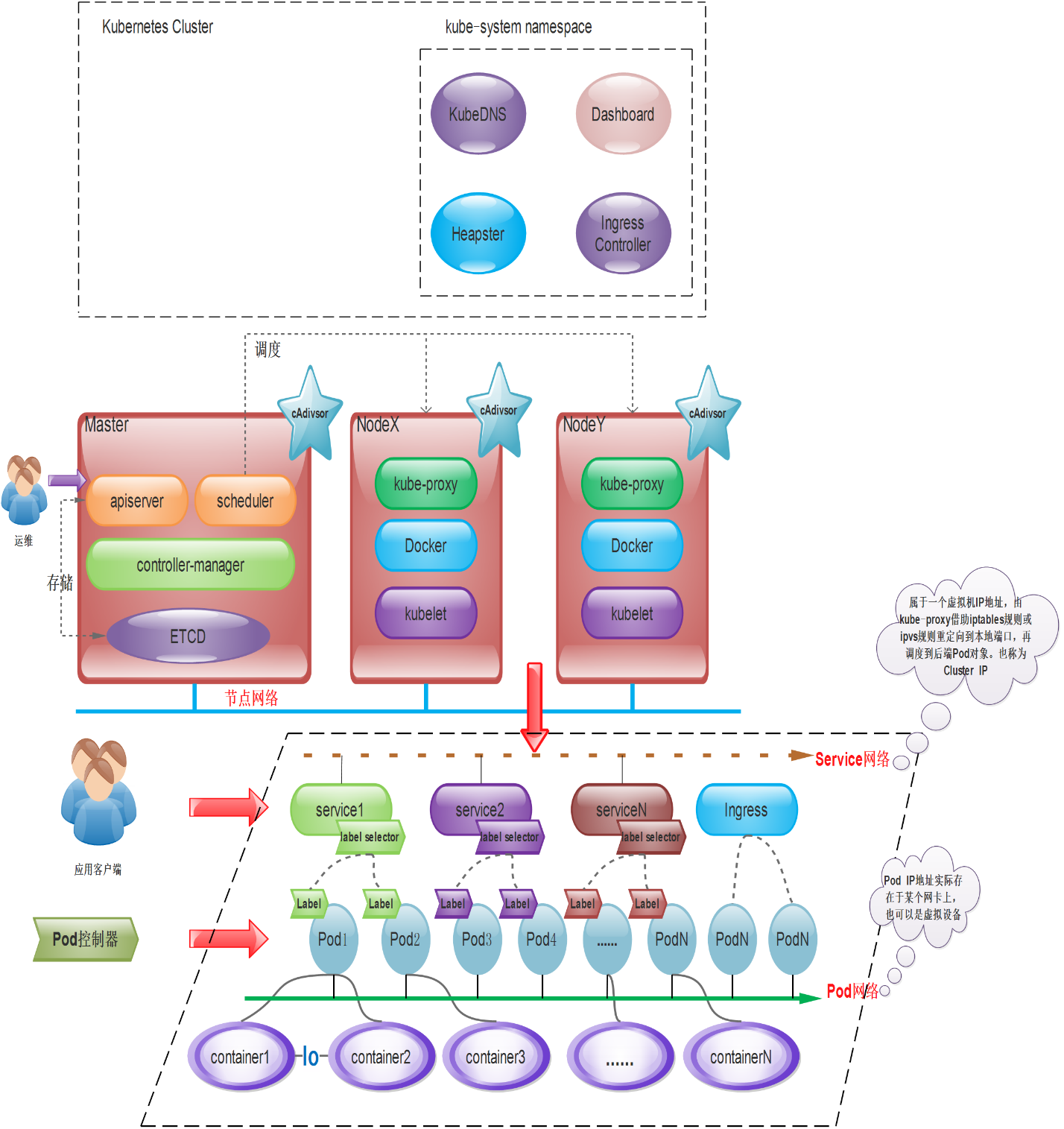
# [Kubernetes学习之路（二十）之K8S组件运行原理详解总结](https://www.cnblogs.com/linuxk/p/10291178.html)

# 一、看图说K8S

先从一张大图来观看一下K8S是如何运作的，再具体去细化K8S的概念、组件以及网络模型。



从上图，我们可以看到K8S组件和逻辑及其复杂，但是这并不可怕，我们从宏观上先了解K8S是怎么用的，再进行庖丁解牛。从上图我们可以看出：

* Kubernetes集群主要由Master和Node两类节点组成
* Master的组件包括：apiserver、controller-manager、scheduler和etcd等几个组件，其中apiserver是整个集群的网关。
* Node主要由kubelet、kube-proxy、docker引擎等组件组成。kubelet是K8S集群的工作与节点上的代理组件。
* 一个完整的K8S集群，还包括CoreDNS、Prometheus（或HeapSter）、Dashboard、Ingress Controller等几个附加组件。其中cAdivsor组件作用于各个节点（master和node节点）之上，用于收集及收集容器及节点的CPU、内存以及磁盘资源的利用率指标数据，这些统计数据由Heapster聚合后，可以通过apiserver访问。

