云原生应用的 12 要素，只不过当时是以虚拟机技术作为云原生应用的主流技术来实施的。由于虚拟机镜像大、镜像标准、打包流程和工具不统一，导致业界无法广泛接受，限制了云原生应用的发展。而 Docker 技术的出现正好解决了云原生应用构建、交付和运行的瓶颈，使得构建云原生应用成为使用 Docker 的开发者的优先选择。那么 Docker 从单机走向集群已经成为必然趋势。Kubernetes 作为当前唯一一个被广泛认可的 Docker 分布式解决方案，在未来几年内，会有大量的系统选择它。↵

## 二．Kubernetes 概述

Kubernetes 是由 Google 在 2014 年 6 月开源的容器编排调度引擎，使用 Go 语言开进行发，最初源于谷歌内部.Brog,引擎。由于 Kubernetes 的 K 和 s 间有 8 个字母，因此国内行业人员简称为 K8S。2015 年 7 月 Kubernetes V1.0 正式发布，截止到目前最新稳定版本是 V1.18.x。Kubernetes 拥有一个庞大且快速增长的生态系统。

Kubernetes 这个名字，起源于古希腊，是舵手的意思，所以它的 logo 即像一张渔网又像一个罗盘，谷歌选择这个名字还有一个深意:既然 docker 把自己比作一只鲸鱼，装箱，在大海上遨游，google 就要用 Kubernetes 去掌握大航海时代的话语权，去捕获和指引着这条鲸鱼按照主人设定的路线去巡游。

Kubernetes 是一个可移植、可扩展的开源 Docker 容器编排调度引擎，和 Docker 容器结合在一起，可实现容器技术的分布式架构方案。主要用于自动化部署、扩展和管理容器类应用，提供资源调度、部署管理、服务发现、扩容缩容、监控等功能。它提供完善的管理工具，涵盖开发、部署测试、运维监控等各个环节。它的目标不仅仅是一个容器编排系统而是提供一个应用规范，用户可以描述集群的架构，定义服务的最终状态，Kubernetes 可以将系统达到和维持在这个状态。对于负载均衡、服务发现、高可用、滚动升级、自动伸缩等容器云平台的功能要求有原生支持。

随着对 k8s 系统架构与设计理念的深入了解，可以发现 k8s 系统正是为运行原生应用而设计考虑，使得基于 k8s 系统设计和开发生产级的复杂云原生应用，变得像启动一个单机版容器的服务这样简单易用。

Kubernetes 可以调度计算集群节点、动态管理节点上应用，并保证它们按用户期望状态运行。通过使用「Labels（标签)」和「Pods（豆荚)」的概念，Kubernetes 将应用按逻辑单元进行分组，方便管理和服务发现。

官网：https://www.kubernetes.io
文档：https://www.kubernetes.io/zh/docs/home
CNCF：云原生计算基金会（关注点）

## 2.1 使用 bukernetes 具备的好处

·具备微服务架构

微服务架构的核心是将一个巨大的单体应用分解为很多小的互相连接的微服务。一个微服务背后可能有多个实例副本支撑，副本的数量可能会根据系统负荷变化而进行调整（弹性收缩），而 K8S 平台中内嵌的负载均衡器发挥着重要作用。微服务架构使得每个服务都可以由专门的开发团队来开发，开发者可以自由选择开发技术，这对于大规模团队来说很有价值。另外，每个微服条独立开发、升级、扩展，使得系统具备很高的稳定性和快速迭代进化能力。

·具备超强的横向扩容能力

对于互联网公司来说,用户规模等价于资产,谁拥有更多的用户,谁就能在竞争中胜出，因此超强的横向扩容能力是互联网业务系统的关键指标之一。K8S 集群中可从只包含几个 Node 的小集群，平滑扩展到拥有成百上千 Node 的大规模集群，利用 K8S 提供的工具，甚至可以在线完成集群的扩容。只要微服务设计的合理,结合硬件或者公有云资源的线性增加，系统就能够承受大量用户并发访问所带来的压力。

## 2.2 kubernetes 服务功能

·数据卷

当 Pod 中容器之间想要共享数据时，可以使用数据卷

·应用程序监控检查

容器内服务可能进程阻塞无法处理请求，可以设置监控检查策略保证应用健壮性
·复制应用程序示例
控制器维护着 Pod 副本数量，保证一个 Pod 或一组同类的 Pod 数量始终可用。
·服务发现
使用环境变量或 DNS 服务插件保证容器中程序发现 Pod 入库访问地址
·弹性伸缩
根据指定的指标（CPU 利用率）自动缩放 Pod 副本数
·负载均衡
一组 Pod 副本分配一个私有的集群 IP 地址，负载均衡转发请求到后端容器。在集群内部其他 Pod 可以通过这个 cluster lP 访问应用。
·滚动更新
更新时，服务是不中断的，一次更新一个 Pod，而不是同时删除整个服务，类似于灰度发布
·服务编排
通过文件描述部署服务，是的应用程序部署变得更高效
·资源监控
node 节点集成 cAdvisor 资源收集工具，可通过 Heapster 汇总整个集群节点资源数据，然后存储到!nfluxDB 时序数据库，再由 Grafana 展示。
·提供认证和授权
支持角色访问控制（RBAC 基于角色的权限访问控制 Role-Based Access Control)认证授权等策略。