**要了解K8S的所有组件，没去走一遍，永远不知道它是怎么跑起来的，那么下面就带着几个新手疑问来看K8S**

* **1、K8S是如何对容器编排？**

在K8S集群中，容器并非最小的单位，K8S集群中最小的调度单位是Pod，容器则被封装在Pod之中。由此可知，一个容器或多个容器可以同属于在一个Pod之中。

* **2、Pod是怎么创建出来的？**

Pod并不是无缘无故跑出来的，它是一个抽象的l逻辑概念，那么Pod是如何创建的呢？Pod是由Pod控制器进行管理控制，其代表性的Pod控制器有Deployment、StatefulSet等。这里我们先有这样的一个概念，后面再详细解刨。

* **3、Pod资源组成的应用如何提供外部访问的？**

Pod组成的应用是通过Service这类抽象资源提供内部和外部访问的，但是service的外部访问需要端口的映射，带来的是端口映射的麻烦和操作的繁琐。为此还有一种提供外部访问的资源叫做Ingress。

* **4、Service又是怎么关联到Pod呢？**

在上面说的Pod是由Pod控制器进行管理控制，对Pod资源对象的期望状态进行自动管理。而在Pod控制器是通过一个YAML的文件进行定义Pod资源对象的。在该文件中，还会对Pod资源对象进行打标签，用于Pod的辨识，而Servcie就是通过标签选择器，关联至同一标签类型的Pod资源对象。这样就实现了从service-->pod-->container的一个过程。

* **5、Pod的怎么创建逻辑流程是怎样的？**
* （1）客户端提交创建请求，可以通过API Server的Restful API，也可以使用kubectl命令行工具。支持的数据类型包括JSON和YAML。
* （2）API Server处理用户请求，存储Pod数据到etcd。
* （3）调度器通过API Server查看未绑定的Pod。尝试为Pod分配主机。
* （4）过滤主机 (调度预选)：调度器用一组规则过滤掉不符合要求的主机。比如Pod指定了所需要的资源量，那么可用资源比Pod需要的资源量少的主机会被过滤掉。
* （5）主机打分(调度优选)：对第一步筛选出的符合要求的主机进行打分，在主机打分阶段，调度器会考虑一些整体优化策略，比如把容一个Replication Controller的副本分布到不同的主机上，使用最低负载的主机等。
* （6）选择主机：选择打分最高的主机，进行binding操作，结果存储到etcd中。
* （7）kubelet根据调度结果执行Pod创建操作： 绑定成功后，scheduler会调用APIServer的API在etcd中创建一个boundpod对象，描述在一个工作节点上绑定运行的所有pod信息。运行在每个工作节点上的kubelet也会定期与etcd同步boundpod信息，一旦发现应该在该工作节点上运行的boundpod对象没有更新，则调用Docker API创建并启动pod内的容器。

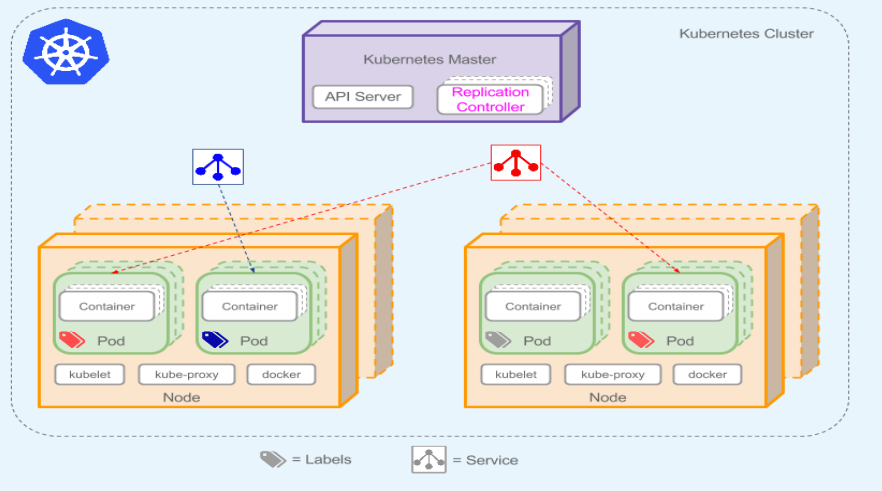
**从上面的几个疑问，大致了解了K8S怎么工作的，那么现在再从三个面去了解Kubernetes，分别是Kubernetes概念和术语、集群组件、网络模型。**