## 2.3 kubernetes 服务特点（重点）

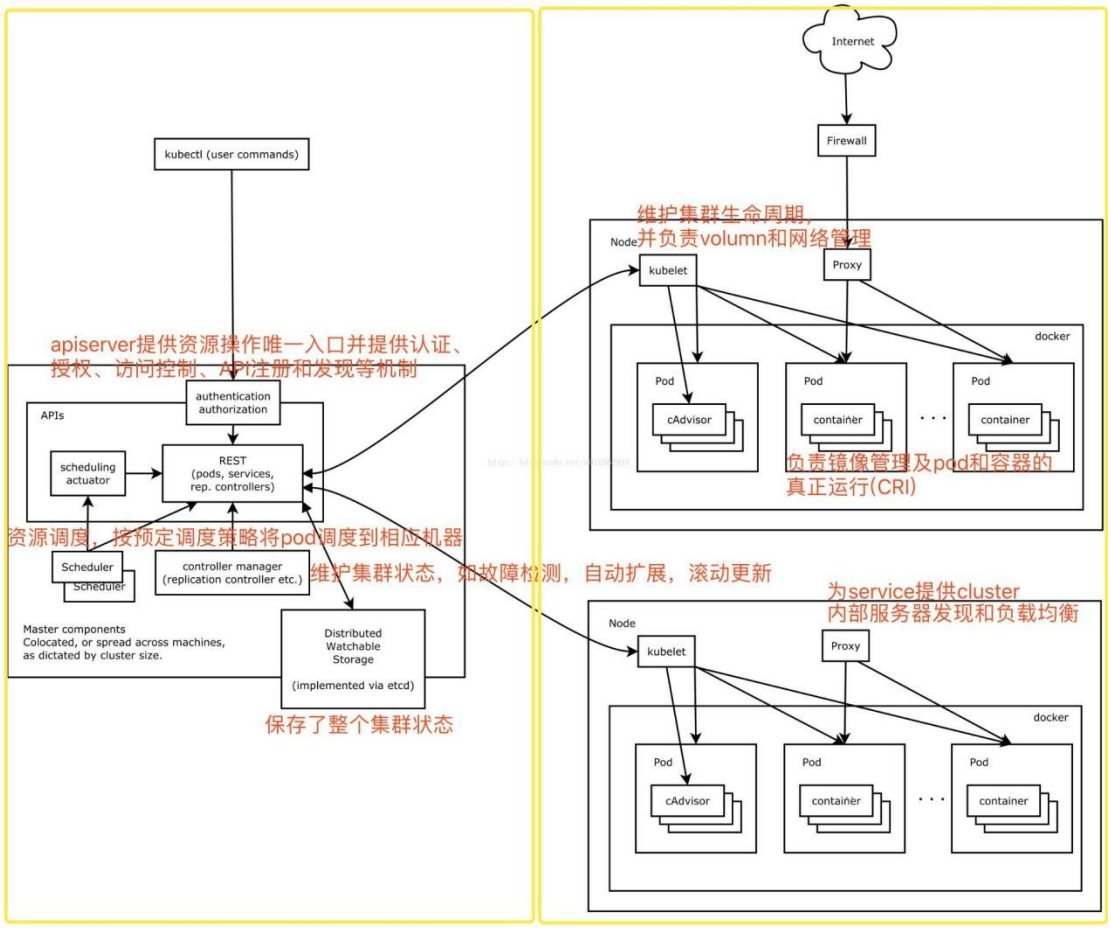
·服务发现和负载均衡
Kubernetes 可以使用 DNS 或自己的 IP 地址公开容器。如果容器的流量很大，kubernetes 可以负载均很分配网络流量，从而保证服务部署的稳定性
·储存编排
Kubernetes 允许您自动挂载您选择的存储系统。例如本地存储，公共云提供商等
·自动部署和回滚
Kubernetes 可以描述已部署容器的所需状态，可以以受控的速率将实际状态更改为所需状态。例如:您可以以自动化使 Kubernetes 来部署创建新容器，删除现有容器并将它们的所有资源用于新容器。
·自定二进制打包
Kubernetes 可以指定每个容器所需 CPU 和内存（RAM)。当容器进行资源请求时 Kubernetes 可以做出更好的决策来管理容器的资源。
·自我修复
Kubernetes 重新启动失败的容器、替换容器、杀死不响应用户定义的运行状况检查的容器，并且在准备好服务之前不将其通告给客户端。
·密钥于配置管理
Kubernetes 可以存储和管理敏感信息，例如密码、OAuth 令牌和 ssh 密钥。可以在不重建容器镜像的情况下部黑和更新密钥和应用程序配置，也无需在堆栈配置中暴露密钥。

## 2.4 kubernetes 应用场景

Kubernetes 常见应用场景
·快速部署应用
·快速扩展应用
·无缝对接新的应用功能
·节省资源，优化硬件资源的使用
Kubernetes 适应场景
·可移植：支持公有云，私有云，混合云，多重云
·可扩展：模块化，插件化，可挂载，可组合
·自动化：自动部署，自动重启，自动复制，自动伸缩/扩展

## 2.5 kubernetes 架构（重点）

重点图：




## 2.6 kubernetes 架构节点

### （1）master 节点

Maste 节点提供集群的管理控制中心。对集群进行全局决策，并检测和响应集群事件（例如：随时监控复制控制器（RC）“副本”字段不满足时启动新的 Pod（容器））。基本上 k8s 所有的控制命令都是发个 master，master 负责具体的执行过程。Master 可以在集群的任何一台服务器上运行，但是建议最好单独使用一台服务器并做好高可用。因为 master 作为 k8s 集群的大脑，如果出现宕机，那么集群将控制中心将会失效

在 master 节点上运行的关键进程

·kube-APIserver：用于暴露 kubernetes API 接口，任何的资源请求/调用操作都是通过 kube-APIserver 提供的接口进行的，提供 HTTP rest 接口的关键服务进程，是实现，增，删，查，改等操作的唯一入口，也是集群控制入口的进程

·Etcd：是 kubernetes 提供的默认存储，所有集群的数据都保存在 Etcd 中，使用时建议为 Etcd 数据提供备份计划

·kube-scheduler：是负责资源调度的进程，监视新创建且没有分配到 Node 的 Pod，为 Pod 选择一个 Node

·kube-controller-manager：运行管理控制器，集群中处理常规任务的后台进程，是 kubernetes 里所有的资源的自动化控制中心。

这些控制器主要包括：

节点控制器（Node Controller）：

负责在 Node 节点出现故障时及时发现和响应；

复制控制器：（Replication Controller）：

复制维护正确数量的 pod

端点控制器：（Endpoints Controller）：

填充端点对象（即 services 和 Pods）

服务账户和令牌控制器：（service Account & Tokeb Controllers）：为新的命名空间创建默认账户和 API 访问令牌

**（2）Node 节点**

Kubernetes 集群除了 master 节点，其他服务器都可以称为 node 节点，与 master 节点一样，node 节点可以是一台物理机，也可以是一台虚拟机。Node 节点时 k8s 集群中的工作负载节点，每个 node 都会被 master 分配一些工作负载，当某个 node 宕机时，node 上的工作负载会被 master 自动转移到其他的 node 节点上去

每个 node 节点上都运行的以下关键进程

·kubelet：负责 pod 对应容器的创建，启停等任务，同时与 master 节点密切协作，实现集群管理的基本功能

·kube-proxy：用于实现 kubernetes service 之间的通信与负载均很机制

·Docker enging(docker)：Docker 引擎负责本机的容器运行和管理工作

注意：node 节点下走的就是 docker 服务

Node 节点可以在运行期间动态增加到 Kubernetes 集群中，前提是这个节点上已经正确安装、配置和启动了上述关键进程。在默认情况下，Kubelet 会向 Master 注册自己，这也是 Kubernetes 推荐的 Node 管理方式。一旦 Node 被纳入集群管理范围，Kubelet 进程会定时向 Master 汇报自身的情况，例如操作系统、Docker 版本、机器的 CPU 和内存情况，以及之前有哪些 Pod 在运行等。这样 Master 可以获知每个 Node 的资源使用情况，并实现高效负载均衡资源调度策略。而某一个 Node 超过指定时间不上报信息时，会被 Master 判定为失联的状态，被标记为丕可用，随后 Master 会触发节点转移进程。

### （3）插件（Addons）

插件使用 kubernetes 资源（daemonset,Deployment）等实现集群功能，因为这些提供集群级别的功能，所以插件的命名空间资源属于 kube-system 命名空间

·DNS：尽管并非严格要求其他附加组件，但所有示例都依赖集群 DNS,因此所有

Kubernetes 集群都应具有 DNS，除了环境中的其他 DNS 服务器之外，集群 DNS 还是一个 DNS 服务器，它为 Kubernetes 服务提供 DNS 记录, Cluster DNS 是一个 DNS 服务器，和部署环境中的其他 DNS 服务器一起工作,为 Kubernetes 服务提供 DN 记录，Kubernetes 启动的容器自动将 DNS 服务包含在 DNS 搜索中。