# 二、K8S的概念和术语

​ Kubernetes是利用共享网络将多个物理机或者虚拟机组成一个集群，在各个服务器之间进行通信，该集群是配置Kubernetes的所有租金啊啊、功能和负载的物理平台。

​ 一个Kubernetes集群由master和node组成。如下图：

* Master：是集群的网关和中枢枢纽，主要作用：暴露API接口，跟踪其他服务器的健康状态、以最优方式调度负载，以及编排其他组件之间的通信。单个的Master节点可以完成所有的功能，但是考虑单点故障的痛点，生产环境中通常要部署多个Master节点，组成Cluster。
* Node：是Kubernetes的工作节点，负责接收来自Master的工作指令，并根据指令相应地创建和销毁Pod对象，以及调整网络规则进行合理路由和流量转发。生产环境中，Node节点可以有N个。



Kubernetes从宏观上看分为2个角色：Master和Node，但是在Master节点和Node节点上都存在着多个组件来支持内部的业务逻辑，其包括：运行应用、应用编排、服务暴露、应用恢复等等，在Kubernetes中这些概念被抽象为Pod、Service、Controller等资源类型。先来了解一下这些常用概念和术语：

* （1）Pod

从上图，我们可以看到K8S并不直接地运行容器，而是被一个抽象的资源对象--Pod所封装，它是K8S最小的调度单位。这里要注意的是，Pod可以封装一个活多个容器！同一个Pod中共享网络名称空间和存储资源，而容器之间可以通过本地回环接口：lo 直接通信，但是彼此之间又在Mount、User和Pid等名称空间上保持了隔离。

* （2）资源标签（Label）

标签（Label）是将资源进行分类的标识符，就好像超市的商品分类一般。资源标签具体化的就是一个键值型（key/values)数据，相信了解redis的友友应该知道什么是键值数据。使用标签是为了对指定对象进行辨识，比如Pod对象。标签可以在对象创建时进行附加，也可以创建后进行添加或修改。要知道的是一个对象可以有多个标签，一个标签页可以附加到多个对象。如图：

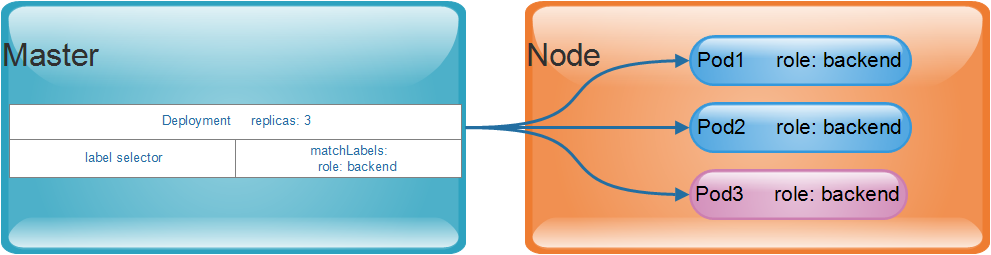
* （3）标签选择器（Selector）

有标签，当然就有标签选择器，它是根据Label进行过滤符合条件的资源对象的一种 机制。比如将含有标签role: backend的所有Pod对象挑选出来归并为一组。通常在使用过程中，会通过标签对资源对象进行分类，然后再通过标签选择器进行筛选，最常见的应用就是讲一组这样的Pod资源对象创建为某个Service的端点。如图：

* （4）Pod控制器（Controller）

虽然Pod是K8S的最小调度单位，但是K8S并不会直接地部署和管理Pod对象，而是要借助于另外一个抽象资源--Controller进行管理。其实一种管理Pod生命周期的资源抽象，并且它是一类对象，并非单个的资源对象，其中包括：ReplicationController、ReplicaSet、Deployment、StatefulSet、Job等。

以Deployment为例，它负责确保定义的Pod对象的副本数量符合预期的设置，这样用户只需要声明应用的期望状态，控制器就会自动地对其进行管理。如图：



* （5）服务资源（Service）

Service是建立在一组Pod对象之上的资源对象，在前面提过，它是通过标签选择器选择一组Pod对象，并为这组Pod对象定义一个统一的固定访问入口（通常是一个IP地址），如果K8S存在DNS附件（如coredns）它就会在Service创建时为它自动配置一个DNS名称，用于客户端进行服务发现。

通常我们直接请求Service IP，该请求就会被负载均衡到后端的端点，即各个Pod对象，从这点上，是不是有点像负载均衡器呢，因此Service本质上是一个4层的代理服务，另外Service还可以将集群外部流量引入至集群，这就需要节点对Service的端口进行映射了。