·Dashboard：是 kubernetes 集群通用基于 Web 的 UI。它使用户可以管理集

群中运行的应用程序以及集群本身并进行故障排除。

容器资源控制将关于容器的一些常见的时间序列量值保存到一个集中的数据库中，并提供用于浏览这些数据的界面

集群层面日志机制负责将容器的日志数据保存到一个集中的日志存储中，该存储能够提供搜索和浏览接口。

## 2.7 kunernetes 资源对象

Kubernetes 包含多种类型的资源对象：Pod、Replication Controller、Service、Deployment、Job、DaemonSet.等。所有的资源对象都可以通过 Kubernetes 提供的 kubectl 工具进行增、删、改、查等操作，并将其保存在 Etcd 中持久化存储。从这个角度来看，Kubernets 其实是一个高度自动化的资源控制系统，通过跟踪对比 Etcd 存储里保存的资源期望状态与当前环境中的实际资源状态的差异,来实现自动控制和自动纠错等高级功能。下面对常用的资源对象分别进行介绍。

·Pod

Pod(豆荚)是 kubernetes 创建或部署的最小/最简单的基本单位，一个 Pod 代表集群上正在运行的一个进程，一个 pod 由一个或多个容器组成，pod 中的容器可以共享存储的网络，在同一台 docker 主机上运行，每个 pod 都有一个特殊特殊的被称为“根容器”的 pause 容器，pause 容器对应的镜像属于 kubernetes 平台的一部分。除了 pause 容器。每个 pod 还包含一个或多个紧密相关的用户业务容器

·label

label(标签)是 Kubernetes 系统中另外一个核心概念。一个 Label 是一个

key-value 的键值对（变量），其中 key 与 value 由用户自己指定。Label 可以附加到各种资源对象上，例如 node，service、RC 等。一个资源对象可以定义任意数量

的 label，同一个 label 也可以被添加到任意数量的资源对象中，也可以在对象创建后动态添加或者删除。

另外可以通过给指定的资源对象捆绑一个或多个不同的 Label，来实现多纬度的资源分组管理功能，以便于灵活、方便地进行资源分配、调度、部署等管理工作 label 标签选择源对象定义一个 Label，就相当于给它打了一个杯签，随后可以通过 label selector 标签选择器查询和筛选拥有某些 label 的资源对象，kubernetes 通过这种方式实现了类似于 SQL 的简单又通用的对象查询机制

·ReplicaSet（RC 的升级版）

Replication Controller(复制控制器，RC）是 Kubernetes 集群中最早的保证 Pod 高可用的 APIl 对象。通过监控运行中的 Pod 来保证集群中运行指定数目的 Pod 副本。指定的数目可以是 1 个或多个;如果少于指定数目，RC 就会运行新的 Pod 副本。如果多于指定数目，RC 就会杀死多余的 Pod 副本。即使在数目为 1 的情况下，通过 RC 运行 Pod 也比直接运行 Pod 更明智，因为 RC 可以发挥它高可用的能力，保证永远有 1 个 Pod 在运行。RC 是 K8s 较早期的技术概念,只适用于长期伺服型的业务类型。

RC 与 RS (ReplicaSet）唯一区别就是 label selector 支持不同，RS 支持新的基于集合的标签，RC 仅支持基于等式的标签。推荐使用 Rs，后面 RC 将可

能被淘汰。

·Deployment

Deployment(部署）表示用户对 K8s 集群的一次更新操作。部署是一个比 RS 应用模式更广的 API 对象，可以是创建一个新的服务，更新一个新的服务，也可以是滚动升级一个服务。滚动升级一个服务，实际是创建一个新的 RS，然后逐渐将新 RS 中副本数增加到理想状态，将旧 RS 中的副本数减小到 0 的复合操作;这样一个复合操作用一个 RS 是不太好描述的，需要用一个更通用的 Deployment 来描述。未来对所有长期服务型的的业务的管理，都会通过 Deployment 来管理。

·service

RC 和 Deployment 只是保证了支撑 Service（服务）的微服务 Pod 的数量，但是没有解决如何访问这些服务的问题。一个 Pod 只是一个运行服务的实例，随时可能在一个节点上停止，在另一个节点以一个新的 IP 地址启动一个新的 Pod，因此不能以固定的 IP 地址和端口号提供服务。要稳定地提供服务，需要服务发现和负载均衡能力。服务发现完成的工作，是针对客户端访问的服务，找到对应的的后端服务实例。

在 K8s 集群中，客户端需要访问的服务就是 Service 对象。每个 Service 会对应一个集群内部有效的虚拟 IP，集群内部通过虚拟 IP 访问一个服务。在 K8s 集群中微服务的负载均衡是由 Kube-proxy 实现的。Kube-proxy 是 K8s 集群内部的负载均衡器。它是一个分布式代理服务器，在 K8s 的每个 Node 节点上都会运行一个 Kube-proxy 组件;这一设计体现了它的伸缩性优势，需要访问服务的节点越多，提供负载均衡能力的 Kube-proxy 就越多，高可用节点也

随之增多。与之相比，通过在服务器端部置反向代理做负载均衡，还需要进一步解决反向代理的负载均衡和高可用问题。

（重点结束）

·job

Job 是 Kubernetes 用来控制批处理型任务的 API 对象。批处理业务与长期服务业务的主要区别是批处理业务的运行有头有尾，而长期伺服业务在用尸个停 l 止 L 的情优下水想 n 根据用户的设置，Job 管理的 Pod 把任务成功完成就自动退出了。成功完成的标志根据个同的 spe.c.campletions.策略而不同:单 Pod 型任务有一个 Pod 成功就标志完成;定数成功型任务保证有 N 太任务全部成功;工作队列型任务根据应用确认的全局成功而标志成功。

·DaemonSet

后台支撑型服务的核心关注点在 Kubernetes 集群中的节点（物理机或虚拟机)，DaemonSet （守护程序集）确保所有或某些节点运行同一个 Pod，要保证每个节点上都有一个此类 Pod 运行。节点可能是所有集群节点也可能是通过 nodeSelector 选定的一些特定节点。典型的后台支撑型服务包括，存储，日志和监控等在每个节点上支持，K8s 集群运行的服务

·Volume

数据卷，共享 pod 中容器使用的数据

·Namespace

命名空间将对象逻辑上分配到不同 namespace，可以是不同的项目、用户等区分管理，并设定控制策略，从而实现多租户。命名空间也成为虚拟集群。

·StatefulSet

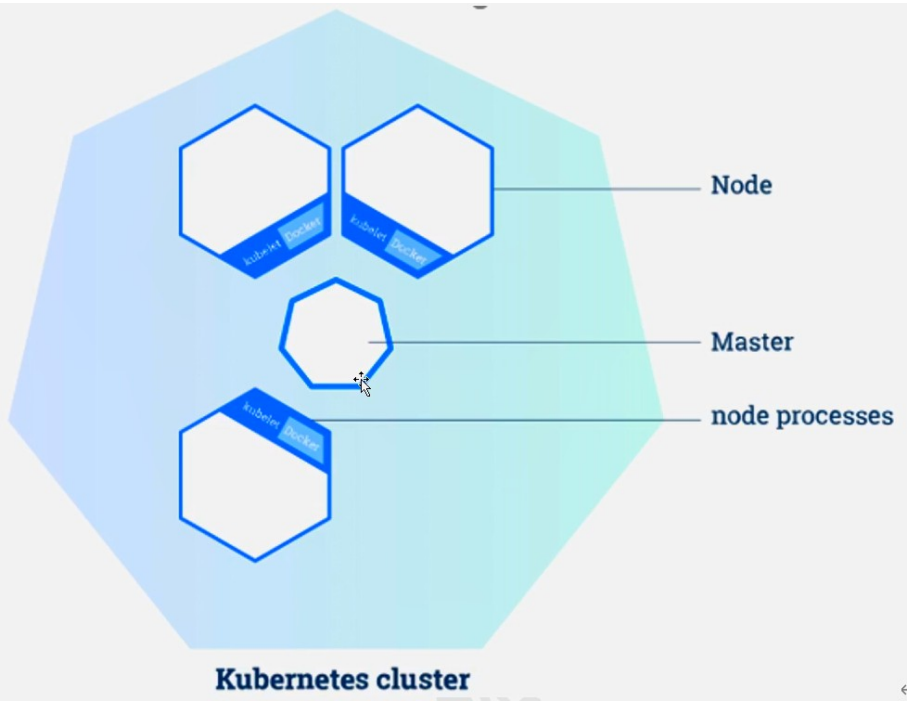
StatefulSet 适合持久性的应用程序，有唯一的网络标志符(IP），持久存储，有序的部署、扩展、删除和滚动更新。

## 2.8 kubernetes 原理解图

集群有两种角色：master 和 node

·master 是集群的“大脑”，负责管理整个集群：像应用的调度，更新，扩容。收缩等

·node 就是具体“干活”的，一个 node,一般是一个虚拟机或者是一台物理机，它上面事先运行着 docekr 和 kubelet 服务（kubernetes 的一个组件，master 派来用于响应任务的小弟），当收到 master 节点下发的任务时，node 就要去完成任务（用 docker 运行一个指定的应用）