* （6）存储卷（Volume）

在使用容器时，我们知道，当数据存放于容器之中，容器销毁后，数据也会随之丢失。这就是需要一个外部存储，以保证数据的持久化存储。而存储卷就是这样的一个东西。

存储卷（Volume）是独立于容器文件系统之外的存储空间，常用于扩展容器的存储空间并为其提供持久存储能力。存储卷在K8S中的分类为：临时卷、本地卷和网络卷。临时卷和本地卷都位于Node本地，一旦Pod被调度至其他Node节点，此类型的存储卷将无法被访问，因为临时卷和本地卷通常用于数据缓存，持久化的数据通常放置于持久卷（persistent volume）之中。

* （7）Name和Namespace

名称（Name）是K8S集群中资源对象的标识符，通常作用于名称空间（Namespace），因此名称空间是名称的额外的限定机制。在同一个名称空间中，同一类型资源对象的名称必须具有唯一性。

名称空间通常用于实现租户或项目的资源隔离，从而形成逻辑分组。关于此概念可以参考：<https://www.jb51.net/article/136411.htm>

如图：创建的Pod和Service等资源对象都属于名称空间级别，未指定时，都属于默认的名称空间default

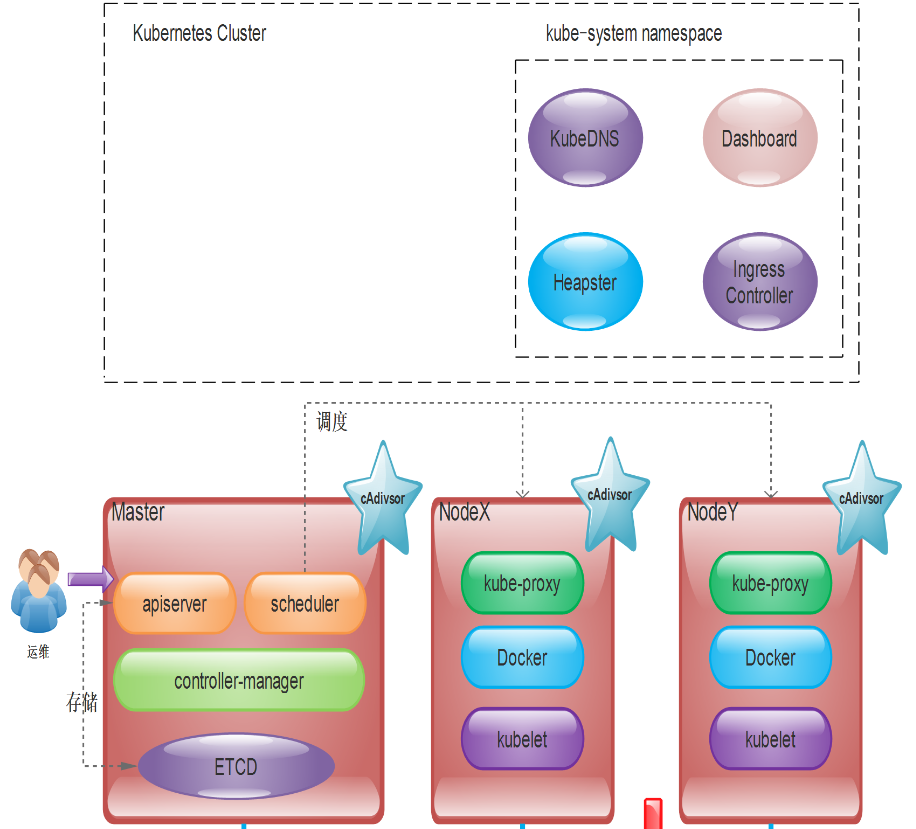
* （8）注解（Annotation）

Annotation是另一种附加在对象上的一种键值类型的数据，常用于将各种非标识型元数据（metadata）附加到对象上，但它并不能用于标识和选择对象。其作用是方便工具或用户阅读及查找。

* （9）Ingress

K8S将Pod对象和外部的网络环境进行了隔离，Pod和Service等对象之间的通信需要通过内部的专用地址进行，如果需要将某些Pod对象提供给外部用户访问，则需要给这些Pod对象打开一个端口进行引入外部流量，除了Service以外，Ingress也是实现提供外部访问的一种方式。

# 三、K8S集群组件



## 1、Master组件

* **1、API Server**

K8S对外的唯一接口，提供HTTP/HTTPS RESTful API，即kubernetes API。所有的请求都需要经过这个接口进行通信。主要负责接收、校验并响应所有的REST请求，结果状态被持久存储在etcd当中，所有资源增删改查的唯一入口。

* **2、etcd**

负责保存k8s 集群的配置信息和各种资源的状态信息，当数据发生变化时，etcd会快速地通知k8s相关组件。etcd是一个独立的服务组件，并不隶属于K8S集群。生产环境当中etcd应该以集群方式运行，以确保服务的可用性。

etcd不仅仅用于提供键值数据存储，而且还为其提供了监听（watch）机制，用于监听和推送变更。在K8S集群系统中，etcd的键值发生变化会通知倒API Server，并由其通过watch API向客户端输出。

* **3、Controller Manager**

负责管理集群各种资源，保证资源处于预期的状态。Controller Manager由多种controller组成，包括replication controller、endpoints controller、namespace controller、serviceaccounts controller等 。由控制器完成的主要功能主要包括生命周期功能和API业务逻辑，具体如下：

* 生命周期功能：包括Namespace创建和生命周期、Event垃圾回收、Pod终止相关的垃圾回收、级联垃圾回收及Node垃圾回收等。
* API业务逻辑：例如，由ReplicaSet执行的Pod扩展等。
* **4、调度器（Schedule）**

资源调度，负责决定将Pod放到哪个Node上运行。Scheduler在调度时会对集群的结构进行分析，当前各个节点的负载，以及应用对高可用、性能等方面的需求。

## 2、Node组件

Node主要负责提供容器的各种依赖环境，并接受Master管理。每个Node有以下几个组件构成。

* **1、Kubelet**

kubelet是node的agent，当Scheduler确定在某个Node上运行Pod后，会将Pod的具体配置信息（image、volume等）发送给该节点的kubelet，kubelet会根据这些信息创建和运行容器，并向master报告运行状态。

* **2、Container Runtime**

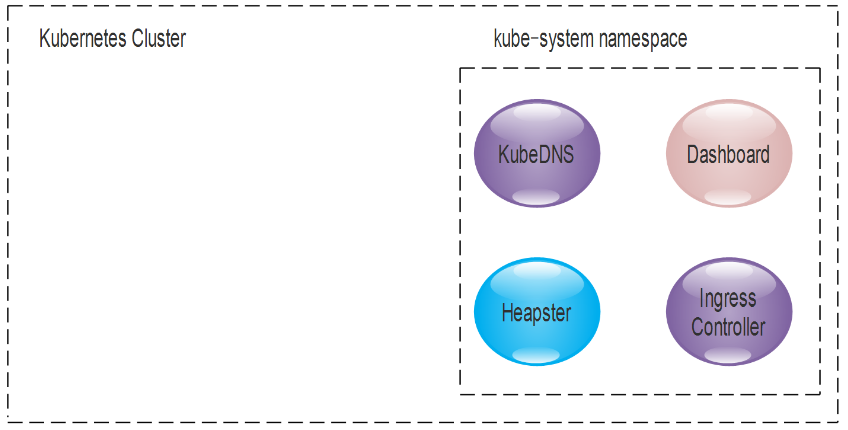
每个Node都需要提供一个容器运行时（Container Runtime）环境，它负责下载镜像并运行容器。目前K8S支持的容器运行环境至少包括Docker、RKT、cri-o、Fraki等。

* **3、Kube-proxy**

service在逻辑上代表了后端的多个Pod，外借通过service访问Pod。service接收到请求就需要kube-proxy完成转发到Pod的。每个Node都会运行kube-proxy服务，负责将访问的service的TCP/UDP数据流转发到后端的容器，如果有多个副本，kube-proxy会实现负载均衡，有2种方式：LVS或者Iptables

## 3、核心附件

K8S集群还依赖一组附件组件，通常是由第三方提供的特定应用程序。如下图：



* **1、KubeDNS**

在K8S集群中调度并运行提供DNS服务的Pod，同一集群内的其他Pod可以使用该DNS服务来解决主机名。K8S自1.11版本开始默认使用CoreDNS项目来为集群提供服务注册和服务发现的动态名称解析服务。