Deployment -应用管理者

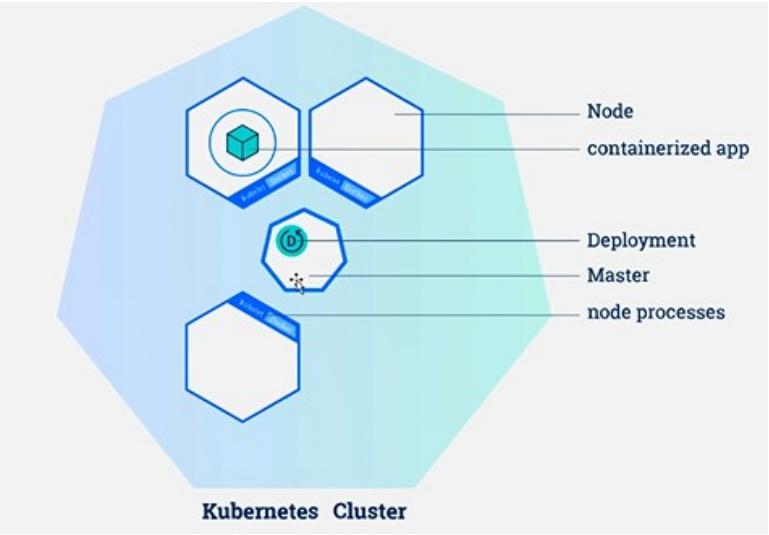
当我们拥有一个 Kubernetes 集群后，就可以在上面跑我们的应用了，前提是我们的应用必须支持在 docker 中运行，也就是我们要事先准备好 docker 镜像。

有了镜像之后，一般我们会通过 Deployment 配置文件去描述应用，比如应用叫什么名字、使用的镜像名字、要运行几个实例、需要多少的内存资源、cpu 资源等等。

有了配置文件就可以通过 Kubernetes 提供的命令行客户端 kubectl 去管理这个应用了。kubectl 会跟 Kubernetes 的 master 先通过验证及授权连接上 API 在和 RestAPI,通信，最终完成应用的管理。

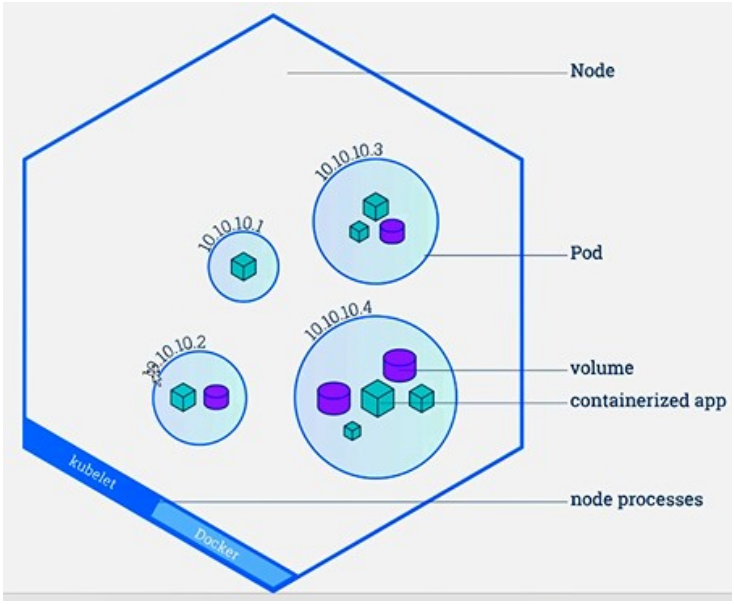
比如我们刚才配置好的 Deployment 配置文件叫 app.xaml，我们就可以通过"kubectl create -

f app.vaml”来创建这个应用，之后就由 Kubernetes 来保证我们的应用处于运行状态,当某个实例运行失败了或者运行着应用的 Node 突然宕机了，Kubernetes 会自动发现并在新的 Node 上调度出一个新的实例，保证我们的应用始终达到我们预期的结果。




Pod-kubernetes 最小调度单位

其实在上一步创建完 Deployment 之后，kubernetes 的 node 做的事情并不简单的 docekr run 一个容器。出于易用性，灵活性，稳定性等考虑，kubernetes 提出一个叫作 pod 的东西。作为 kubernetes 的最小调度单位。所以我们的应用在每个 node 上运行的其实就是一个 pod。Pod 也能在 node 上运行。如下图：

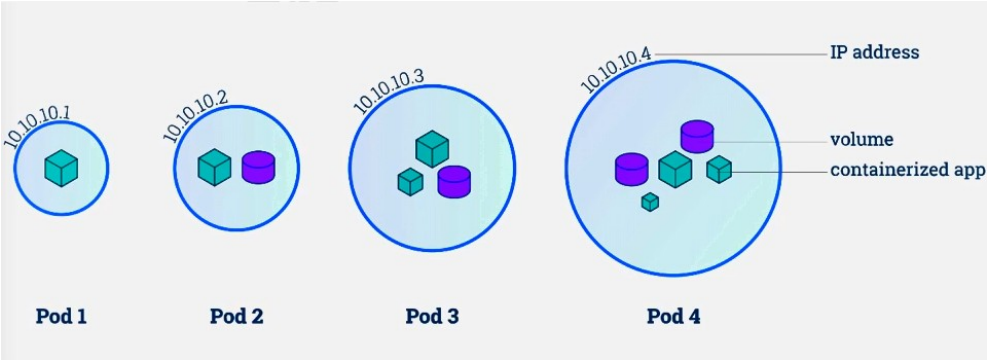



· 那么什么是 Pod 呢? Pod 是一组容器当然也可以只有一个)。容器本身就是一个小盒子了，Pod 相当于在容器上又包了一层小盒子。这个盒子里面的容器有什么特点呢?