* **2、Dashboard**

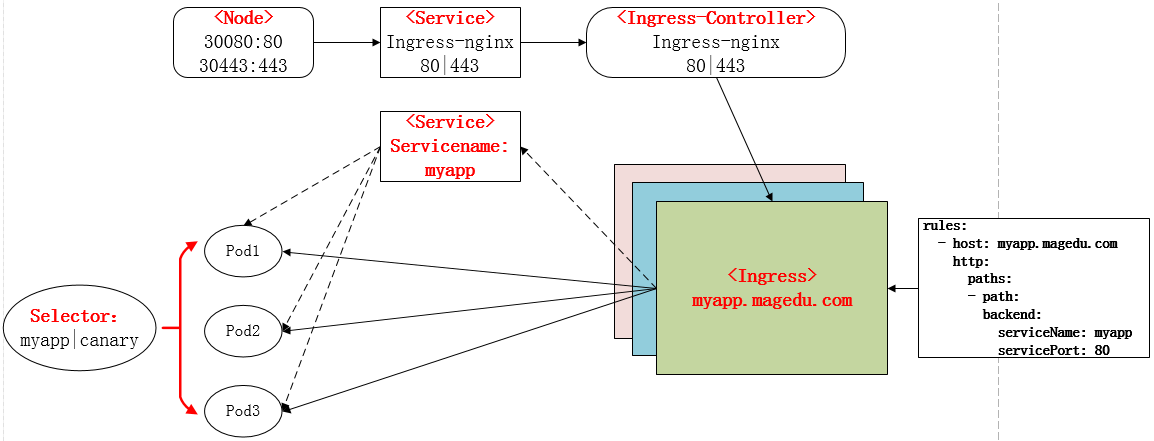
K8S集群的全部功能都要基于Web的UI，来管理集群中的应用和集群自身。

* **3、Heapster**

容器和节点的性能监控与分析系统，它收集并解析多种指标数据，如资源利用率、生命周期时间，在最新的版本当中，其主要功能逐渐由Prometheus结合其他的组件进行代替。

* **4、Ingress Controller**

Service是一种工作于4层的负载均衡器，而Ingress是在应用层实现的HTTP(S)的负载均衡。不过，Ingress资源自身并不能进行流量的穿透，，它仅仅是一组路由规则的集合，这些规则需要通过Ingress控制器（Ingress Controller）发挥作用。目前该功能项目大概有：Nginx-ingress、Traefik、Envoy和HAproxy等。如下图就是Nginx-ingress的应用，具体可以查看博文：<https://www.cnblogs.com/linuxk/p/9706720.html>