·可以直接通过 volume 共享存储。

·有相同的网络空间，通俗点说就是有一样的 ip 地址，有一样的网卡和网络设置。

·多个容器之间可以“了解”对方，比如知道其他人的镜像，知道别人定义的端口等。

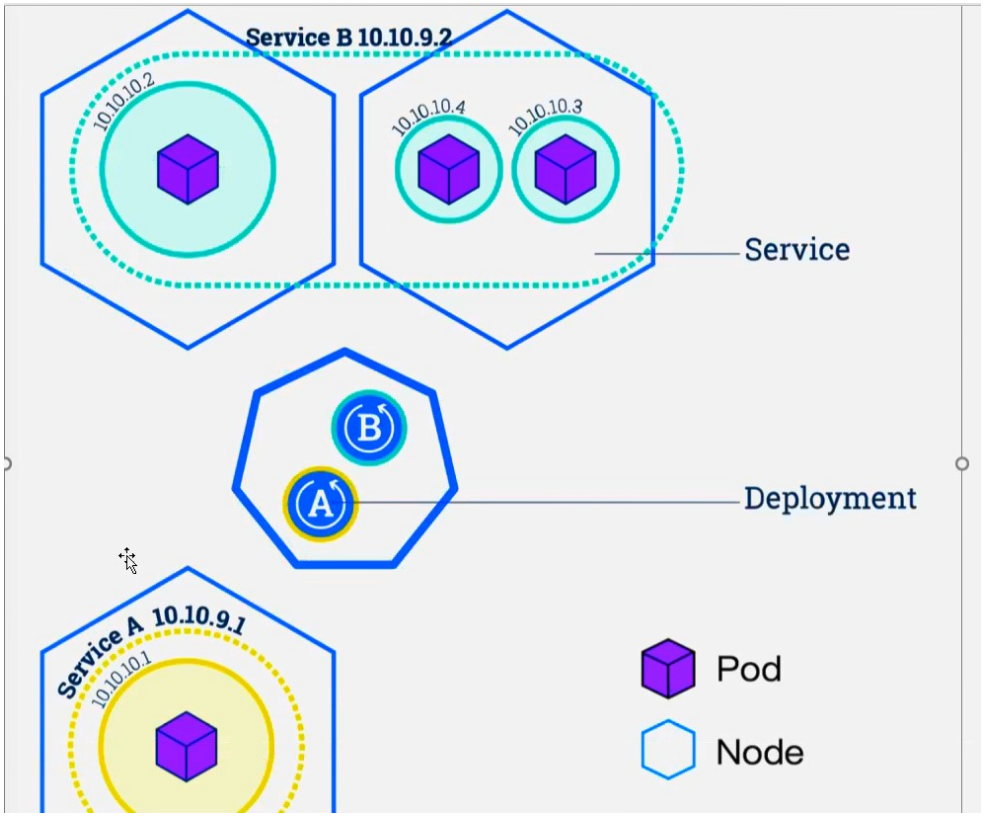



Service -服务发现 -找到每个 pod

上面的 Deployment 创建了，Pod 也运行起来了。如何才能访问到我们的应用呢?-最直接想到的方法就是直接通过 Pod.intport 去访问,但如果实例数很多呢?先拿到所有的 Pod-ip。列表，再配置到负载均衡器中，轮询访问。但上面我们说过，Pod 可能会死掉，甚至 Pod 所在的 Node 也可能煮机，Kubernetes 会自动帮我们重新创建新的 Pod。再者每次更新服务的时候也会重建 Pod。而每个 Pod 都有自己的 ip。所以 Pod 的 ip 是不稳定的，会经常变化的。