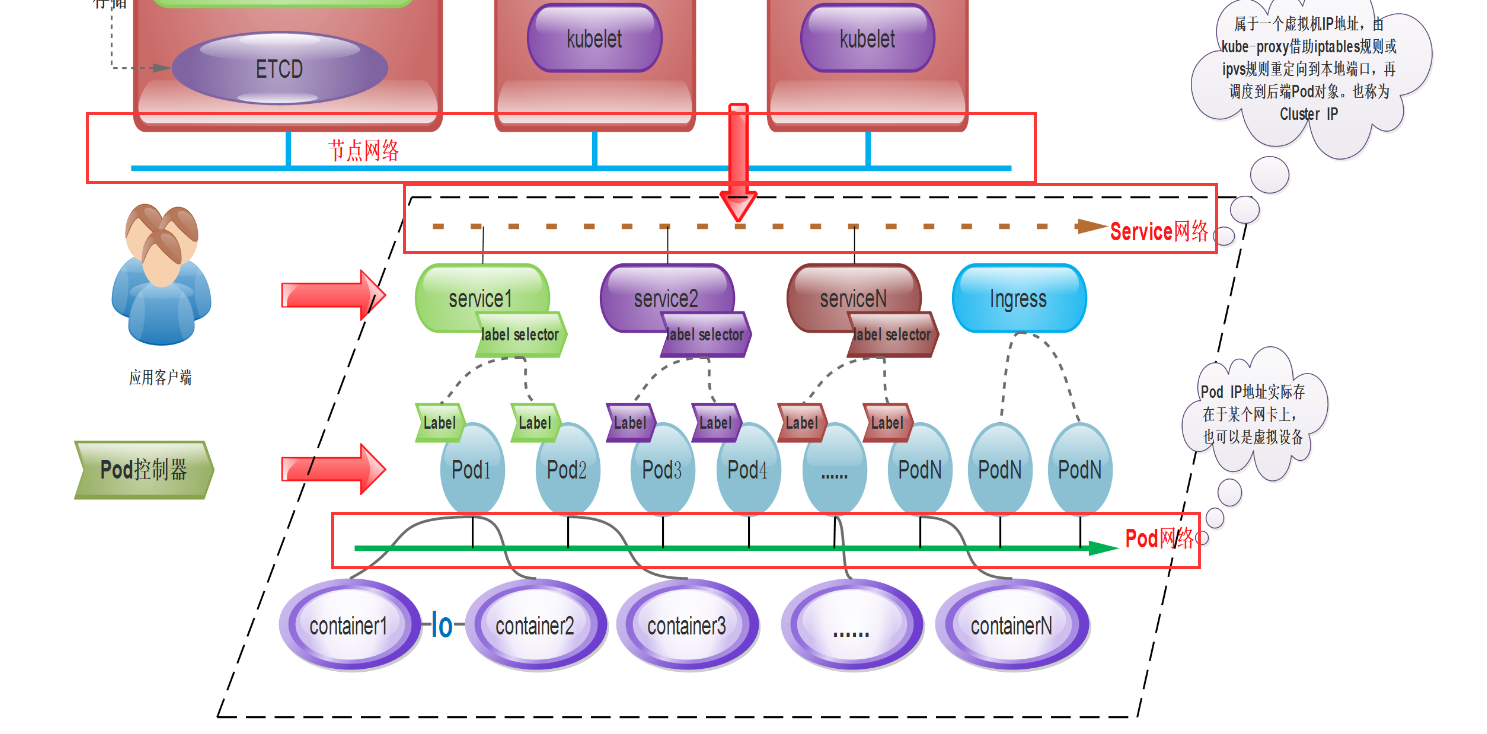


# 四、K8S的网络模型

K8S的网络中主要存在4种类型的通信：

* ①同一Pod内的容器间通信
* ②各个Pod彼此间的通信
* ③Pod和Service间的通信
* ④集群外部流量和Service之间的通信

K8S为Pod和Service资源对象分别使用了各自的专有网络，Pod网络由K8S的网络插件配置实现，而Service网络则由K8S集群进行指定。如下图：



K8S使用的网络插件需要为每个Pod配置至少一个特定的地址，即Pod IP。Pod IP地址实际存在于某个网卡（可以是虚拟机设备）上。

而Service的地址却是一个虚拟IP地址，没有任何网络接口配置在此地址上，它由Kube-proxy借助iptables规则或ipvs规则重定向到本地端口，再将其调度到后端的Pod对象。Service的IP地址是集群提供服务的接口，也称为Cluster IP。

Pod网络和IP由K8S的网络插件负责配置和管理，具体使用的网络地址可以在管理配置网络插件时进行指定，如10.244.0.0/16网络。而Cluster网络和IP是由K8S集群负责配置和管理，如10.96.0.0/12网络。

从上图进行总结起来，一个K8S集群包含是三个网络。

* **（1）节点网络：**各主机（Master、Node、ETCD等）自身所属的网络，地址配置在主机的网络接口，用于各主机之间的通信，又称为节点网络。
* **（2）Pod网络：**专用于Pod资源对象的网络，它是一个虚拟网络，用于为各Pod对象设定IP地址等网络参数，其地址配置在Pod中容器的网络接口上。Pod网络需要借助kubenet插件或CNI插件实现。
* **（3）Service网络：**专用于Service资源对象的网络，它也是一个虚拟网络，用于为K8S集群之中的Service配置IP地址，但是该地址不会配置在任何主机或容器的网络接口上，而是通过Node上的kube-proxy配置为iptables或ipvs规则，从而将发往该地址的所有流量调度到后端的各Pod对象之上。

# 五、Kubernetes的核心对象详解

## 1、Pod资源对象

​ Pod资源对象是一种集合了一个或多个应用容器、存储资源、专用ip、以及支撑运行的其他选项的逻辑组件。如下图：Pod其实就是一个应用程序运行的单一实例，它通常由共享资源且关系紧密的一个或2多个应用容器组成。

​ Kubernetes的网络模型要求每个Pod的IP地址同一IP网段，各个Pod之间可以使用IP地址进行通信，无论这些Pod运行在集群内的哪个节点上，这些Pod对象都类似于运行在同一个局域网内的虚拟机一般。

​ 我们可以将每一个Pod对象类比为一个物理主机或者是虚拟机，那么运行在同一个Pod对象中的多个进程，也就类似于跑在物理主机上的独立进程，而不同的是Pod对象中的各个进程都运行在彼此隔离的容器当中，而各个容器之间共享两种关键性资源：网络和存储卷。

* 网络：每一个Pod对象都会分配到一个Pod IP地址，同一个Pod内部的所有容器共享Pod对象的Network和UTS名称空间，其中包括主机名、IP地址和端口等。因此，这些容器可以通过本地的回环接口lo进行通信，而在Pod之外的其他组件的通信，则需要使用Service资源对象的Cluster IP+端口完成。
* 存储卷：用户可以给Pod对象配置一组存储卷资源，这些资源可以共享给同一个Pod中的所有容器使用，从而完成容器间的数据共享。存储卷还可以确保在容器终止后被重启，或者是被删除后也能确保数据的持久化存储。

​ 一个Pod代表着某个应用程序的特定实例，如果我们需要去扩展这个应用程序，那么就意味着需要为该应用程序同时创建多个Pod实例，每个实例都代表着应用程序的一个运行副本。而这些副本化的Pod对象的创建和管理，都是由一组称为Controller的对象实现，比如Deployment控制器对象。