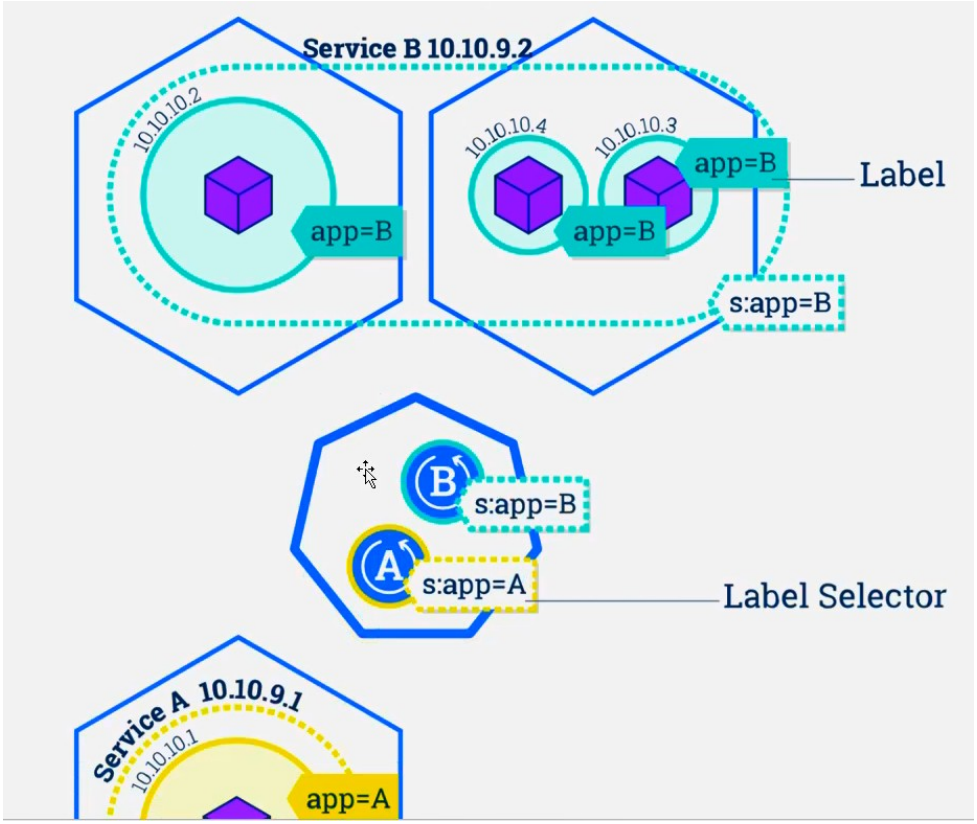
面对这种变化我们就要借助另一个概念:Service。它就是来专门解决这个问题的。不管 Deployment 的 Pod 有多少个，不管它是更新、销毁还是重建，Service 总是能发现并维护好它的 ip 列表。Service 对外也提供了多种入口:

1. ClusterIP：service 在集群内的唯一 IP 地址，我们可以通过这个 IP 地址访问到后端的 pod，而无需关心具体的 pod
2. Nodebalancer：在 nodeport 的基础上，借助公有云环境创建一个外都的负载均衡器，并将请求转发到 nodeIP：nodeport
3. Nodeport：service 会在每个 node 上都启动一个端口。我们可以通过任意 node 的端口来访问到 node
4. ExternaLName：将服务通过 DNS CNAME 记录的方式转发到指定的域名（通过 sepc.externLName 设定）



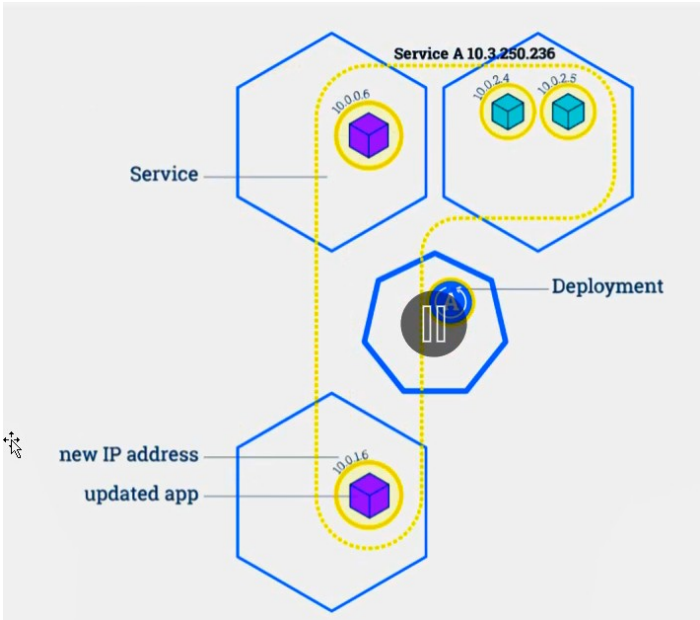

看似服务访问的问题解决了。但大家有没有想过, service 是如何知道它负责哪些 Pod 呢?又是如何跟踪这些 Pod 变化的?

最容易想到的方法是使用 Deployment 的名字。一个 Service 对应一个 Deployment o 当然这样确实可以实现。但 kubernetes, 使用了一个更加灵活、通用的设计 Label 标签，通过给 Pod 打标签，Service 可以只负责一个 Deployment 的 Pod 也可以负责多个 Deployment 的 Pod 了。Deployment 和 Service 就可以通过 Label 解耦了。




## RollingUpdate-滚动升级

滚动升级是 Kubernetes 中最典型的服务升级方案，主要思路是一边增加新版本应用的实例数，一边减少旧版本应用的实例数，直到新版本的实例数达到预期，旧版本的实例数减少为 0，滚动升级结束。在整个升级过程中，服务一直处于可用状态。并且可以在任意时刻回滚到旧版本。




管理员 --->kubectl -->(nginx.yaml) -->label -->Pod -->container