​ 当创建Pod时，我们还可以使用Pod Preset对象为Pod注入特定的信息，比如Configmap、Secret、存储卷、卷挂载、环境变量等。有了Pod Preset对象，Pod模板的创建就不需要为每个模板显示提供所有信息。

​ 基于预定的期望状态和各个节点的资源可用性，Master会把Pod对象调度至选定的工作节点上运行，工作节点从指向的镜像仓库进行下载镜像，并在本地的容器运行时环境中启动容器。Master会将整个集群的状态保存在etcd中，并通过API Server共享给集群的各个组件和客户端。

## 2、Controller

​ 在K8S的集群设计中，Pod是一个有生命周期的对象。那么用户通过手工创建或者通过Controller直接创建的Pod对象会被调度器（Scheduler）调度到集群中的某个工作节点上运行，等到容器应用进程运行结束之后正常终止，随后就会被删除。而需要注意的是，当节点的资源耗尽或者故障，也会导致Pod对象的回收。

​ 而K8S在这一设计上，使用了控制器实现对一次性的Pod对象进行管理操作。比如，要确保部署的应用程序的Pod副本数达到用户预期的数目，以及基于Pod模板来重建Pod对象等，从而实现Pod对象的扩容、缩容、滚动更新和自愈能力。例如，在某个节点故障，相关的控制器会将运行在该节点上的Pod对象重新调度到其他节点上进行重建。

​ 控制器本身也是一种资源类型，其中包括Replication、Controller、Deployment、StatefulSet、DaemonSet、Jobs等等，它们都统称为Pod控制器。如下图的Deployment就是这类控制器的代表实现，是目前用于管理无状态应用的Pod控制器。

​ Pod控制器的定义通常由期望的副本数量、Pod模板、标签选择器组成。Pod控制器会根据标签选择器来对Pod对象的标签进行匹配筛选，所有满足选择条件的Pod对象都会被当前控制器进行管理并计入副本总数，确保数目能够达到预期的状态副本数。

​ 需要注意的是，在实际的应用场景中，在接收到的请求流量负载低于或接近当前已有Pod副本的承载能力时，需要我们手动修改Pod控制器中的期望副本数量以实现应用规模的扩容和缩容。而在集群中部署了HeapSet或者Prometheus的这一类资源监控组件时，用户还可以通过HPA（HorizontalPodAutoscaler）来计算出合适的Pod副本数量，并自动地修改Pod控制器中期望的副本数，从而实现应用规模的动态伸缩，提高集群资源的利用率。

​ K8S集群中的每个节点上都运行着cAdvisor，用于收集容器和节点的CPU、内存以及磁盘资源的利用率直播数据，这些统计数据由Heapster聚合之后可以通过API server访问。而HorizontalPodAutoscaler基于这些统计数据监控容器的健康状态并作出扩展决策。

## 3、Service

​ 我们知道Pod对象有Pod IP地址，但是该地址无法确保Pod对象重启或者重建之后保持不变，这会带来集群中Pod应用间依赖关系维护的麻烦。比如前段Pod应用无法基于固定的IP地址跟中后端的Pod应用。

​ 而Service资源就是在被访问的Pod对象中添加一个有着固定IP地址的中间层，客户端向该地址发起访问请求后，由相关的Service资源进行调度并代理到后端的Pod对象。

​ Service并不是一个具体的组件，而是一个通过规则定义出由多个Pod对象组成而成的逻辑集合，并附带着访问这组Pod对象的策略。Service对象挑选和关联Pod对象的方式和Pod控制器是一样的，都是通过标签选择器进行定义。如下图：

​ Service IP是一种虚拟IP，也称为Cluster IP，专用于集群内通信，通常使用专有的地址段，如：10.96.0.0/12网络，各Service对象的IP地址在该范围内由系统动态分配。

​ 集群内的Pod对象可直接请求这类Cluster IP，比如上图中来自Pod client的访问请求，可以通过Service的Cluster IP作为目标地址进行访问，但是在集群网络中是属于私有的网络地址，仅仅可以在集群内部访问。

​ 而需要将集群外部的访问引入集群内部的常用方法是通过节点网络进行，其实现方法如下：

* 通过工作节点的IP地址+端口（Node Port）接入请求。
* 将该请求代理到相应的Service对象的Cluster IP的服务端口上，通俗地说：就是工作节点上的端口映射了Service的端口。
* 由Service对象将该请求转发到后端的Pod对象的Pod IP和 应用程序的监听端口。

​ 因此，类似于上图来自Exxternal Client的集群外部客户端，是无法直接请求该Service的Cluster IP，而是需要实现经过某一工作节点（如 Node Y）的IP地址，着了请求需要2次转发才能到目标Pod对象。这一类访问的缺点就是在通信效率上有一定的延时。